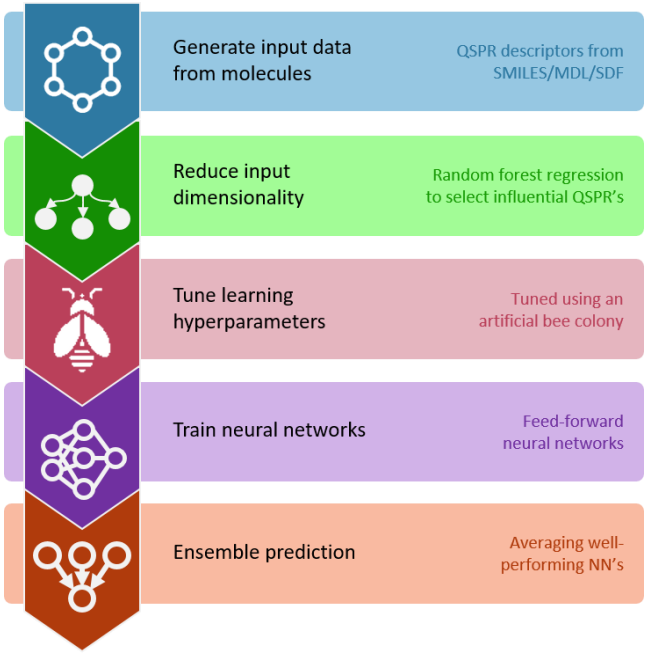
Getting Started with Predicting Fuel Properties

A Guide from the ECRL ANN Team

This guide will illustrate ECRL's current workflow for predicting fuel properties for potential next-generation biofuels. You will learn how to use ECNet, our machine learning toolkit, and the reasoning behind why we perform specific tasks. Specifically, you will learn how and why we:

- Use quantitative structure-property relationship (QSPR) values as "features" for molecules
- Limit the number of QSPR features to the most influential for a given property
- Tune neural network learning/architecture hyperparameters
- Train candidate neural networks to be evaluated with certain criteria
- Employ "ensemble models" to provide a final prediction



First and foremost, you will need to install the ECNet Python package. Make sure you have **Python 3.5** or **Python 3.6** installed, < 3.5 and 3.7 are currently not supported by all our dependency packages.

Assuming you have installed Python 3.5/3.6 with default settings (i.e. included PIP, added Python to your PATH), installing ECNet is as easy as opening a command prompt/terminal and executing:

```
pip install ecnet
```

This will install ECNet and its dependencies (TensorFlow, various optimization algorithm packages, a logging package) to your version of Python.

Refer to ECNet's documentation page for more detailed information on how specific functions in this guide's scripts operate: https://ecnet.readthedocs.io/en/latest/usage/quickstart.html

QSPR: what are they, why do we use them?

Quantitative structure-property relationship (QSPR) values, what we refer to as "descriptors", numerically represent the physical properties of molecules. They include basic geometric, electronic and steric properties.

Why do we care about these values *from a predictive model standpoint?* We care because the correct set* of descriptors will *never be the same* for two different molecules. Due to this variation, we can correlate the QSPR values of a molecule to a known experimental property value, such as Cetane Number (a measurement of how well a molecule performs in a diesel engine) or Yield Sooting Index (how much soot a molecule forms during combustion).

By supplying a machine learning algorithm with QSPR inputs and known experimental outputs, the algorithm can learn from hundreds of molecules and predict values for molecules it has never seen before. Being able to predict what the property of a molecule is before a costly and time-consuming synthesis and experiment will ultimately reduce the feedback loop inherent to fuel research.

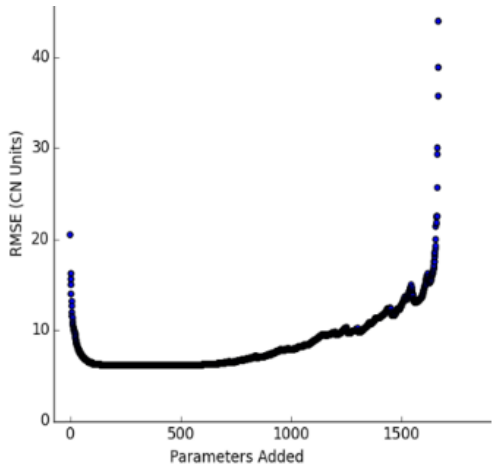
Included in this guide is an ECNet-formatted CSV database, "kv_model_v1.0_full.csv". This file contains 216 molecules, each with over 1600 QSPR descriptors and experimental data for kinematic viscosity. We will be using this database for the rest of the guide.

*Some descriptors do not contribute to property prediction, and can be detrimental to the prediction

There are a lot of descriptors… which ones should we use?

The software we currently use to generate QSPR descriptors provides us with over 1600 descriptors per molecule. You may think, "Why not use them all? More data is better!" Well, many descriptors have the same value for many (if not all) molecules we have. Take the descriptor "nN", the number of nitrogen atoms in a molecule, as an example. Most of our data is hydrocarbons and oxygenated compounds, where we rarely see a nitrogen atom – therefore, the QSPR value for this descriptor, across most molecules, will be zero. There is no variation for nN whatsoever, which will make it hard for a learning algorithm to tell molecules apart.

We ran an experiment where we retained the most-influential descriptors, one at a time, and evaluated the model's fitness after each descriptor was added:



As you can see, after about 500 descriptors are added, performance starts to degrade. In practice we have found the optimal number of descriptors (balancing accuracy and run-time) to be between 15 and 30.

Let's run an experiment to find the 15 most influential parameters for predicting kinetic viscosity from our "kv_model_v1.0_full.csv" database file. Locate the python script "limit_input_descriptors.py" in the "scripts" directory. Navigate to this directory in a command prompt/terminal, and execute:

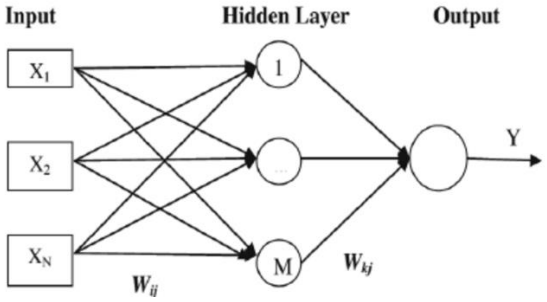
```
python limit_input_descriptors.py
```

The execution of this script will take a bit of time, the algorithm used (random forest regression from the scikit-learn Python package) is evaluating how well each descriptor correlates to a given value of kinetic viscosity.

Once the script has completed, you will have a database with 15 descriptors (kv_model_v1.0.csv).

Tuning neural network learning and architecture hyperparameters

Neural network architecture consists of an input layer, one or more hidden layers, and an output layer:



Input and output layers represent input data (QSPR descriptors) and output data (the property to predict), and remain constant for our application (15 inputs, 1 output). We typically work with two hidden layers for our neural networks, and the **number of neurons** in each hidden layer can vary (and be tuned to increase predictive accuracy).

In neural networks, each layer is connected to the next with a matrix of weights (matrix is of size L_{N_size} by L_{N+1_size}). During training, the error o f the neural network's prediction is fed back through the neural network (a process called backpropagation), and these weights are adjusted to decrease this error. The rate at which these weights change is called **learning rate**, another hyperparameter we are interested in tuning.

The optimization algorithm we use, AdamOptimizer, has a few more hyperparameters we are interested in tuning (**beta_1**, **beta_2**, **epsilon**, and **decay**). For more information on what these hyperparameters affect, refer to https://arxiv.org/abs/1412.6980.

Now that we have our limited database, "kv_model_v1.0.csv", we can tune these hyperparameters to best fit our selected QSPR descriptors to kinetic viscosity. Navigate to the scripts directory and execute:

```
python tune_hyperparameters.py
```

Tuning our hyperparameters involves creating dozens of neural networks to test for different hyperparameter permutations, so it will take a considerable amount of time. Once tuning is complete, the tuned hyperparameters will be saved to "scripts/config.yml" so they can be used for training.

Training candidate neural networks

Our training process involves creating "candidate" neural networks which we evaluate based on their ability to predict values for unseen data. We shuffle the training set (learning and validation sets) for each candidate neural network to ensure a selected candidate has learned from the best representation of data. Once all candidates are trained, we select the best performer(s) and save them to a .prj file.

To train your candidate neural networks and select the best performers, navigate to the scripts directory and execute:

```
python train_select_candidates.py
```

This script will train 25 candidates, each, for 5 pools of candidates, and select the best performer from each pool.

Ensemble predictions

At this point we have created 5x25 neural networks, selected the best 5 (one from each pool), and saved our project to a .prj file. Now we can predict values for our test set.

The individual predictions from the selected candidates are averaged to obtain a final prediction – this is known as an ensemble model. We employ the five selected models to ensure variation in training data and our models, and this reduces the error of the final prediction.

To open the .prj file, predict data for your test set and save results to a .csv file, navigate to the scripts directory and execute:

```
python use_project.py
```

This script will create a .csv file, "kv_test_results.csv", housing all molecules from the test set as well as predictions computed by our ensemble model. It will also determine statistics for the test set:

- RMSE
- Mean absolute error
- Median absolute error
- R-squared correlation coefficient

Visualizing predictions

Now that we know how to create and use an ECNet project to obtain predictions, let's visualize our predictions using a parity plot. A parity plot will show predicted values versus experimental values in relation to a 1:1 parity line. To create a plot for both training and test data, navigate to the scripts directory and execute:

```
python create_parity_plot.py
```

Summary

This guide has taught you ECRL's general workflow for predicting properties of molecules and how to utilize ECNet to accomplish this. If you have any questions regarding motivations for this research, contact Professor Hunter Mack (Hunter_Mack@uml.edu). If you have any questions related to software, contact Travis Kessler (Travis_Kessler@student.uml.edu).