Error variance estimation with 'eve'
Error **V**ariance **E**stimator

Daniel S. Murrell*[1]

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

July 23, 2014

In the following sections, we demonstrate the utility of the `eve` [**eve**] package by presenting a pipeline which trains a support vector machine to predict aqueous solubility using 2D molecular descriptors calculated by the PaDEL-Descriptor package. `eve` is then used to train an error variance model to predict the variance in the errors of new molecules.

Firstly, the package is loaded and the working directory set:

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

1 Compounds

1.1 Reading and Preprocessing

The compounds are read in and standardised. Internally, Indigo's C API [1], incorporated into the `camb` package, is use to perform this task. Molecules are represented with implicit

*dsmurrell@gmail.com

hydrogens, dearomatized, and passed through the InChI format to ensure that tautomers are represented by the same SMILES.

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`. This is the function's default behaviour. 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 by the argument `structures.file`.

```
std.options <- 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 once they have been 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 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).

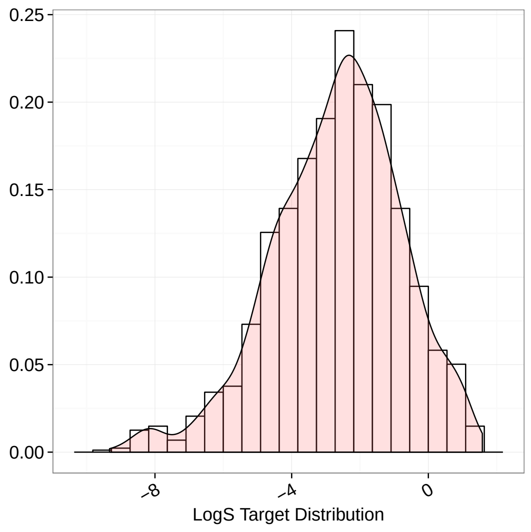

Figure 1: LogS Target Distribution

2 Target Visualisation

camb provides a function to visualise the density of the target variable. Visualising the distribution of the target variable gives can give a measure of how accurate a trained model is. The narrower the distribution the lower the RMSE should for the model to exhibit predictive power.

```
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 [2] Java library built into the `camb` package:

```
descriptor.types <- c("2D")
descriptors <- GeneratePadelDescriptors(standardised.file = "standardised.sdf",
    types = descriptor.types, 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 is the block of 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 in the molecules that are closest to the molecule with the missing information. ”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`.

```
## Loading required package:  impute


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

The dataset is randomly split into a training set (80%) used for training and a holdout set (20%) which 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 support vector machine:

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

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.

```r
dataset <- readRDS("dataset_logS_preprocessed.rds")
# register the number of cores to use in training
registerDoMC(cores = 1)
```

4.1 Support Vector Machines (SVM)

A SVM using a radial basis function kernel is trained [3]. A base 2 exponential grid is used to optimize over the hyperparameters. The `train` function from the `caret` package is used directly for model training.

```r
library(kernlab)
method <- "svmRadial"
tune.grid <- expand.grid(.sigma = expGrid(-10, -6,
    1, 2), .C = c(0.5, 1, 2, 4, 8, 16, 32))
model <- train(dataset$x.train, dataset$y.train, method,
    tuneGrid = tune.grid, trControl = dataset$trControl,
    )
saveRDS(model, file = paste(method, ".rds", sep = ""))
```

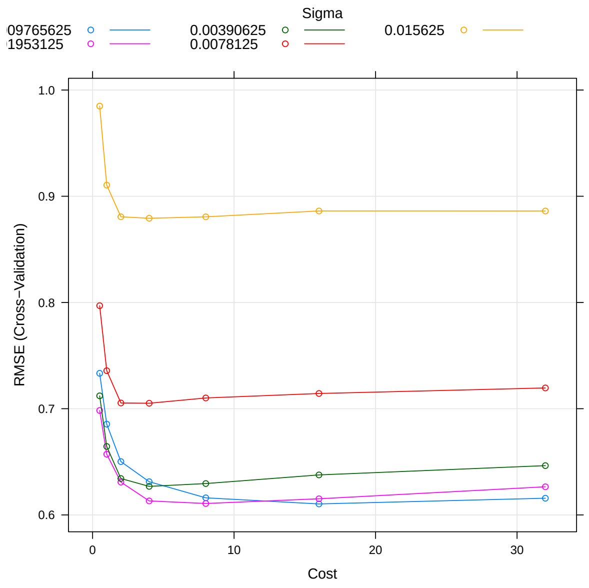

Figure 2: CV RMSE over the sigma and cost hyperparameters for the SVM

We determine if our hyper-parameter search needs to be altered. If the hyper-parameters scanned 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.

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

5 Model Evaluation

Once the SVM model is trained, the cross validated metrics for the optimised hyper-parameters become visible:

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


## [1] 0.61


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

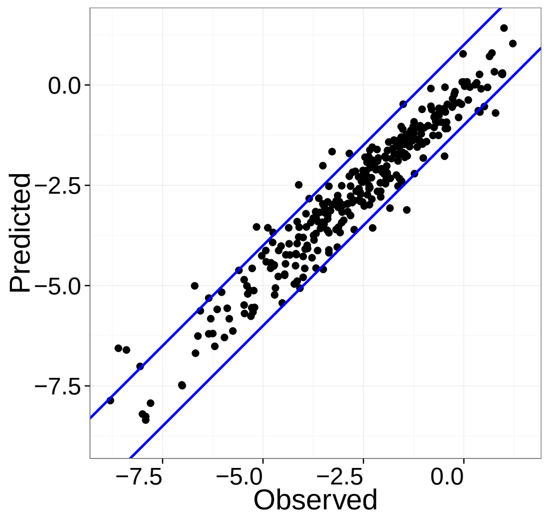

Figure 3: Observed vs Predicted

```
## [1] 0.8911
```

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")
```

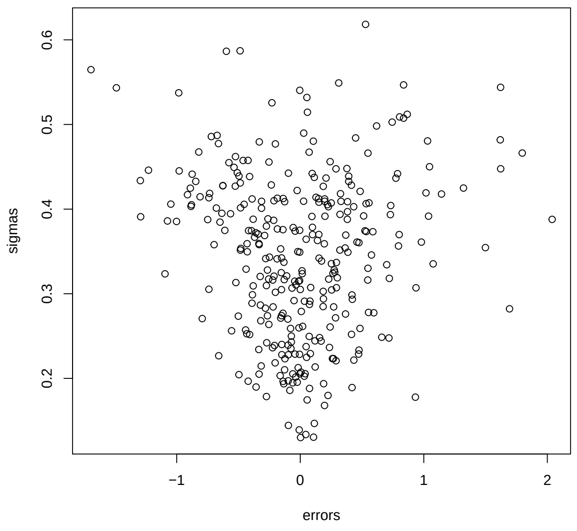

Figure 4: Estimate of the standard deviation (sigma) plotted against the actual error for molecules in the holdout set

6 Error Variance Estimation

In the following section, the `eve` package is invoked to train an error variance model.

The trained caret model in used by the `eve` package to train an error variance model. This model assumes a Guassian distribution on the errors made by the support vector machine. The

```r
eve <- BuildCaretEVEstimator(x = dataset$x.train, model = model,
    Nmax = 20, cores = 1)

sigmas <- PredictSigmas(x = dataset$x.holdout, estimator = eve)$sigmas
errors <- holdout.predictions - dataset$y.holdout

plot(errors, sigmas)
```

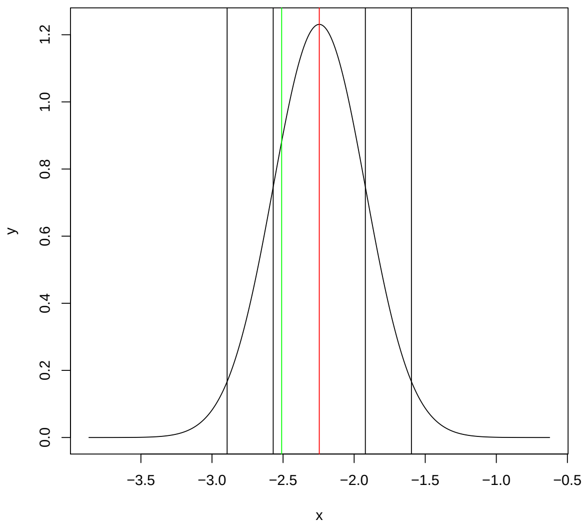

Figure 5: Prediction for molecule 100 of the holdout set (red). Distribution of possible observed values with black lines denoting 1 and 2 sigma. Observation (green).

```r
i <- 100
prediction <- holdout.predictions[i]
observation <- dataset$y.holdout[i]
sigma <- sigmas[100]
x <- seq(prediction - 5 * sigma, prediction + 5 * sigma,
    length = 1000)
y <- dnorm(x, mean = prediction, sd = sigma)
plot(x, y, type = "l", lwd = 1)
abline(v = prediction - sigma)
abline(v = prediction + sigma)
abline(v = prediction - 2 * sigma)
abline(v = prediction + 2 * sigma)
abline(v = prediction, col = "red")
abline(v = observation, col = "green")
```

7 External Predictions

One of the main attractions of this package is that it makes standardising and making predictions on new molecules a simple task. It is essential to ensure that the same standardisation options and descriptor types are used when the model is applied to make predictions for new molecules.

```
test_structures_file <- system.file("test_structures",
    "structures_10.sdf", package = "camb")
predictions <- PredictExternal(test_structures_file,
    std.options, descriptor.types, dataset, model)


## [1] "Standardising Structures: Reading SDF (R)"
## [1] "Generating Descriptors"


print(predictions)


##          id prediction
## 1  B000088    -1.9523
## 2  B000139    -2.3624
## 3  B000236    -7.4301
## 4  B000310    -4.9088
## 5  B000728    -2.8525
## 6  B000785    -2.5881
## 7  B000821    -2.1716
## 8  B000826    -3.1698
## 9  B001153     0.3744
## 10 B001156    -1.5038
```

References

[1] Indigo, “Indigo cheminformatics library,” 2013. [Online]. Available: http://ggasoftware.com/opensource/indigo/.

[2] 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.

[3] 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.