Cytotoxicity modelling in R

Isidro Cortes-Ciriano*[1], Georgios Drakakis[1], and Andreas Bender†[1]

[1] *Centre for Molecular Science Informatics, Department of Chemistry, University of Cambridge, Cambridge, United Kingdom.*

December 11, 2017

This protocol presents a pipeline to generates various models to model the sensitivity (cytotoxicity) of the cancer cell line DU−145 to small molecules using the R package `camb` Murrell et al. (2015). The data set used in this tutorial was downloaded from ChEMBL version 21 using the file `download_IC50_data.sql` provided with this tutorial. Further information and scripts on how to download cytotoxicity data from ChEMBL for other cell lines can be found in the Supplementary Information of reference Cortes-Ciriano & Bender (2016). Along with this tutorial, we provide all intermediate files generated by running it. To download the data, type in a bash terminal the following: mysql < download_IC50_data.sql > IC50_DU145.txt. ChEMBL database needs to be installed (https://www.ebi.ac.uk/chembl/faq).

To gather a high-quality data set, we only considered IC50 data in nM units and with an activity standard_relation equal to '='. The models generated in the following sections are trained on 2D molecular descriptors calculated by the PaDEL-Descriptor package as input features. These models are then ensembled to create a single model with higher predictive power Cortes-Ciriano, Murrell, van Westen, Bender & Malliavin (2015). The trained models are used to make predictions for new molecules. Finally, we illustrate how the conformal package Cortes-Ciriano (2015), Norinder et al. (2014), Cortés-Ciriano, Bender & Malliavin (2015*a*) can be used to generate confidence intervals for individual predictions. Firstly, the package `camb` is loaded and the working directory set:

```
library(camb)
path_to_working_directory <- "" # change to the appropriate value
setwd(path_to_working_directory)
```

```
print(sessionInfo())
```

```
## R version 3.3.2 (2016-10-31)
## Platform: x86_64-apple-darwin13.4.0 (64-bit)
## Running under: macOS Sierra 10.12.6
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
##  [1] grid      parallel  splines   stats     graphics  grDevices utils
```

*isidrolauscher@gmail.com
†ab454@cam.ac.uk

```
##  [8] datasets  methods   base
##
## other attached packages:
##  [1] doMC_1.3.4              iterators_1.0.8
##  [3] conformal_0.3           e1071_1.6-8
##  [5] camb_1.0                directlabels_2017.03.31
##  [7] maptools_0.9-2          sp_1.2-4
##  [9] protr_1.4-0             vegan_2.4-3
## [11] permute_0.9-4           gridExtra_2.2.1
## [13] gbm_2.1.3               survival_2.41-2
## [15] kernlab_0.9-25          caret_5.17-7
## [17] reshape2_1.4.2          plyr_1.8.4
## [19] lattice_0.20-34         foreach_1.4.3
## [21] cluster_2.0.5           caTools_1.17.1
## [23] pbapply_1.3-2           ggplot2_2.2.1
## [25] knitr_1.15.1
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_0.12.13        highr_0.6           class_7.3-14
##  [4] bitops_1.0-6        tools_3.3.2         evaluate_0.10
##  [7] tibble_1.3.4        gtable_0.2.0        nlme_3.1-128
## [10] mgcv_1.8-15         rlang_0.1.4         Matrix_1.2-7.1
## [13] stringr_1.2.0       foreign_0.8-67      magrittr_1.5
## [16] scales_0.4.1        codetools_0.2-15    MASS_7.3-45
## [19] randomForest_4.6-12 colorspace_1.3-2    quadprog_1.5-5
## [22] stringi_1.1.3       lazyeval_0.2.0      munsell_0.4.3
```

1 Normalization of chemical structures and descriptor calculation

We will firstly read the cytotoxicity data extracted from ChEMBL 21. Given that some SMILES strings contain smarts patterns where the hash symbol is present, it is necessary to switch off the argument *comment.char* in order not to clip them. We next calculate the mean IC50 value for compounds annotated with multiple bioactivity values. For the effect of using heterogeneous cytotoxicity data (i.e. generated by multiple laboratories) for the modelling of cell line sensitivity data please see ref Cortes-Ciriano & Bender (2016).

```
d <- read.csv("IC50_DU-145.txt",header=T,sep="\t",comment.char = "")
library(plyr)
d <- ddply(d,.(chembl_id,canonical_smiles),summarize,
           mean_value=mean(as.vector(as.numeric(standard_value))))
print(head(d))
```

1.1 Reading and preprocessing

We next proceed to standardize/normalized the compounds using the Indigo's C API, on which the `camb` package relies to accomplish this step. Molecules are represented with implicit hydrogens, dearomatized, and passed through the InChI format to ensure that tautomers are represented by the same SMILES strings.

The `StandardiseMolecules` function allows representation of the molecular structures in a similarly processed form. The arguments of this function allow control over the maximum number of (i) fluorines, (ii) chlorines, (iii) bromines, and (iv) iodines the molecules can contain in order to be retained for training. Inorgnaic molecules (those containing atoms not in {H, C, N, O, P, S, F, Cl, Br, I}) are removed if the argument `remove.inorganic` is set to `TRUE` (default). The upper and lower limits for molecular mass can be set with the arguments `min.mass.limit` and `max.mass.limit`. The name of the file containing the chemical structures is provided through the argument `structures.file`.

We first save the molecule structures in SMILES format to a .smi file, and then standardize the molecules:

```
write.table(d$canonical_smiles, col.names = F,
    row.names = F, quote = F, file = "DU145_toxicity_smiles.smi")

std.options <- StandardiseMolecules(structures.file = "DU145_toxicity_smiles.smi",
    standardised.file = "standardised.sdf",
    removed.file = "removed.sdf", properties.file = "standardization_info.csv",
    remove.inorganic = TRUE, fluorine.limit = 3,
    chlorine.limit = 3, bromine.limit = 3,
    iodine.limit = 3, min.mass.limit = 20,
    max.mass.limit = 900)
saveRDS(std.options, "standardisation_options.rds")
```

Molecules that Indigo manages to parse and that pass the filters are written to the file indicated by the `standardised.file` argument once they have gone through the standardisation procedure. Molecules that were discarded are written to the file indicated by the `removed.file` argument. The molecule name and molecular properties specified in the structure file are written to the CSV file indicated in the argument `properties.file`. The column `kept` indicates which molecules were deleted (0) or kept (1).

```
properties <- read.table("standardization_info.csv", header = TRUE, sep = "\t")
d <- d[properties$Kept == 1, ]
```

1.2 Circular Morgan fingerprints

To calculate circular Morgan fingerprints, `camb` requires the python library RDkit, as the function "MorganFPs" calls a python script for their calculation. The python code is available in the "extdata" folder of the package `camb` or at:
https://github.com/isidroc/FingerprintCalculator.
For a detailed discussion about circular Morgan fingerprints, we refer the interested reader to ref. Rogers & Hahn (2010). 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, these variables can be passed as arguments to the function MorganFPs: PythonPath,path to python in the system, and RDkitPath, path to the RDkit library. For instance, the information of the latter is contained in the environment variable `$RDBASE` in Mac OS. Although we will not use this descriptor type to train the models here, we illustrate how they can be computed. These fingerprints are widely used in the modelling community, as they have proved highly efficient in virtual screening campaigns Bender et al. (2009).

```
Sys.setenv(RDBASE="/usr/local/share/RDKit")
Sys.setenv(PYTHONPATH="/usr/local/lib/python2.7/site-packages")

fps_DU145_512 <- MorganFPs(bits=256,radii=c(0,1,2,3),mols='standardised.sdf',
```

```
                          output='DU145',keep='hashed_counts',
                          RDkitPath='/usr/local/share/RDKit',
                          PythonPath='/usr/local/lib/python2.7/site-packages',
                          images = FALSE, unhashed = FALSE,
                          extMols = FALSE, unhashedExt = FALSE,
                          logFile = FALSE)

#saveRDS(fps_DU145_512,file="fps_DU145_512.rds")
#fps_DU145_512 <- readRDS("fps_DU145_512.rds")
```

1.3 PaDEL descriptors

The following piece of code serves to compute one- and two-dimensional physicochemical descriptors using the function `GeneratePadelDescriptors`, which relies on the Java library PaDEL Yap (2011).

```
descriptor.types <- c("2D")
padel_descs <- GeneratePadelDescriptors(standardised.file = "standardised.sdf",
    threads = 12, types = c("2D"))
saveRDS(padel_descs, file = "padel_descs.rds")
padel_descs <- readRDS("padel_descs.rds")

mol_idx <- as.numeric(unlist(sapply(as.vector(padel_descs$Name),
    function(x) {
        unlist(strsplit(x, "\\_"))[2]
    })))
padel_descs <- padel_descs[order(mol_idx, decreasing = F),
    ]
sum(is.na(padel_descs))
padel_descs <- padel_descs[, 2:ncol(padel_descs)]
padel_descs <- ReplaceInfinitesWithNA(padel_descs)
library(impute)
padel_descs <- ImputeFeatures(padel_descs)
sum(is.na(padel_descs))
saveRDS(padel_descs, file = "padel_descs.rds")
```

1.4 Preparation of the bioactivity values

Next, we transform the IC50 values extracted in nM units from ChEMBL to pIC50 values. To do that, we multiply by `10^-9` to convert the bioactivity units to M. Subsequently, the negative logarithm to base 10 is calculated. We remove extreme values by keeping only cases with pIC50 values in the 4-10 range.

```
bioactivity <- d$mean_value
bioactivity <- bioactivity * 10^-9
bioactivity <- -log(bioactivity, base = 10)
idx = which(bioactivity > 10 | bioactivity < 4 | is.infinite(bioactivity))
if (length(idx) > 0) {
    bioactivity <- bioactivity[-idx]
    d <- d[-idx, ]
```

```
    padel_descs <- padel_descs[-idx, ]
}
# check that the dimension of the descriptors, IC50
# data information and IC50 values is the same:
print(dim(d))
```

```
## [1] 2378    3
```

```
print(dim(padel_descs))
```

```
## [1] 2378  729
```

```
print(length(bioactivity))
```

```
## [1] 2378
```

2 Statistical pre-processing

Next, we remove descriptors carrying little or no predictive signal. These descriptors have comparable values across the training examples, and, thus, display a variance close to zero (near-zero variance). Descriptors are further converted to z-scores by centering to zero mean and unit variance.

```
dataset = SplitSet(1:nrow(padel_descs), padel_descs,
    bioactivity, percentage = 20, seed = 1)
dataset <- RemoveNearZeroVarianceFeatures(dataset,
    frequencyCutoff = 30)
```

```
## 333 features removed with variance below cutoff
```

```
dataset <- RemoveHighlyCorrelatedFeatures(dataset,
    correlationCutoff = 0.95)
```

```
## 137 features removed with correlation above cutoff
```

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

All models are trained with the same CV options, *i.e.* the arguments of the function `GetCVTrainControl`. This is necessary because the procedure that will be used later to generate model ensembles requires the same data partition (i.e., fold composition) over all methods used. It is important to note that the functions presented in the previous code blocks depend on functions from the `caret` package, namely:

- RemoveNearZeroVarianceFeatures : nearZeroVar

- RemoveHighlyCorrelatedFeatures : findCorrelation
- PreProcess : preProcess
- GetCVTrainControl : trainControl

Experienced users might want to have more control over the underlying `caret` functions. In fact, experienced users may want to learn from the internals of the functions `camb` provides and create their own specialised pipeline that fits their own modelling needs. `camb` is intended to speed up the modelling process and should not limit use of the extremely valuble `caret` package which `camb` utilises. The default values of these function permit the less experienced user to quickly handle the statistical preprocessing steps with ease, making a reasonable default choice for the argument values for less experienced users.

3 Model training

In the following section we present the different steps required to build predictive models using `camb`. It should be noted that the above steps can be run locally on a low powered computer, such as a laptop, and the preprocessed dataset saved to disk. This dataset can then be copied to a high powered machine or a farm with multiple cores for model training and the resulting models saved back to the local machine.
We will use grid search and cross validation (CV) to estimate the most suitable parameter values. Once these are estimated, a model will be trained on the entire training set. This model will be subsequently applied on the test set (unseen data) to assess its predictive power on unseen molecules. In this tutorial we will use three widely used algorithms: Random Forest (RF), Gradient Booston Machines, and Support Vector Machines. Please see the documentation of the `caret` package for a full list of algorithms currently available.

```
# register the number of cores to use in training
library(doMC)
registerDoMC(cores = 6)

folds = 5
set.seed(123)
dataset <- GetCVTrainControl(dataset, seed = 1, folds = 5,
    returnResamp = "none", returnData = FALSE, savePredictions = TRUE,
    verboseIter = TRUE, allowParallel = TRUE)
saveRDS(dataset, file = "dataset_preprocessed.rda")
```

All models are trained with the same CV options, *i.e.* the arguments of the function `GetCVTrainControl`, to allow ensemble modeling as ensemble modelling requires the same data fold split over all methods used.

3.1 Random Forest (RF)

Firstly, an RF model Breiman (2001) is trained using the train function from the caret package.

```
library(randomForest)
method <- "rf"
set.seed(123)
model <- train(dataset$x.train, dataset$y.train,
    method, trControl = dataset$trControl)
saveRDS(model, file = paste(method, ".rds",
```

```
    sep = ""))
model = readRDS(paste(method, ".rds", sep = ""))
print(RMSE_CV(model))
print(Rsquared_CV(model))
```

On the basis of the soundness of the obtained models, evaluated through the value of the cross-validated metrics using the functions RMSE_CV and Rsquared_CV, we apply it on the test set to assess its predictive power on unseen data:

```
holdout.predictions <- as.vector(predict(model$finalModel,
    newdata = dataset$x.holdout))
print(RMSE(holdout.predictions, dataset$y.holdout))


## [1] 0.7836013


print(Rsquared(holdout.predictions, dataset$y.holdout))


## [1] 0.5766363


print(cor(holdout.predictions, dataset$y.holdout))


## [1] 0.7593657
```

3.2 Support Vector Machines (SVM)

A base 2 exponential grid is used to optimize over the hyperparameters of the SVM Ben-Hur et al. (2008). The `train` function from the `caret` package is used directly for model training.

```
method = "svmRadial"
tune.grid <- expand.grid(.sigma = expGrid(-8, 4, 2,
    2), .C = c(1e-04, 0.001, 0.01, 0.1, 1, 10, 100))
set.seed(123)
model <- train(dataset$x.train, dataset$y.train, method,
    tuneGrid = tune.grid, trControl = dataset$trControl)
saveRDS(model, file = paste(method, ".rds", sep = ""))
model = readRDS(paste(method, ".rds", sep = ""))
print(RMSE_CV(model))
print(Rsquared_CV(model))
```

```
holdout.predictions <- as.vector(predict(model, newdata = dataset$x.holdout))
print(RMSE(holdout.predictions, dataset$y.holdout))


## [1] 0.8399737
```

```
print(Rsquared(holdout.predictions, dataset$y.holdout))
```

```
## [1] 0.4923828
```

```
print(cor(holdout.predictions, dataset$y.holdout))
```

```
## [1] 0.7017
```

3.3 Gradient Boosting Machine (GBM)

A GBM model Friedman (n.d.) is trained optimising over the number of trees and the parameters interaction depth and shrinkage. We set the value of the GBM parameter bag fraction to 0.1, as values in the 0.2-0.2 range have proved an optimal choice to reduce the effect of the noise in the data Cortes-Ciriano, Bender & Malliavin (2015*b*).

```
library(gbm)
method <- "gbm"
tune.grid <- expand.grid(.n.trees = c(500), .interaction.depth = c(3,
    4, 5, 6, 7, 8, 9, 10), .shrinkage = c(0.1, 0.05,
    0.15, 0.2))
set.seed(123)
model <- train(dataset$x.train, dataset$y.train, method,
    tuneGrid = tune.grid, trControl = dataset$trControl,
    bag.fraction = 0.1)
saveRDS(model, file = paste(method, ".rds", sep = ""))
model <- readRDS(file = paste(method, ".rds", sep = ""))
```

```
holdout.predictions <- as.vector(predict(model, newdata = dataset$x.holdout))
print(RMSE(holdout.predictions, dataset$y.holdout))
```

```
## [1] 0.8547598
```

```
print(Rsquared(holdout.predictions, dataset$y.holdout))
```

```
## [1] 0.4759572
```

```
print(cor(holdout.predictions, dataset$y.holdout))
```

```
## [1] 0.6898965
```

For each model we determine if our hyper-parameter search needs to be altered

If the hyper-parameters scanned lead to low errors in prediction (ideally close to the level of experimental uncertainty in the data), the parameter values can be considered to be reasonable (see Figures 1 and 2). Otherwise,

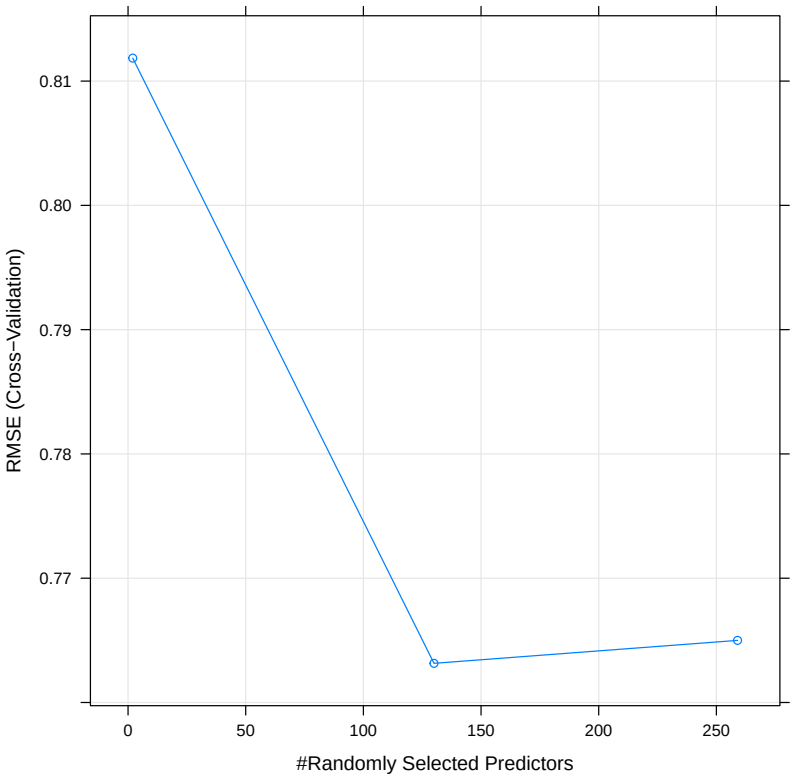

Figure 1: CV RMSE over the mtry hyperparameter for the RF model

the parameter grid needs to be extended and the cross-validation steps repeated.

```
model <- readRDS("rf.rds")
holdout.predictions <- as.vector(predict(model, newdata = dataset$x.holdout))
plot(model, metric = "RMSE")
```

```
model <- readRDS("gbm.rds")
holdout.predictions <- as.vector(predict(model, newdata = dataset$x.holdout))
plot(model, metric = "RMSE")
```

To visualize the correlation between predicted and observed values, we use the `CorrelationPlot` function:

```
CorrelationPlot(pred = holdout.predictions, obs = dataset$y.holdout,
    PointSize = 3, ColMargin = "blue", TitleSize = 10,
    XAxisSize = 10, YAxisSize = 10, TitleAxesSize = 10,
    margin = 2, PointColor = "black", PointShape = 16,
    MarginWidth = 1, AngleLab = 0, xlab = "Observed",
    ylab = "Predicted")
```

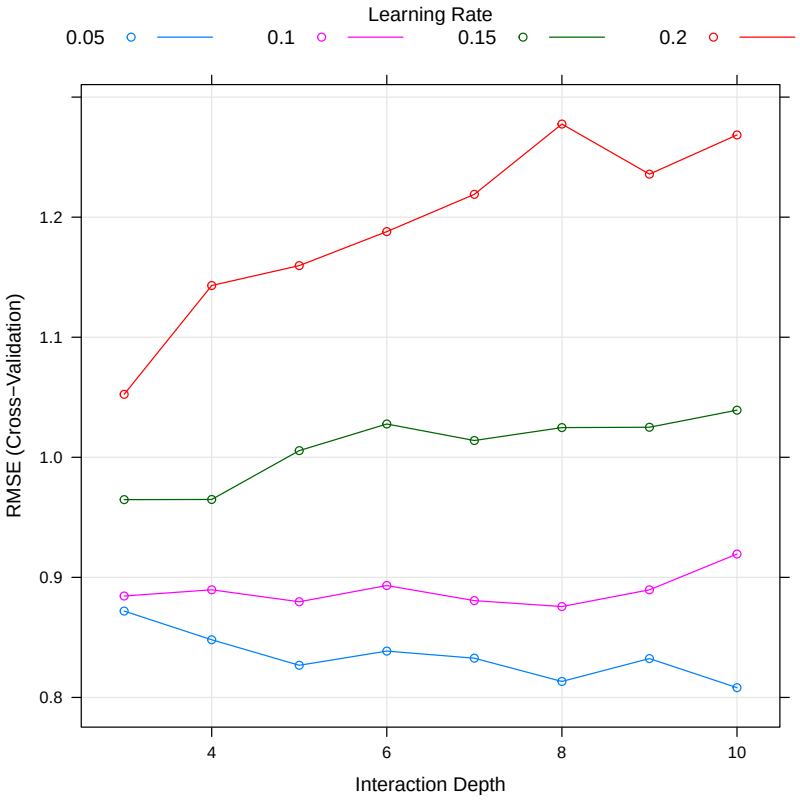

Figure 2: CV RMSE over the hyperparameters for the GBM model

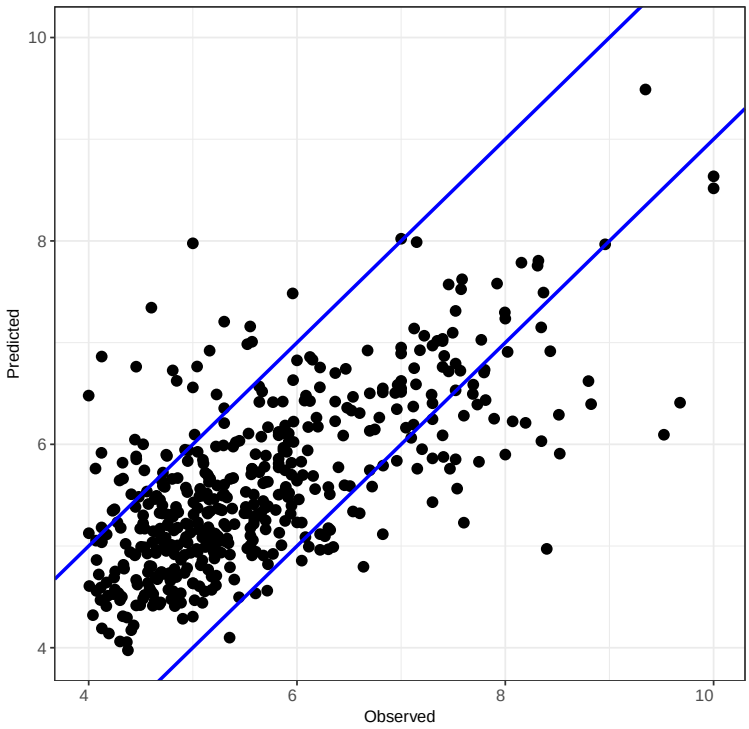

Figure 3: Observed vs Predicted

4 Ensemble modeling

In the following section, two ensemble modeling techniques are applied, namely greedy optimization and model stacking. Further information about these methods can be found in Mayer (2013), Caruana et al. (2004), Cortes-Ciriano, Murrell, van Westen, Bender & Malliavin (2015).

Firstly, we append all the trained models to a list. The `sort` function shows the cross-validation RMSEs of the trained models in ascending order.

```
all.models <- list()
all.models[[length(all.models) + 1]] <- readRDS("gbm.rds")
all.models[[length(all.models) + 1]] <- readRDS("rf.rds")
all.models[[length(all.models) + 1]] <- readRDS("svmRadial.rds")

# sort the models from lowest to highest RMSE
names(all.models) <- sapply(all.models, function(x) x$method)
all.models$gbm$control$savePredictions <- T
all.models$rf$control$savePredictions <- T
all.models$svmRadial$control$savePredictions <- T
```

A greedy ensemble is then trained using 1,000 iterations. The Greedy ensemble picks a linear combination of model outputs that is a local minimum in the RMSE landscape. The weights for each model can be seen in the `greedy$weights` variable. The RMSE of the greedy model can be found in the `greedy$error` variable.

```
greedy <- caretEnsemble(all.models, iter = 1000)
sort(greedy$weights, decreasing = TRUE)
saveRDS(greedy, file = "greedy.rds")
```

```
greedy$error
```

```
##      RMSE
## 0.7224215
```

Next, we create a linear stack ensemble that uses the trained model inputs as input into the stack.

```
linear <- caretStack(all.models, method = "glm", trControl = trainControl(method = "cv"))
saveRDS(linear, file = "linear.rds")
```

```
linear$error
```

```
##   parameter      RMSE  Rsquared     RMSESD RsquaredSD
## 1      none 0.7180705 0.6218412 0.04739973 0.06602659
```

We also create a non-linear stack ensemble using a RF model. Other algorithms could be used for this task. For a detailed discussion on ensemble modelling in bioactivity prediction, please see Cortes-Ciriano, Murrell, van Westen, Bender & Malliavin (2015).

```
tune.grid <- expand.grid(.mtry = seq(1, length(all.models),
    1))
nonlinear <- caretStack(all.models, method = "rf",
    trControl = trainControl(method = "cv"), tune.grid = tune.grid)
saveRDS(nonlinear, file = "nonlinear.rds")
```

```
nonlinear$error
```

```
##   mtry      RMSE  Rsquared     RMSESD RsquaredSD
## 1    2 0.7467472 0.5957130 0.05211962 0.06231944
## 2    3 0.7489750 0.5932807 0.05110615 0.06296513
```

The greedy and the linear stack ensembles have cross validated RMSE values that are lower than any of the individual models. We then test to see if these ensemble models outperform the individual models on the holdout set.

```
preds <- data.frame(sapply(all.models, predict, newdata = dataset$x.holdout))
preds$ENS_greedy <- predict(greedy, newdata = dataset$x.holdout)
preds$ENS_linear <- predict(linear, newdata = dataset$x.holdout)
preds$ENS_nonlinear <- predict(nonlinear, newdata = dataset$x.holdout)
sort(sqrt(colMeans((preds - dataset$y.holdout)^2)))
```

```
##    ENS_linear            rf    ENS_greedy ENS_nonlinear     svmRadial
##     0.7689456     0.7836013     0.7859444     0.7968667     0.8399737
##           gbm
##     0.8547598
```

```
# R02
apply(preds, 2, function(x) {
    Rsquared0(x, dataset$y.holdout)
})
```

```
##           gbm            rf     svmRadial    ENS_greedy    ENS_linear
##     0.4738336     0.5599681     0.4917497     0.5569973     0.5751169
## ENS_nonlinear
##     0.5455251
```

The linear and greedy ensembles slightly outperform other models on the holdout set as well. This leads us to choose this as the most predictive model for future predictions. In the case that the ensemble models underperform the single models on the holdout set by a slight margin, it is advisable to pick the best single model for future predictions as a simpler model is preferable (i.e. principle of parsimony). We note that the performance of these models is in line with previous estimates of the maximum model performance achievable given the uncertainty of cell line sensitivity data. For a detailed discussion on this topic see Cortes-Ciriano & Bender (2016).

5 Conformal prediction

In this section, we are going to illustrate how to generate confidence intervals for individual predictions using the conformal prediction framework. Please see the documentation of the R package conformal for an in-depth description of the underlying theory and implementation Cortes-Ciriano (2015).

```
require(conformal)
model <- readRDS("rf.rds")
# Train an error model
error_model <- ErrorModel(PointPredictionModel = model,
    x.train = dataset$x.train, savePredictions = TRUE,
    algorithm = "rf", trControl = dataset$trControl)
```

Once the error model is trained, we can apply conformal prediction in the following manner:

```
# Instantiate the class and get the confidence
# intervals
example <- ConformalRegression$new()


## Conformal Prediction Class for Regression Instantiated


example$CalculateAlphas(model = model, error_model = error_model,
    ConformityMeasure = StandardMeasure)


## [1] "Calculating alphas.."


example$confidence <- 0.8
# consider only 20 examples to avoid a cluttered
# plot
example$GetConfidenceIntervals(new.data = dataset$x.holdout[1:20,
    ])


## [1] "Predicting (i) the value, and (ii) the error for the new data.."


example$CorrelationPlot(obs = dataset$y.holdout[1:20])


## [1] "This method generates a correlation plot for the observed against the predicted values for a set of
```

Next, we plot the predicted agains the experimental IC50 values for the test set along with the predicted confidence intervals.

```
example$plot
```

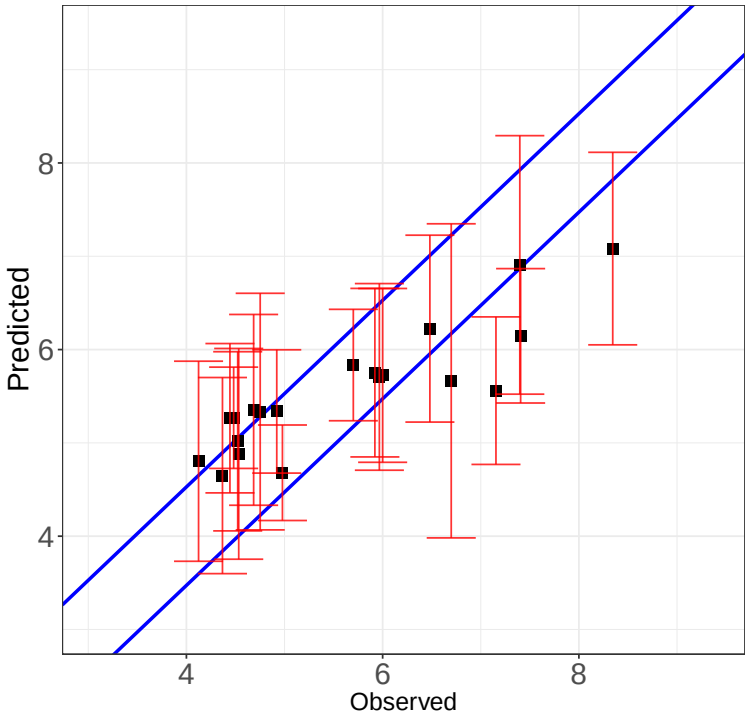

Figure 4: Observed vs predicted. Confidence intervals calculated with conformal prediction at a confidence level of 0.80

6 External predictions

One of the main advantages of `camb` is that it permits to standardize, process and make predictions for external molecules with little coding effort. In predictive modelling it is paramount to ensure that the same standardisation options and descriptor types are used when a model is applied to make predictions for new molecules. The argument descriptor.types indicates the type of PaDEL descriptors used to trained the models. The argument standardisation.options corresponds to the standardisation options used when normalizing the compounds encompassed in the training data. These two arguments guarantee that both the training data and novel molecules are processed uniformly. The function PredictExternal can be used for this task in the following way:

```
test_structures_file <- system.file("test_structures",
    "structures_10.sdf", package = "camb")
std.options <- readRDS("standardisation_options.rds")
descriptor.types = c("2D")
require(impute)
predictions <- PredictExternal(structures.file = test_structures_file,
    standardisation.options = std.options, descriptor.types = descriptor.types,
    dataset = dataset, model = readRDS("rf.rds")$finalModel)


## [1] "Standardising Structures: Reading SDF (R)"
## Verifying that rJava is in the library. If not, it will be installed..
## [1] "Generating Descriptors"


print(predictions)


##         id prediction
## 1  B000088   5.488768
## 2  B000139   5.570930
## 3  B000236   5.665863
## 4  B000310   5.259458
## 5  B000728   5.668292
## 6  B000785   5.814345
## 7  B000821   5.772311
## 8  B000826   5.340928
## 9  B001153   5.660730
## 10 B001156   5.449546
```

References

Ben-Hur, A., Ong, C. S., Sonnenburg, S., Schlkopf, B. & Rtsch, G. (2008), 'Support Vector Machines and Kernels for Computational Biology', *PLoS Computational Biology* **4**(10), e1000173.

Bender, A., Jenkins, J. L., Scheiber, J., Sukuru, S. C. K., Glick, M. & Davies, J. W. (2009), 'How similar are similarity searching methods? a principal component analysis of molecular descriptor space', *Journal of Chemical Information and Modeling* **49**(1), 108–119. PMID: 19123924.
URL: *http://dx.doi.org/10.1021/ci800249s*

Breiman, L. (2001), 'Random forests', *Machine Learning* **45**(1), 5–32.

Caruana, R., Niculescu-Mizil, A., Crew, G. & Ksikes, A. (2004), Ensemble selection from libraries of models, *in* 'Proceedings of the Twenty-first International Conference on Machine Learning', ICML '04, ACM, New York, NY, USA, p. 18.

Cortes-Ciriano, I. (2015), 'R Package Conformal'.
URL: *http://cran.at.r-project.org/web/packages/conformal/index.html*

Cortes-Ciriano, I. & Bender, A. (2016), 'How consistent are publicly reported cytotoxicity data? large-scale statistical analysis of the concordance of public independent cytotoxicity measurements', *ChemMedChem* **11**(1), 57–71.
URL: *http://dx.doi.org/10.1002/cmdc.201500424*

Cortés-Ciriano, I., Bender, A. & Malliavin, T. (2015*a*), 'Prediction of PARP Inhibition with Proteochemometric Modelling and Conformal Prediction', *Molecular Informatics* **34**(6-7), 357–366.
URL: *http://dx.doi.org/10.1002/minf.201400165*

Cortes-Ciriano, I., Bender, A. & Malliavin, T. E. (2015*b*), 'Comparing the influence of simulated experimental errors on 12 machine learning algorithms in bioactivity modeling using 12 diverse data sets', *Journal of Chemical Information and Modeling* **55**(7), 1413–1425. PMID: 26038978.
URL: *http://dx.doi.org/10.1021/acs.jcim.5b00101*

Cortes-Ciriano, I., Murrell, D. S., van Westen, G. J., Bender, A. & Malliavin, T. E. (2015), 'Prediction of the potency of mammalian cyclooxygenase inhibitors with ensemble proteochemometric modeling', *Journal of Cheminformatics* **7**(1), 1.

Friedman, J. H. (n.d.), 'Greedy function approximation: A gradient boosting machine.', *The Annals of Statistics* **29**(5), 1189–1232.

Mayer, Z. (2013), 'caretEnsemble: Framework for combining caret models into ensembles. [r package version 1.0]'.

Murrell, D. S., Cortes-Ciriano, I., van Westen, G. J. P., Stott, I. P., Bender, A., Malliavin, T. E. & Glen, R. C. (2015), 'Chemically aware model builder (camb): an r package for property and bioactivity modelling of small molecules', *Journal of Cheminformatics* **7**(1), 45.
URL: *http://dx.doi.org/10.1186/s13321-015-0086-2*

Norinder, U., Carlsson, L., Boyer, S. & Eklund, M. (2014), 'Introducing Conformal Prediction in Predictive Modeling. A Transparent and Flexible Alternative To Applicability Domain Determination', *J. Chem. Inf. Model.* **54**(6), 1596–1603.

Rogers, D. & Hahn, M. (2010), 'Extended-connectivity fingerprints.', *J. Chem. Inf. Model.* **50**(5), 742–754.

Yap, C. W. (2011), 'PaDEL-descriptor: an open source software to calculate molecular descriptors and fingerprints.', *J. Comput. Chem.* **32**, 1466–1474.
URL: *http://www.ncbi.nlm.nih.gov/pubmed/21425294*