Installing the Anaconda® environment

The Anaconda® distribution of Python installs with many popular data science packages, including those used in the k-DNN workflow. An advantage of using Anaconda® is the built-in capability to define environments that load only a program's necessary packages. Further, Anaconda® simplifies the installation of additional field-specific packages and the transfer of environments between computers. This workflow uses a pre-defined Anaconda® environment that users can easily replicate on their machines.

Windows: Navigate to the computer directory containing the un-zipped workflow files in Windows File Explorer. Right-click in this directory while holding down the shift key, and select "Open PowerShell window here." In the PowerShell window, execute the command `conda env create -f environment.yml`.

macOS and Linux: Open a new Terminal window, and go to the computer directory containing the un-zipped workflow files. Execute the command `conda env create -f environment.yml`.

Running the k-DNN workflow

The k-DNN workflow consists of several Python scripts designed to be executed from the command line using PowerShell (Windows users) or Terminal (macOS and Linux users). First, the previously installed Anaconda® environment must be activated by executing the following command: `conda activate kdnn_env`. The first script in the workflow is `model_training.py`, which trains the k-DNN. The script accepts six flagged arguments, in no required order:

1. The assay list filename without *.csv* ending (`-af`). This input value must correspond to a file present in the /data directory and contain the complete list of assays to be used in the modeling architecture as the first column and the layer that each assay should be placed into as the second column. Further, each of the listed assays should be a property in the training set *sdf*, with a value of 1 for active responses, 0 for inactive responses, and -1 for inconclusive/missing responses.
    a. The file used to train the k-DNN as shown in the publication was all_included_assay_information.csv, included in the /data directory here.
    b. The file used to train Control #2 in the publication (i.e., the network as trained without any bioassays known to encompass key events in the nuclear estrogen receptor adverse outcome pathway) was non_ER_assay_information.csv, included in the /data directory here.
2. The training set filename without *.sdf* ending (`-ds`). This input value must correspond to a file present in the /data directory. The file used to train the k-DNN in the publication of this work was training_set.sdf, included in the /data directory here.
3. The name of the *sdf* property containing the endpoint for modeling (`-ep`). In training_set.sdf, this property is "Bioactivity" and contains the training chemicals' *in vivo* uterotrophic activities as found in the NICEATM-curated database.
4. The chemical features with which to train the k-DNN (`-f`). The valid inputs for calculating binary chemical fingerprints for use in model training are: `FCFP6` to calculate functional connectivity fingerprints (FCFPs) with a bond radius of three, `MACCS` to calculate Molecular Access System keys, `rdkf` to calculate RDKit fingerprints, and `FCFP_ER` to calculate FCFPs with three extra

toxicophores related to estrogen receptor activity (steroid skeleton, phenol group, and diethylstilbestrol skeleton).

5. The name of the *sdf* property containing the training set chemicals' unique identifier (`-i`). The training_set.sdf chemicals' identifier is "Code" and is related to CAS number (e.g., for CAS 57-63-6, the Code would be C57636).
6. A comma-separated values string with the number of nodes in each hidden layer, in order (`-nn`). For the estrogen receptor k-DNN, the first hidden layer (i.e., the layer after the chemical fragment input layer) contains six nodes. The subsequent four layers contain 12, 6, 26, and 7 nodes, respectively. So, using training_set.sdf to train the estrogen receptor k-DNN would need the parameter: 6,12,6,26,7. The output layer is assumed to have one node.

The PowerShell or Terminal prompt to train the estrogen receptor k-DNN as published is, therefore: `python model_training.py -af all_included_assay_information -ds training_set -ep Bioactivity -f FCFP_ER -i Code -nn 6,12,6,26,7`.

This prompt generates a cache file in the /data directory containing model training information entitled training_set_Bioactivity_all_included_assay_information_FCFP_ER. It also generates six edge weight files in the /data directory, one for the input layer, four for each hidden layer, and one for the last hidden layer leading to the output node. The last hidden layer's edge weight file will be named training_set_Bioactivity_all_included_assay_information_FCFP_ER_output_W.csv. All other edge weight files will follow a similar format. For example, the input layer's edge weight file will be named training_set_Bioactivity_all_included_assay_information_FCFP_ER_layer_1_output_W.csv.

The `cross-validation.py` script does leave-one-out cross-validation with the training set. It takes the same six flagged arguments as `model_training.py`. The prompt to replicate the cross-validation process is: `python cross_validation.py -af all_included_assay_information -ds training_set -ep Bioactivity -f FCFP_ER -i Code -nn 6,12,6,26,7`. The resulting cross-validation predictions for each training set compound will be saved into one file called training_set_all_included_assay_information_ FCFP_ER_cv_predictions.csv.

To replicate the estrogen receptor k-DNN Control #2 as published, the prompt is: `python model_training.py -af non_ER_assay_information -ds training_set -ep Bioactivity -f FCFP_ER -i Code -nn 5,6,4,22,6`.

The `randomized_cross_validation.py` script replicates the estrogen receptor k-DNN Control #1 as published (i.e., training the k-DNN using 100 permutations of the *in vitro* bioassay and *in vivo* rodent uterotrophic training data). This script takes the same six flagged arguments as `model_training.py`. The prompt to replicate the randomized cross-validation process is: `python randomized_cross_validation.py -af all_included_assay_information -ds training_set -ep Bioactivity -f FCFP_ER -i Code -nn 6,12,6,26,7`. This prompt will generate two files. The first file, aucs_100_randomizations.csv, contains the area under the receiver operating curve (AUC) for each of the 100 randomizations. The second file, mean_roc_100_randomizations.csv, contains the average receiver operating curve (ROC) across the 100 randomizations for plotting.

Making predictions on new compounds after k-DNN training

The workflow script used to make predictions for new chemicals is `make_new_predictions.py`, which takes eight flagged arguments, in no required order:

1. The assay list filename without *.csv* ending used for model training (`-af`), as described previously.
2. The prediction set filename without *.sdf* ending (`-ds`).
3. The name of the *sdf* property containing the endpoint used for model training (`-ep`), as described previously.
4. The chemical features used to train the k-DNN (`-f`), as described previously.
5. The name of the *sdf* property containing the training set chemicals' unique identifier (`-i`), as described previously.
6. The name of the *sdf* property containing the prediction set chemicals' unique identifier (`-ip`).
7. A comma-separated values string with the number of nodes in each hidden layer, in order (`-nn`), as used for model training.
8. The training set filename without *.sdf* ending (`-ts`).