QSPR with 'camb'
Chemically **A**ware **M**odel **B**uilder

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

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

May 29, 2014

In the following sections, we demonstrate the utility of the *camb* package by presenting a pipeline to generate various aqueous solubility models using 2D molecular descriptors calculated by the PaDEL-Descriptor package as input features. These models are then ensembled to create a single model with a greater predictive accuracy.

Firstly, the package needs to be loaded and the working directory set:

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

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

1 Compounds

1.1 Reading and Preprocessing

The compounds are read in and standardised. Internally, the Indigo C API [**TBD**], which is incorporated into the `camb` package, is use to perform this task. Molecules are represented with implicit hydrogens, dearomatized, and passed through the InChI format to ensure that tautomers are represented by the same SMILES.

The `StandardiseMolecules` function enables the representation of molecular structures in a similarly processed form. The different arguments of this function allow control over the maximum number of (i) fluorines, (ii) chlorines, (iii) bromines, and (iv) iodines the molecule contains 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`. This is the function's default behaviour. The upper and lower limits for the 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 input to the argument `structures.file`.

```
StandardiseMolecules(structures.file="solubility_2007_ref2.sdf",
                     standardised.file="standardised.sdf",
                     removed.file="removed.sdf",
                     properties.file = "properties.csv",
                     remove.inorganic=TRUE,
                     fluorine.limit=3,
                     chlorine.limit=3,
                     bromine.limit=3,
                     iodine.limit=3,
                     min.mass.limit=20,
                     max.mass.limit=900)
```

Molecules that Indigo manages to parse and that pass the filters are written to the file indicated in the argument "standardised.file". By contrast, molecules that are discarded for training purposes are written to the file indicated in the "removed.file" argument.

2 Target Visualisation

The properties specified in the structure file of all molecules and an index, in the column "kept", indicating which molecules were deleted (0) and kept (1), are written to the file indicated in the argument "properties.file" which is in CSV format. In this case the file is `"properties.csv"`.

```
properties <- read.table("properties.csv", header=TRUE, sep="\t")
properties <- properties[properties$Kept==1, ]
head(properties)
targets <- data.frame(Name = properties$NAME, target = properties$EXPT)
p <- DensityResponse(targets$target) + xlab("LogS Target Distribution")
p
```

2.1 PaDEL Descriptors

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

```
descriptors <- GeneratePadelDescriptors(standardised.file = "standardised.sdf",
    types = c("2D"), threads = 1)
descriptors <- RemoveStandardisedPrefix(descriptors)
saveRDS(descriptors, file = "descriptors.rds")
descriptors <- readRDS("descriptors.rds")
```

3 Statistical Pre-processing

Merge the calculated descriptors and the target values by name into a single data.frame. Check that the number of rows of the merged and original data.frames are the same. Split the data.frame into *ids*, x and y where *ids* are the molecule names, x are the descriptor values and y is the target values.

```r
all <- merge(x = targets, y = descriptors, by = "Name")
# check the number of rows are the same
dim(all)
dim(targets)
dim(descriptors)
ids <- all$Name
x <- all[3:ncol(all)]
y <- all$target
```

```r
# replace the infinite values with NA and impute
# the remaining NA values
x.finite <- ReplaceInfinitesWithNA(x)
x.imputed <- ImputeFeatures(x.finite)
```

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 descriptor values, the R package *impute* is required. Depending on the R version, it can be accessed from either *CRAN* or *Bioconductor*.

Split the dataset into a training (80%) and a holdout (20%) set that will be used to assess the predictive ability of the models. Remove the following descriptors: (i) those with a variance close to zero (near-zero variance), and (ii) those highly correlated:

```r
dataset <- SplitSet(ids, x.imputed, y, percentage = 20)
dataset <- RemoveNearZeroVarianceFeatures(dataset)
dataset <- RemoveHighlyCorrelatedFeatures(dataset)
```

Center and scale the descriptors:

```r
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:

```r
dataset <- GetCVTrainControl(dataset)
saveRDS(dataset, file = "dataset.rds")
```

4 Model Training

```r
# register the number of cores to use in training
registerDoMC(cores = 1)

# train and save a support vector machine model
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))
model <- train(dataset$x.train, dataset$y.train, method,
    tuneGrid = tune.grid, trControl = dataset$trControl)
saveRDS(model, file = paste(method, ".rds", sep = ""))

model <- readRDS("svmRadial.rds")
plot(model, metric = "RMSE")

# train and save a random forest model
method <- "rf"
tune.grid <- expand.grid(.mtry = seq(5, 100, 5))
model <- train(dataset$x.train, dataset$y.train, method,
    tuneGrid = tune.grid, trControl = dataset$trControl)
saveRDS(model, file = paste(method, ".rds", sep = ""))
```

```
# train and save a generalised boosted regression
# model
method <- "gbm"
tune.grid <- expand.grid(.n.trees = c(500, 1000), .interaction.depth = c(25),
    .shrinkage = c(0.04, 0.08, 0.16))
model <- train(dataset$x.train, dataset$y.train, method,
    tuneGrid = tune.grid, trControl = dataset$trControl)
saveRDS(model, file = paste(method, ".rds", sep = ""))
```

Determine if your hyper-parameter search needs to be altered.

```
model <- readRDS("svmRadial.rds")
plot(model, metric = "RMSE")


## Error:  There are no tuning parameters for this model.
```