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

Author: Benjamin P. Brown (benjamin.p.brown17@gmail.com)
Author: Jeffrey Mendenhall (jeffreymendenhall@gmail.com)
Date: 01-2022

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 a preprint that we will distribute with the workshop materials[1]. At the time of writing, the Brown et al. reference manuscript[1] is in revision for publication. **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 the necessary command-lines to complete the tutorial with the provided files will still be given.

Once again, set the path to your BCL directory as an environment variable. For example, the main BCL directory for these tutorials in the Meiler Lab is in `/sb/apps/bcl/bcl`. In bash, we can set this as an environment variable by doing the following:

```
export BCL=/sb/apps/bcl/bcl
```

In tcsh, this is accomplished with a similar command:

```
setenv BCL "/sb/apps/bcl/bcl"
```

We can also add the BCL executable to our PATH environment variable.

```
export PATH=/sb/apps/bcl/bcl/build/linux64_release/bin:$PATH
```

Note that the "bcl.exe" is interchangeable with "bcl-apps-static.exe" in
`${BCL}/build/linux64_release/bin`

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 in Butkiewicz et al.[2]. 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.[1] 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 Butkiewicz et al.[2]. The molecules are already labeled with a classification result label "IsActive" based on the criteria from Butkiewicz et al.[2].

Create a dataset using the 1315 features described in Mendenhall & Meiler, 2016[3].

```
bcl.exe descriptor:GenerateDataset \
-source "SdfFile(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.

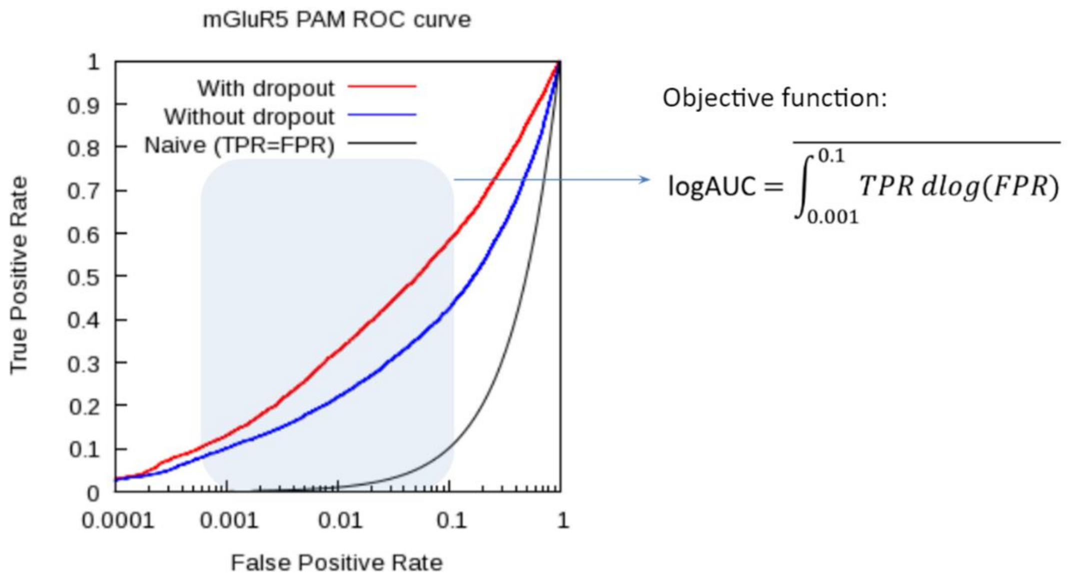

Figure 1. Graphical illustration of the logAUC objective function.

```
[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.[1].

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 several 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.

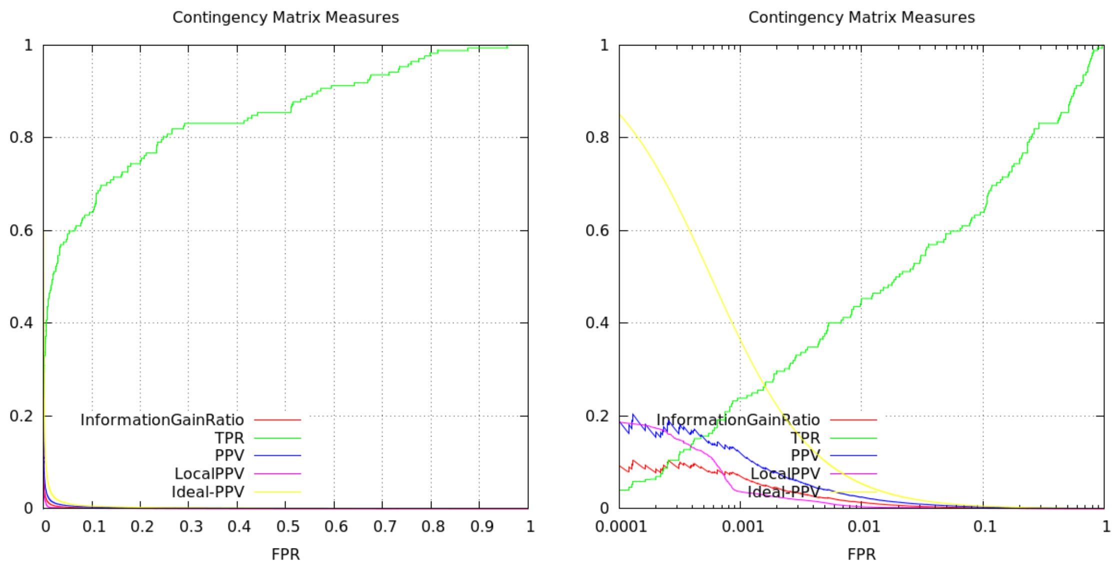

Figure 2. AUC and locAUC curves from the 5x cross-validation 1843 model training. Standard AUC is displayed on the left and logAUC is displayed on the right. This is a representative example of the standard automatically generated output from BCL QSAR model training.

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 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*. In contrast, 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.

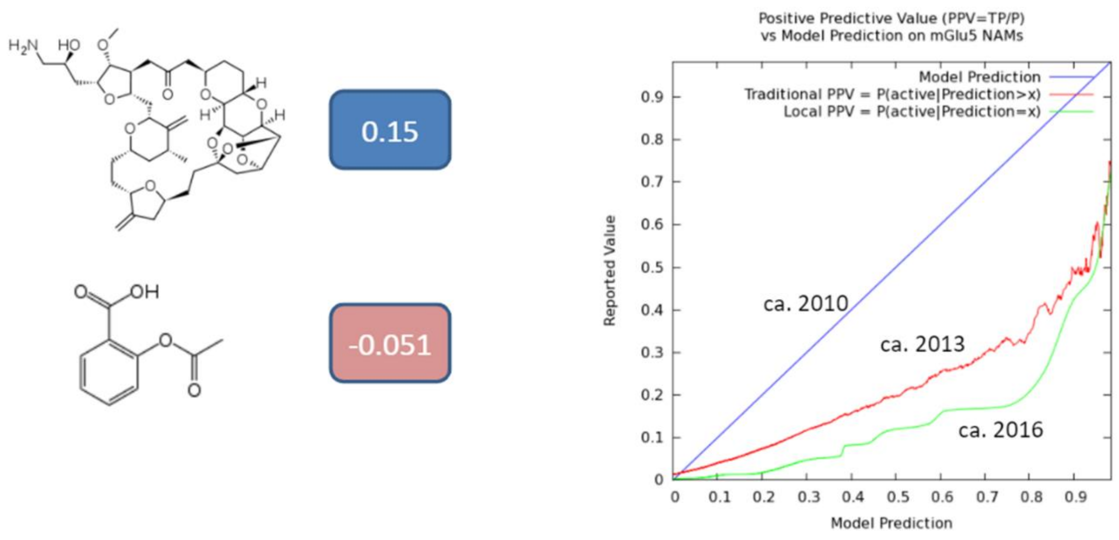

Figure 3. Comparison of PPV and localPPV metrics in a sample mGlu5 NAM virtual screening QSAR model. The molecules on the left are each scored with a QSAR model using the localPPV metric. The localPPV metric is an approximation of the likelihood that each compound is active based on the model prediction output.

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:

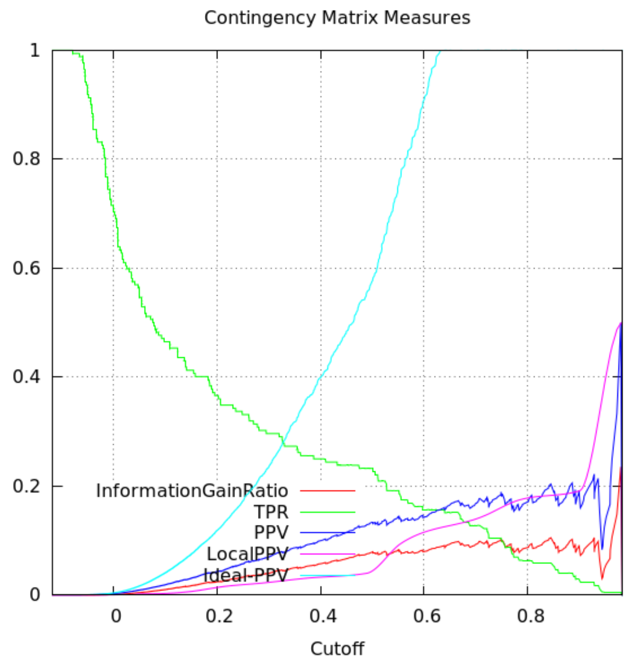

Figure 4. Prediction confidence as a function of model output value. The model output value is given on the x-axis (cutoff). This is a representative example of the standard automatically generated output from BCL QSAR model training.

We see that as our cutoff approaches its upper limit of 1.0, our localPPV reaches a maximum value of approximately 0.5. This means that if we use our QSAR model to predict the bioactivity of a novel compound and the model returns a value of 0.99, then the tested compound has approximately a 50% chance of being active. You can then determine based on the collection of independent probabilities of molecules in your screening set how many compounds you need to order to obtain a hit.

Granted, there are caveats here. The description in the preceding paragraph assumes that the test set compound is pulled from the same feature distribution as was used to train the model. This is almost never the case, which is why random-split cross validation is overly optimistic. Thus, that 50% chance of being active is really a "best case scenario" percentage. There is another form of cross-validation called "leave class out" cross-validation that can be considered close to a worst-case scenario. In this form of validation, you cluster your training data by the training set feature space and use distinct clusters for training and testing, such that your feature space in the training set is separated from the feature space in your test set (for an example, see the validation performed when we created BCL::AffinityNet for protein-ligand interaction scoring[4]).

Another alternative would be time-split cross-validation. In production settings, it is frequently useful to perform different forms of validation to try and approximate the upper and lower boundaries on your probabilities. In section 7.4.3 of Brown et al.[1], we demonstrate how you can also build an applicability domain model to help determine the validity of predictions made by a QSAR model. Unfortunately, that is beyond the scope of the current tutorial. For additional information on model testing and validation in QSAR/QSPR, check out section 7.3 in Brown et al.[1]. There is an example related to solubility prediction that may be helpful.

Moving on, let's screen a dataset. Enamine provides free virtual screening datasets for download (you may need a free account to download them). We will screen a small, focused Enamine library devoted to potential ion channel scaffolds / modulators.

There are several possible commands one can run to test new molecules with our model. Once again, please see section 7.3 in Brown et al.[1] for details. Here, we will not use the `model:Test` application; instead, we will use `molecule:Properties` to define a new property from our QSAR model prediction.

```
bcl.exe molecule:Properties \
-input_filenames
Enamine_Ion_Channel_Library_plated_36800cmds_20200524.clean.sdf.gz \
-output_table \
Enamine_Ion_Channel_Library_plated_36800cmds_20200524.clean.MendenhallMeile
r2015.Minimal.1x32_005_025.csv \
-tabulate "Define(local_ppv=PredictionInfo(\
predictor=File(\
directory=../models/1834_combined.labeled.MendenhallMeiler2015.Minimal.1x32
_005_025.rand/,\
prefix=model),\
metrics(LocalPPV)))" local_ppv \
-scheduler PThread 6
```

The output CSV file contains two columns: the first column are the 0-indexed indices of molecules in the input SDF, and the second column are the localPPV values are each molecule. For readability, we will refer to former output CSV file using pseudocode as `<file>`. We can sort and view the best molecule:

```
tail -n+2 <file> | sort -rgk2 -t, | head -n1
```

Then we can use `molecule:Filter` to extract the molecule that matches this index. Replace the string ID with the number you obtain from running the previous shell command in the command-line below.

```
bcl.exe molecule:Filter \
-input_filenames \
Enamine_Ion_Channel_Library_plated_36800cmds_20200524.clean.sdf.gz \
-compare_property_values Index equal ID \
-output_matched ID.sdf
```

If you want to look at the top 10 best molecules by localPPV, you can make a small file that repeats "Index equal ID" for each unique ID value:

```
tail -n+2 <file> | sort -rgk2 -t, | head | \
awk -F, '{print $1}' | sed "s:^:Index equal :g" | \
tr '\n' ' ' > top_hits_formatted.txt
```

```
bcl.exe molecule:Filter \
-input_filenames
Enamine_Ion_Channel_Library_plated_36800cmds_20200524.clean.sdf.gz \
-compare_property_values @top_hits_formatted.txt \
-output_matched top_10.sdf \
-any
```

Congratulations! You just performed a virtual screen for the Kir2.1 ion channel. Test your understanding by trying to answer the following questions:

(1) Assuming the best-case scenario, what is the likelihood that your best predicted molecule is an active inhibitor?

(2) Extending from (1), how many molecules would you need to order before you would be confident that you have at least one hit?

(3) Using what you learned from Tutorials 1 and 2, identify the training set molecules that are most similar (based on a LargestCommonSubstructureTanimoto comparison) to each of your top 10 hits. Perform a substructure-based alignment. In what ways are the molecules similar? How are they different?

(4) Extending from (3), perform a flexible property-based alignment instead of a substructure-based alignment. Does this change how you think about the parts of the molecules that may contribute to their bioactivity?

(5) Which features were most salient to your model in making predictions? Don't worry, we did not cover this material in this material; however, the BCL provides tools to perform input sensitivity analysis. For more details, see the reference manuscripts[1,4].

Part 2: Multitask classification QSAR models for dopamine receptor D4 antagonists

In this section you will be training multitask classification models for dopamine receptor D4 (DRD4) orthosteric antagonists. Multitask learning with standard feed-forward fully connected neural networks can be readily achieved in the BCL. Around 2016, a couple important papers on multitask learning in QSAR/QSPR were published[5–7]. Arguably, the most critical findings were that **multitask learning only improves model performance if the result labels of molecules with similar feature space are correlated.**

Consider the following example. You have two proteins, A and B, and a training set of 10,000 molecules with labeled data such you have 5,000 molecules labeled only for A (a5000), 3,000 molecules only labeled for B (b3000), and 2,000 molecules that have labels for both (ab2000). Our primary task will be to predict the activity of molecules on A. Our secondary task will be to predict the activity of molecules on B. We hope that with multitask learning we can improve our predictions over independently training two models.

For us to benefit from multitask learning on b3000, the molecules in b3000 *that are similar in feature space* to molecules in a5000 must have correlated (positively or negatively) result labels to the molecules in a5000. The ab2000 molecules are a special case of similarity because the molecules that have activity labels for A and B are identical. Therefore, the result labels on ab2000 must also be correlated for a training benefit. **Supplementing the primary task with additional training samples in a secondary task will reduce model performance if the secondary task samples are similar in feature space to the primary task samples but the result labels are uncorrelated.** This is intuitive, as it essentially boils down to it being hard to train a neural network on a feature space for which there is little relationship between the features and the result label(s).

So how do we know when there is sufficient similarity and correlation to improve our model? The manuscripts cited above give some tips, but generally it can be challenging and requires trial-and-error. Here, we will demo multitask training in the BCL with a simpler challenge.

It turns out that we can perform multitasking on intrinsically correlated tasks to obtain predictions at multiple activity cutoff values without reducing performance (and, indeed, in some cases it improves performance). In Part 1, you built a single-task model to classify a molecule as active or inactive. Ideally, we could build a regression model that predicts the actual activity value (i.e., the output of the model is the activity itself in a physical unit such as pKd); however, these tasks require substantial quantities of data.

Instead, we can split the difference and perform "multi-tier" classification. In this approach, we train a model simultaneously on multiple classification labels. For example, we can say that we have three result labels: Active_10_nM, Active_100_nM, and Active_1000_nM. These indicate the thresholds at which we consider a molecule to be active. In this example, if you have a molecule with an activity of 50 nM, then our classification result would be (0, 1, 1). The result labels are intrinsically correlated and the samples for all result labels are identical. So, even if the model does not perform any better than three independent single-task models, it will not perform worse (though I do not think anybody has thoroughly investigated this in the case where the number of classification tasks approaches a regression-like task), and it tends to require less preparation and training time to build one multitask model than multiple single-task models.

We have only one difference in dataset preparation if we want to do multitask learning. Create a result label that contains multiple values.

```
Combine(Active_10_nM, Active_100_nM, Active_1000_nM)
```

That's it. I have already generated the feature datasets for you, so next you just need to train the model. See if you can write a model training command for the dataset in BCL_workusing "launch.py" as shown above

When the model is done training, you will see the same three plots that were automatically generated in Part 1 have been generated for each of the result labels.

(1) The DRD4 model was trained using 2000 iterations. Plot the logAUC value at each iteration for each of the result labels. After how many iterations do we stop seeing an improvement in logAUC?
(2) With the provided training data, train multitask classification models for DRD2, DRD3, and DRD5 using the number of iterations at which the DRD4 logAUC values plateaued. We will use these models in the next tutorial to predict the selectivity of DRD4 antagonists that we design with multi-component reaction-based design tools in the BCL.

Part 2 Extension (Optional): Multitasking across multiple targets

Make a new result object file containing the activity classification labels for all four proteins at the 100 nM cutoff. Combine all the SD files for each training set using `molecule:Unique`.

```
bcl.exe molecule:Unique \
-input_filenames *.sdf.gz \
-merge_descriptors \
-output all.labeled.unique.sdf.gz
```

Generate a dataset with the combined SDF. Train a neural network with the same settings that we used previously.

What is the logAUC of DRD4 at 100 nM? Is it better or worse than in our previous multitasking classification model? Are there any models that improved over the protein-specific models?

Part 3: Building a deep neural network for pKa prediction

In this section you will be building a small deep learning model to perform pKa prediction. The goal is to introduce you to a regression task and demonstrate additional syntax for training neural networks in the BCL. At the end of the tutorial we will also think about how to develop an algorithm to apply our model more effectively (i.e., instead of making the QSAR model the endpoint of an algorithm thinking about how to use it as a component of a more complex algorithm).

This tutorial will be hands-off. Instead of walking you through the individual steps, we will outline what needs to happen. In the input files we have provided training sets split into molecules with acidic groups and basic groups, respectively. We have provided an object file with which to create your feature dataset. Finally, we have provided a “config.ini” file similar to what was used in Part 2 that you can modify.

We will make two models: one for acids and one for bases. Here we go:

(1) Create a result object file similar to what we used for the single-task model in Part 1; however, make the result label “pKa”. You will see that the property “pKa” is cached on our SD files.

(2) Generate a dataset using the provided SD files, the feature labels, and the result labels.

(3) Randomize your dataset just because it’s always good to do it (even though it is less critical for regression models)

(4) Update your config file. We need the following:
   a. An objective function for regression tasks (AUC is no longer appropriate)
   b. A neural network that uses a rectified linear unit transfer function (soft-max so the bottom is at 0.05); contains 2 layers (“deep”) with 128 neurons in the first layer and 32 neurons in the second layer; dropout of 5% on the input layer, 25% on the first input layer, and 5% on the second hidden layer; and a learning rate (eta) of 0.25%. **Hint – look at some of the commented models that I left as templates**
   c. Train for 200 iterations
   d. 5x cross-validation

(5) Use the launch.py script to train your models

And that is that! Analyze your model output at the end. How does it look? Any thoughts on what you might want to change to improve the model further? Here is a sample of my output:

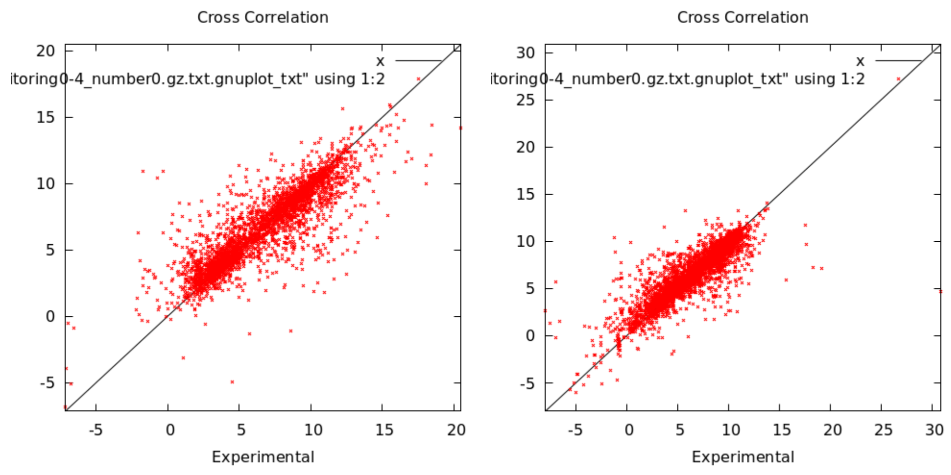

Figure 5. QSPR model training results from a sample pKa estimate task. Results are shown for the pKa prediction model for (A) acids (left) and (B) bases (right).

Okay, let's take a step back. You made a model that predicts pKa. Great. Is it actually useful? Can you run your molecules through this neural network and actually get a meaningful result? One way to evaluate this would be to build an applicability domain model, as mentioned in Part 2 (see Brown et al.[1]). For the sake of argument, we can go with a scenario in which the feature space of your candidate screening compounds are similar enough in feature space to your training molecules that they are considered within the domain of applicability. This is entirely plausible depending on how the feature space is described in each model. Does this mean that the model predicted pKa reliable?

I do not know because I have not benchmarked it; however, one primary concern might be that the training set is composed entirely of relatively small chemical fragments and generally are limited to one protonation site. You can use `molecule:Properties statistics` to collect basic information on the molecular weight, topological polar surface area, number of hydrogen bond donors and acceptors, etc. of all the molecules in your training set. If you were to compare that to test molecules, you may find that for the most part your mean values do not overlap well between the datasets.

The pKa prediction is performed across the entire molecule, but it corresponds to specific acidic and/or basic sites on a molecule. You learned in the previous tutorials about `molecule:Split`. We could, alternatively, split our test molecules into smaller fragments (e.g., using the Rigid splitter) and screen each of the fragments with our QSAR model.

Or we could split our molecules, filter out fragments without acidic or basic sites, predict the pKa of fragments containing basic sites with our basic model, and predict the pKa of fragments containing acidic sites with our acidic model, and then assign both values back to the original whole molecule. Going deeper down the rabbit hole, we could follow that fragment strategy, but then also mutate acidic/basic heteroatoms into e.g., carbon or hydrogen in our fragments. This would allow us to score each fragment in its native state and in a perturbed state to measure the impact of each atom on the predicted pKa of the fragment. By the time we are done, we could assign a pKa value to each acidic and basic atom. Indeed, this is highly analogous to a strategy we utilized for per-atom contributions to the predicted difference in binding affinity of congeneric inhibitors using a deep learning model[4].

These are all strategies that we could test using the command-line applications, but it would be tedious. What we could do instead is write a new descriptor at the C++ level. We regularly use machine learning models to assist as components of larger algorithms in descriptors. If you are interested in programming, think about what type of algorithm you would be interested in attempting. This is one example of a programming project that you could contribute to the BCL. As you will see again in Tutorial 4 and again in Tutorial 7, this type of descriptor would be very useful in both ligand-based virtual screening in the BCL and structure-based drug design in the BCL-Rosetta interface.

Congratulations! You have completed Tutorial 3 and survived my sales pitch. Looking forward to seeing you in Tutorial 4 on reaction-based drug design!

References

(1) Brown, B. P.; Vu, O.; Geanes, A. R.; Sandeepkumar, K.; Butkiewicz, M.; Lowe Jr, E. W.; Mueller, R.; Pape, R.; Mendenhall, J.; Meiler, J. Introduction to the BioChemical Library (BCL): An Application-Based Open-Source Toolkit for Integrated Cheminformatics and Machine Learning in Computer-Aided Drug Discovery.
(2) Butkiewicz, M.; Lowe, E. W.; Mueller, R.; Mendenhall, J. L.; Teixeira, P. L.; Weaver, C. D.; Meiler, J. Benchmarking Ligand-Based Virtual High-Throughput Screening with the PubChem Database. *Molecules* **2013**, *18* (1), 735–756. https://doi.org/10.3390/molecules18010735.
(3) Mendenhall, J.; Meiler, J. Improving Quantitative Structure-Activity Relationship Models Using Artificial Neural Networks Trained with Dropout. *J. Comput. Aided Mol. Des.* **2016**, *30* (2), 177–189. https://doi.org/10.1007/s10822-016-9895-2.
(4) Brown, B. P.; Mendenhall, J.; Geanes, A. R.; Meiler, J. General Purpose Structure-Based Drug Discovery Neural Network Score Functions with Human-Interpretable Pharmacophore Maps. *J. Chem. Inf. Model.* **2021**, *61* (2), 603–620. https://doi.org/10.1021/acs.jcim.0c01001.
(5) Dahl, G. E. Multi-Task Neural Networks for QSAR Predictions. *arXiv preprint arXiv:1406.1231* **2014**.
(6) Xu, Y.; Ma, J.; Liaw, A.; Sheridan, R. P.; Svetnik, V. Demystifying Multitask Deep Neural Networks for Quantitative Structure–Activity Relationships. *J. Chem. Inf. Model.* **2017**, *57* (10), 2490–2504. https://doi.org/10.1021/acs.jcim.7b00087.
(7) Ma, J.; Sheridan, R. P.; Liaw, A.; Dahl, G. E.; Svetnik, V. Deep Neural Nets as a Method for Quantitative Structure–Activity Relationships. *J. Chem. Inf. Model.* **2015**, *55* (2), 263–274. https://doi.org/10.1021/ci500747n.