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*

June 29, 2014

In the following sections, we demonstrate the utility of the `camb` [1] package by presenting a pipeline which generates 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. The trained ensemble is then put to use in making predictions for new molecules.

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 [2], 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 molecules 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`. 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 provided by 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 by the `standardised.file` argument. Molecules that were discarded before training are written to the file indicated by the `removed.file` argument.

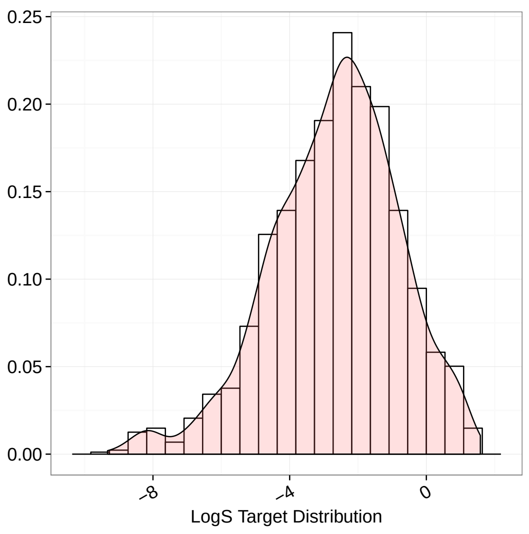

Figure 1: LogS Target Distribution

2 Target Visualisation

The molecular properties specified in the structure file as well as an index are written to the file indicated in the argument `properties.file` which is in CSV format. A column `kept` is added which indicates which molecules were deleted (0) or kept (1). `camb` provides a function to visualise the density of the target variable.

```
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 descriptors can be calculated with the function `GeneratePadelDescriptors` provided by the PaDEL-Descriptor[3] Java library built into the `camb` package:

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

3 Statistical Pre-processing

The descriptors and the target values are then merged by name into a single *data.frame*. We check that the number of rows of the merged and original *data.frames* are the same. We then 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.

```
all <- merge(x = targets, y = descriptors, by = "Name")
ids <- all$Name
x <- all[3:ncol(all)]
y <- all$target
```

Sometimes, some descriptors are not calculated for all molecules, giving a "NA" or "Inf" as the 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. "Inf" descriptor values are first 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`.

```
x.finite <- ReplaceInfinitesWithNA(x)
x.imputed <- ImputeFeatures(x.finite)
```

```
## Loading required package:  impute


## Cluster size 1606 broken into 375 1231
## Done cluster 375
## Done cluster 1231
```

The dataset is split into a training (80%) set used for training and a holdout (20%) set used to assess the predictive ability of the models on molecules drawn from the same distribution as the training set. Unhelpful descriptos are removed: (i) those with a variance close to zero (near-zero variance), and (ii) those highly correlated with one another:

```
dataset <- SplitSet(ids, x.imputed, y, percentage = 20)
dataset <- RemoveNearZeroVarianceFeatures(dataset,
    frequencyCutoff = 30)


## 397 features removed with variance below cutoff


dataset <- RemoveHighlyCorrelatedFeatures(dataset,
    correlationCutoff = 0.95)


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

The descriptors are converted to z-scores by centering them to have a mean of zero and scaling them to have unit variance:

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

Five fold cross-validation (CV) is be used to optimize the hyperparameters of the models:

```
dataset <- GetCVTrainControl(dataset, folds = 5)
saveRDS(dataset, file = "dataset_logS_preprocessed.rds")
```

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 (see below). It is important to mention 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 control more arguments of the underlying `caret` functions. This is certainly possible as the arguments given to the `camb` functions will be subsequently given to the `caret` counterparts. 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.

4 Model Training

In the following section we present the different steps required to train a QSPR model with `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. Pro tip: Dropbox can be used to sync this proceedure so that manual transfer is not required.

```
dataset <- readRDS("dataset_logS_preprocessed.rds")
# register the number of cores to use in training
registerDoMC(cores = 10)
```

4.1 Support Vector Machines (SVM)

Firstly, a SVM using a radial basis function is trained[4]. We then define an exponential grid (base 2) to optimize over the hyperparameters. 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))
model <- train(dataset$x.train, dataset$y.train, method,
    tuneGrid = tune.grid, trControl = dataset$trControl)
saveRDS(model, file = paste(method, ".rds", sep = ""))
```

4.2 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))
model <- train(dataset$x.train, dataset$y.train, method,
    tuneGrid = tune.grid, trControl = dataset$trControl)
saveRDS(model, file = paste(method, ".rds", sep = ""))
```

4.3 Gradient Boosting Machine

We proceed similarly in the case of a gradient boosting machine (GBM) model[6].

```
method <- "gbm"
tune.grid <- expand.grid(.n.trees = c(500, 1000), .interaction.depth = c(25),
    .shrinkage = c(0.01, 0.02, 0.04, 0.08))
model <- train(dataset$x.train, dataset$y.train, method,
```

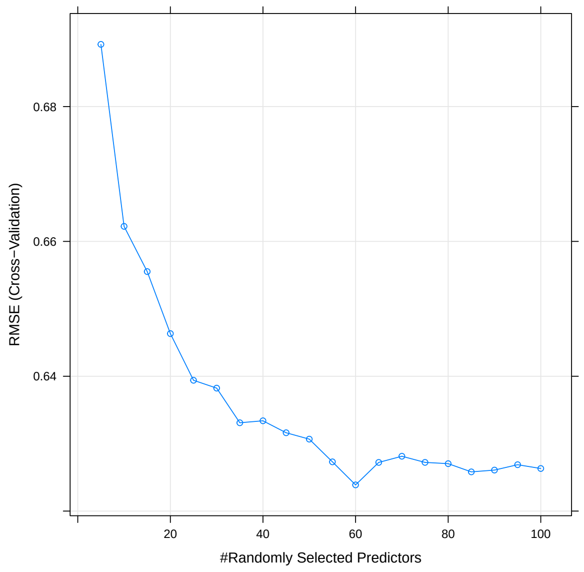

Figure 2: CV RMSE over the hyperparameters

```
    tuneGrid = tune.grid, trControl = dataset$trControl)
saveRDS(model, file = paste(method, ".rds", sep = ""))
```

For each model we determine if our hyper-parameter search needs to be altered. In the following we focus on the RF model, though the same steps can be applied to the GBM and SVM models. If your hyper-parameters lead you to what looks like a global minimum then you can stop scanning the space of hyper-parameters, otherwise you need to adjust the grid and retrain your model.

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

5 Model Evaluation

Once the models are trained, the cross validated metrics for the optimised hyper-parameters become visible:

```
print(RMSE_CV(model, digits = 3))

## [1] 0.624

print(Rsquared_CV(model, digits = 3))

## [1] 0.8895
```

On the basis of the soundness of the obtained models, assessed through the value of the cross-validated metrics, we proceed to predict the values for the external (hold-out) set:

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

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 = 26,
    XAxisSize = 20, YAxisSize = 20, TitleAxesSize = 24,
    margin = 2, PointColor = "black", PointShape = 16,
    MarginWidth = 1, AngleLab = 0, xlab = "Observed",
    ylab = "Predicted")

## Error:  object 'holdout.predictions' not found
```

6 Ensemble Modeling

In the following section, two ensemble modeling techniques will be applied, namely greedy optimization and model stacking. Further information can be found in ref [**caretEnsemble**] and [7].

We insert all the trained models into a list. Output the cross-validation RMSE of the trained models.

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

# sort the models from lowest to highest RMSE
names(all.models) <- sapply(all.models, function(x) x$method)
sort(sapply(all.models, function(x) min(as.vector(na.omit(x$results$RMSE)))))


##       gbm        rf svmRadial
##    0.5903    0.6239    0.6320
```

A greedy ensemble is then trained using 1000 iterations. The Greedy ensemble picks a liner combination of model output that minimises the RMSE.

```
greedy <- caretEnsemble(all.models, iter = 1000L)


## Loading required package:  pbapply

sort(greedy$weights, decreasing = TRUE)


##       gbm svmRadial        rf
##     0.580     0.336     0.084

saveRDS(greedy, file = "greedy.rds")
greedy$error


##   RMSE
## 0.5742
```

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 0.574   0.9035 0.04047    0.01881
```

We also create a non-linear stack ensemble that uses the trained model inputs as input into the stack. In this case we use a Random Forest as the stacking model.

```
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)
```

```
## Loading required package:  randomForest
## randomForest 4.6-7
## Type rfNews() to see new features/changes/bug fixes.
```

```
## note: only 2 unique complexity parameters in default grid. Truncating the grid to 2 .
```

```
saveRDS(nonlinear, file = "nonlinear.rds")
nonlinear$error
```

```
##   mtry   RMSE Rsquared  RMSESD RsquaredSD
## 1    2 0.6118   0.8919 0.07471    0.02498
## 2    3 0.6184   0.8896 0.07587    0.02516
```

The greedy and the linear stack have a cross validated RMSE that is 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    ENS_greedy           gbm ENS_nonlinear
##        0.5109        0.5126        0.5156        0.5532
##            rf     svmRadial
##        0.5938        0.5984
```

TBD - predictions on new molecules.

7 Bibliography

References

[1] D. S. Murrell, I. Cortes-Ciriano, G. J. van Westen, T. Malliavin, and A. Bender, "Chemistry Aware Model Builder (camb): an R package for predictive bioactivity modeling," *http://github.com/cambDI/camb*, 2014.

[2] Indigo, "Indigo cheminformatics library," 2013. [Online]. Available: http://ggasoftware.com/opensource/indigo/.

[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] 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.

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

[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.

[7] 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.