**Tutorial 3: Quantitative structure-activity/property relationship (QSAR/QSPR) modeling and analysis**

In this tutorial we will cover one of the most fundamental and important components of small molecule cheminformatics – QSAR (and QSPR) modeling. Traditionally, QSAR is a ligand-based method. The premise is that we can predict a property of a small molecule (e.g., bioactivity in the case of QSAR or solubility in the case of QSPR) by training an algorithm to mathematically relate various chemical properties of many molecules to the property of interest. Thus, these methods typically make use of aggregated data from many small molecules for which we have experimental information on the property of interest.

In the previous tutorial we covered how to compute properties, combine properties in custom definitions, and generate feature datasets composed of molecular properties. Here, we will use those datasets to generate models to generate QSAR and QSPR models.

**The syntax for applications relating to model training and validation has been described in detail in the following manuscript:**

“Introduction to the BioChemical Library (BCL): An application-based open-source toolkit for integrated cheminformatics and machine learning in computer-aided drug discovery”

Benjamin P. Brown\* , Oanh Vu, Alexander R Geanes, Sandeepkumar Kothiwale, Mariusz Butkiewicz, Edward W Lowe, Ralf Mueller, Richard Pape, Jeffrey Mendenhall\* and Jens Meiler\*

\*Co-corresponding authors

At the time of writing, this manuscript is in revisionat *Frontiers in Pharmacology*. **To shorten the length of this tutorial and focus on important concepts rather than syntax, we will be referencing sections of this manuscript as supplementary material throughout.** All of the necessary command-lines to complete the tutorial with the provided files will still be given.

Part 1. Single-task classification QSAR models for the potassium ion channel Kir2.1

For the first part of the tutorial we will build shallow (single hidden layer) fully-connected feed forward artificial neural network to classify candidate molecules as either inhibitory (active) or not (inactive) for the Kir2.1 inward rectifier potassium ion channel. This example is an excerpt from a larger benchmark previously published by the Meiler Lab (<https://pubmed.ncbi.nlm.nih.gov/23299552/>). Here, we expand this sample case to demonstrate performance on several core concepts in QSAR modeling.

This section assumes familiarity with the BCL descriptor framework and molecule processing / handling in the BCL. If you need a refresher, please check out Tutorials 1 and 2. See section 7 in Brown et al. (CITATION) for an overview of the model application group in the BCL.

The file we will start with is “1843\_combined.labeled.sdf.gz”. This compressed SDF contains both the active and inactive molecules required for training our QSAR model. For details on how the dataset was assembled, please see Mariusz et al. (CITATION). The molecules are already labeled with a classification result label “IsActive” based on the criteria from Mariusz et al. (CITATION).

Create a dataset using the 1315 features described in Mendenhall & Meiler, 2016 (CITATION).

bcl.exe descriptor:GenerateDataset \

-source "SdfFile(filename=1843\_combined.labeled.sdf.gz)" \

-feature\_labels MendenhallMeiler2015.Minimal.object \

-result\_labels “Combine(IsActive)” \

-output 1843\_combined.labeled.MendenhallMeiler2015.Minimal.bin \

-scheduler PThread 4

Randomize the rows (molecules) of the dataset (note that we could have done that in the previous step as well, but then we would have been limited to using one thread.

bcl.exe descriptor:GenerateDataset \

-source \ "Randomize(Subset(filename=1843\_combined.labeled.MendenhallMeiler2015.Minimal.bin))" \

-output 1843\_combined.labeled.MendenhallMeiler2015.Minimal.rand.bin

Once the randomized dataset is prepared, we can train our model. Section 7.1 in Brown et al. (CITATION) reviews the syntax required to use model:Train. There exists a convenient Python wrapper script included in the BCL that simplifies cross-validation and (if desired) feature selection. This script is described in section 7.2 of Brown et al. (CITATION) and we will make use of it here.

bcl/scripts/machine\_learning/launch.py \

-t cross\_validation --config-file config.ini \

-d 1843\_combined.labeled.MendenhallMeiler2015.Minimal.rand.bin \

--id 1843\_combined.labeled.MendenhallMeiler2015.Minimal.1x32\_005\_025 \

--local --just-submit

Briefly, this command prepares and launches a random-split cross-validation exercise with model:Train. The training data are from the randomized .bin file we created a moment ago. We have an ID on the output directories that reflects our dataset numeric ID (1843), the feature set we used (MendenhallMeiler2015.Minimal), and some details about our model (1x32\_005\_025). The job is submitted locally and sent to the background. If you are a member of the Meiler Lab, you can also SSH into the CSB cluster and pass --slurm instead of --local and the launch.py script will submit your jobs to the cluster. The launch.py script currently supports local, GNU parallels, SLURM, and PBS submission types.

Most of the important options are specified in the “config.ini” file passed via --config-file. **Importantly, flags passed to the terminal command-line override config-file settings.** We will review a few important sections of the config-file below:

[variables]

objective-function: 'AucRocCurve(cutoff=0.5,parity=1,x\_axis\_log=1,min fpr=0.001,max fpr=0.1)'

The objective function that we will use to evaluate model performance is a receiver-operating characteristic (ROC) curve AUC (area under the curve). We will log-scale the x-axis to measure early enrichment (the true positive rate as a function of false positive rate between 0.001 and 0.1), referred to as logAUC.

[learning]

learning-method: 'NeuralNetwork( transfer function = Sigmoid, weight update = Simple(alpha=0.5,eta=0.01),dropout(0.05,0.25),objective function = %(objective-function)s,scaling=AveStd,steps per update=1,hidden architecture(32),balance=True,balance target ratio=0.1,shuffle=True,input dropout type=Zero)'

max-iterations: 50

monitor-independent-set:

Our QSAR model will be a neural network (ANN). Our ANN will have a single hidden layer with 32 neurons. Over-training will be prevented by using dropout fraction of 0.05 and 0.25 on the input and hidden layers, respectively. The use of dropout in a single-result classification task allows us to oversample the minor class, which in this case is the active set (172 actives vs. 301321 inactives). We will resample the active molecules until we achieve a ratio of 1/10 active-to-inactive. Training will proceed for 50 iterations. Our monitoring and independent sets are the same sets (we will not be terminating the training early based on monitor set performance anyway).

[cv]

monitoring-id-range: [0,4]

independent-id-range: [0,4]

cross-validations: 5

cv-repeats: 1

We will make 5 chunks for cross-validation. Each round of cross-validation will create one model. For more details, see section 7 from Brown et al. (CITATION).

Observe that there are now three new directories in the directory where we launched the mode: (1) log\_files, (2) results, (3) models. Within each of those three directories is a sub-directory corresponding to the ID we passed at launch. At the end of the run, the BCL computes a number of statistics related to model performance and validation. ROC curves are also plotted automatically with gnuplot.

Let’s visualize the graphics in the results/ 1843\_combined.labeled.MendenhallMeiler2015.Minimal.1x32\_005\_025/ directory.

<ROC curves>

Our ROC curve has an AUC of 0.85 (no-signal yields 0.5) and a logAUC of 0.44 (no signal yields 0.0215), both of which are highly encouraging. We can also see some familiar trends, such as the reduction in PPV as FPR increases.

So how do we actually *use* the output from our model to make an informed estimate on the likely activity of a candidate molecule? Naturally, we want to maximize our true positive and true negative predictions and minimize our false positive and false negative predictions. One strategy would be to determine a cutoff value (i.e., an output value from the ANN that we treat as a threshold at or above which a molecule is considered active and below which a molecule is considered inactive) that maximizes a statistical metric, such as accuracy, F-score, or Matthew’s correlation coefficient (MCC). Instead, however, we will use a metric that we refer to as localPPV.

The localPPV metric estimated at a given cutoff value is an estimate of the PPV *at a singular model output value*. Given positive parity (i.e., more positive model output values correspond to more positive result label values), standard PPV for a given cutoff value is computed as the total number of true positives divided by the number of predicted positives *at or above* the stated cutoff. In other words, localPPV is constructed to be an estimate of PPV at a single discrete cutoff value instead of at or above that value.

Thus, localPPV is a very useful metric because it can provide us with an estimated probability for our result label for any given cutoff value (under the assumption that the actual probability varies monotonically with the model output value). One of the plots produced at the end of our training run displays PPV and localPPV as a function of model cutoff:

<third image from output>