Package Vignette

SimuChemPC

Mohsen Ahmadi
mohsen_ahmadi989@yahoo.com

Jan 2014

Contents

1 Motivation

QSAR (Quantitative Structure-Activity Relationship) modeling is used to predict the activity of compounds relying on the activity of known compounds and whose ultimate aim is to identify highly potent compounds. In this package, we propose an efficient method (i.e. **EI**) by which finding potent compounds is faster than underlying competitors (i.e. **GP**, **NN** and **RA**)

Molecular Descriptors are calculated for each compound where they are related to the measured biological activity from a mathematical aspect.

In this tutorial, we are going to walk through a workflow example in which you can see how the package works and can be used.

2 Data Initialization

First of all, these libraries should be included into your R (>=2.13).

```
> library(rcdk)
> library(SimuChemPC)
```

Starting from a sample SDF file, We use **rcdk** package to get features/descriptors as the input for **SimuChemPC** package. We load our sample SDF file like so (of course, the path of your file should be set accordingly):

```
> mols <- load.molecules( c('sample.sdf') )
```

We have different descriptors to be used as input data. In our tutorial we use a type of **WeightedPathDescriptor**.

```
> dn <- get.desc.names()
> dn[47]

[1] "org.openscience.cdk.qsar.descriptors.molecular.WeightedPathDescriptor"

> features <- eval.desc(mols, dn[47])
```

Having calculated descriptors in hand, we are going to fetch potencies from our sample SDF file.

In such a SDF file, there are some properties for each molecule inside. We can see the list of properties for first molecule as follows:

```
> get.properties(mols[[1]])
```

With the help of "***get.property***" function we can fetch specific attributes from each molecule in our sample SDF file. In order to fetch "molecule index" and "potency" for each of which we do like so:

```
> get.property(mols[[1]], "Potency[nM]")

[1] "160"

> get.property(mols[[1]], "cr_index")

[1] "1301769"
```

To do so for all molecules in SDF file, we make a data frame object called "potency" and we bind it as follows:

```
> p = data.frame()

> for (i in 1:length(mols))
+ {
+ Potency = get.property(mols[[i]], "Potency[nM]");
+ p = rbind(p, data.frame(Potency))
+ }
```

After taking those values out, it's the time to combine each of descriptors with corresponding potency values. Here we go:

```
> datafile <- cbind(features, p)
> dim(datafile)

[1] 100   6

> datafile[1:5,]

    WTPT.1   WTPT.2   WTPT.3    WTPT.4   WTPT.5 Potency
1 40.60574 2.030287 28.52843  8.170702 15.31685     160
2 44.62161 2.028255 34.18068  8.235517 23.36887    8500
3 40.60574 2.030287 28.52843  8.170702 15.31685     300
4 40.60574 2.030287 28.52843 10.636011 15.31685      40
5 40.60574 2.030287 28.52843  8.170702 17.78216     150
```

At this point, we are done within data initialization. We use this data frame object further as input in package workflow.

3 Utility Functions

We have two functions namely "trainChemPC" and "predictChemPC".
The first function is used to get **train data** and **target values** as input and returns "loghyper" parameters.

The latter function takes "train data", "test data" (for prediction), "target values", "method" (one of four different methods) and "loghyper" parameters already obtained from first function. In order to apply these functions on our

sample data, first we split data into two parts for train and test data. So, we try:

```
> len <- dim(datafile)[1]
> len

[1] 100

> half <- dim(datafile)[1] / 2
> half

[1] 50

> col <- dim(datafile)[2]
> col

[1] 6

> xTrain <- datafile[1:half, 1:(col-1)]
> yTrain <- as.matrix(as.numeric(array(datafile[1:half, col])))

> xTest <- datafile[(half+1):len, (1:col-1)]
> yTest <- as.matrix(as.numeric(array(datafile[(half+1):len, col])))
```

We pass "xTrain" and "yTrain" to trainChemPC function as input and we get loghyper parameters as output of this function. The result of our sample is shown here:

```
> loghyper = trainChemPC(xTrain, yTrain)

> loghyper
            [,1]
[1,]  0.6454240
[2,] -0.3734154
[3,] -0.2242068
```

In the light of predictChemPC function, we make a prediction based on four different methods namely: **EI**, **GP**, **NN**, **RA**.

- EI (*Expected Improvement*): A compound for which maximum expected potency improvement is reached.

- GP (*Gaussian Process Regression*): A compound holding maximum predicted potency in test data is selected.

- NN (*Nearest Neighbor*): A compound that is nearest (Tonimito Coefficient as distance measure) to the most potent compound in training data is selected.

- RA (*Random*): As it's name suggests, a compound is selected randomly.

Here are the results of predictChemPC for our sample data in each method:

```
> predictChemPC(xTrain, yTrain, xTest, loghyper, method="RA")
[1] "index of compound based on: RA => 16"


> predictChemPC(xTrain, yTrain, xTest, loghyper, method="NN")
[1] "index of compound based on: NN => 46"


> predictChemPC(xTrain, yTrain, xTest, loghyper, method="GP")
[1] "index of compound based on: GP => 46"


> predictChemPC(xTrain, yTrain, xTest, loghyper, method="EI")
[1] "index of compound based on: EI => 47"
```

4 Simulation set-up

In this section we touch the simulation set-up in detail. What happens in the heart of simulation is as follows:

- Data are randomly divided into roughly equal size of train and test data.
- Normalization is applied to train and test data.
- Feature Selection is applied to normalized data to ignore irrelevant and redundant features w.r.t. target values(potency). It's, in turn, born by three steps:
  - We compute *spearman rank correlation* between each feature i and potency (we name correlation values as p-values afterwards)
  - We apply *Bejnamini and Hochberg FDR* procedure over p-values
  - Eventually, we select those features for which adjusted p-value <= 0.05 (i.e. their null hypotheses are rejected)
- During a loop, training data are used to fit the Gaussian Process model. Test data are scanned for the compound with maximal expected potency improvement. Subsequently, selected compound is added to training data to further refine the model. The process of adding compounds is continued until all test data are consumed.

The last parameter of "SimuChemPC" is meant to repeat these steps. For the purpose of this tutorial, this parameter is set to 5. At the end of the day, the number of simulation steps that is required to find most potent compound in the original test set is considered.

```
> dataX <- datafile[, 1:(col-1)]
> dataY <- as.matrix(as.numeric(array(datafile[, col])))

> dim(dataX)
[1] 100 5

> dim(dataY)
[1] 100 1

> rank <- SimuChemPC(dataX, dataY, "RA", 5)
> rank <- SimuChemPC(dataX, dataY, "NN", 5)
> rank <- SimuChemPC(dataX, dataY, "GP", 5)
> rank <- SimuChemPC(dataX, dataY, "EI", 5)
```

5 Results

We applied simulation to the sample data and the result is demonstrated in Figure 1 and Figure 2. As you can see in figures, **EI** outperforms other methods as it takes less steps to find most potent compound in the original test set.

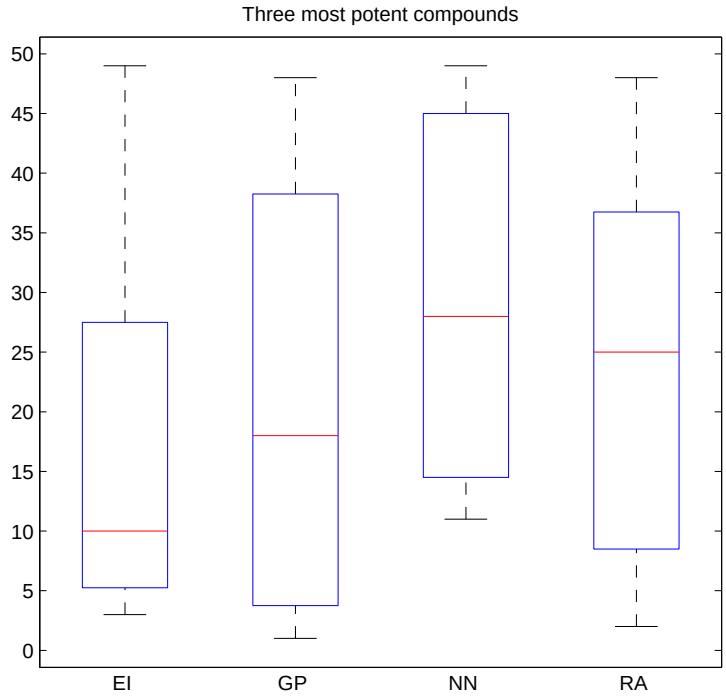

Figure 1: Three most potent compound for each method

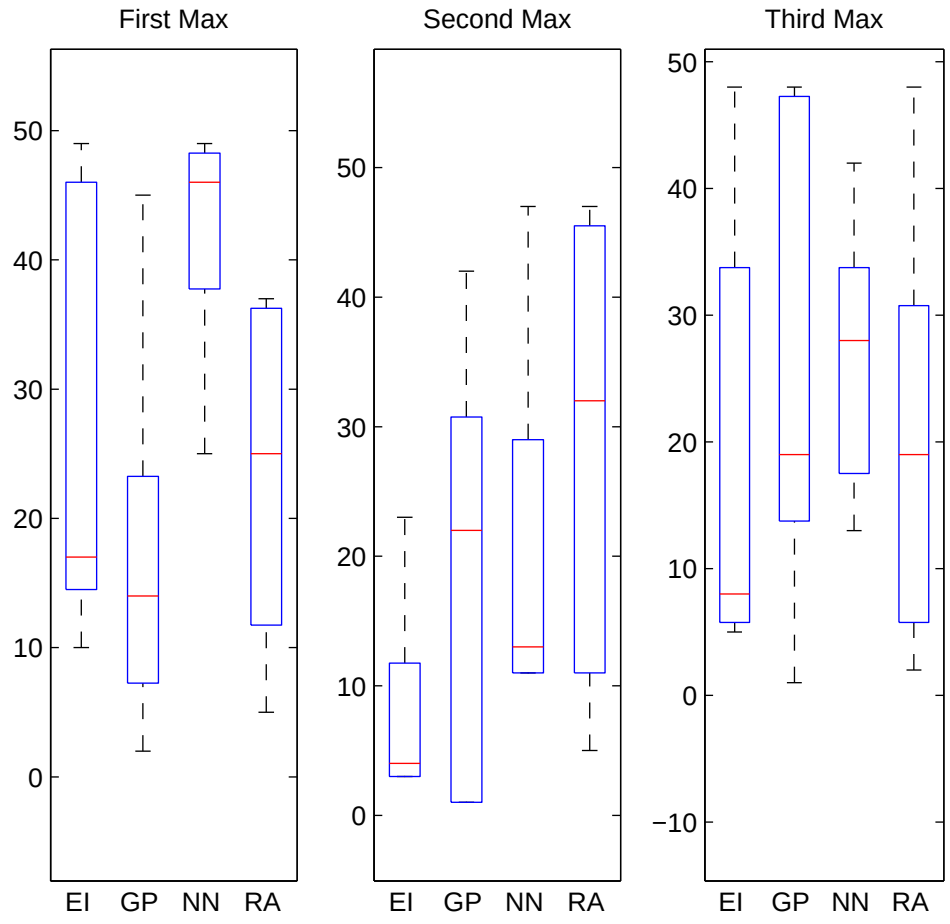

Figure 2: First, second and third max box plots for each method

6 R Session Information

```
> sessionInfo()

R version 2.13.0 (2011-04-13)
Platform: i386-pc-mingw32/i386 (32-bit)

locale:
[1] LC_COLLATE=English_United States.1252
LC_CTYPE=English_United States.1252
LC_MONETARY=English_United States.1252

[4] LC_NUMERIC=C
LC_TIME=English_United States.1252

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods   base

other attached packages:
[1] SimuChemPC_1.2    rcdk_3.1.7        iterators_1.0.5   png_0.1-4
        fingerprint_3.4.7 rcdklibs_1.4.7    rJava_0.9-3
```