MRS 2023 Handbook

Alexander van Teijlingen
alexander.van-teijlingen@strath.ac.uk

April 2023

This workshop is largely based on: van Teijlingen, A. & Tuttle, T. Beyond Tripeptides Two-Step Active Machine Learning for Very Large Data sets. J. Chem. Theory Comput., 2021.

1 Introduction

The aggregation propensity (AP) measured aggregation of monomers by comparing their solvent accessible surface area at the beginning of the simulation ($SASA_0$) and at the end ($SASA$).

$$AP = \frac{SASA_0}{SASA} \tag{1}$$

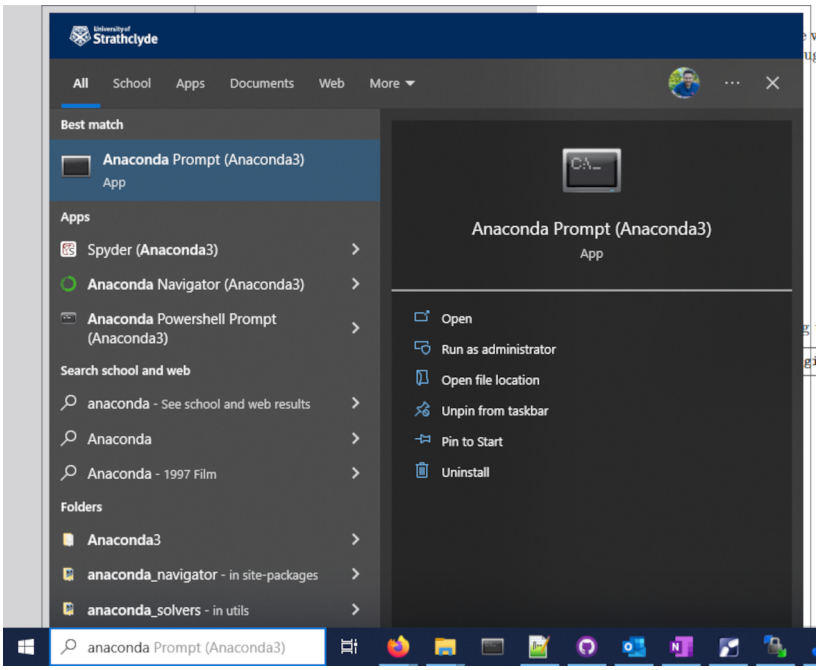
In this workshop we will aim to predict the AP score for the self-assembly of peptides by use of machine learning algorithms without having to spend hours running each individual simulation.

Molecular dynamics can be thought of as an iterative function that takes a set of parameters and provides outputs which we can derive AP from. Using machine learning we will also need a set of parameters from which a function can be derived which predicts the AP score.

Therefore, firstly we will need to generate these parameters, this will be covered in the next section.In order to work through this manual you will need Anaconda from which you can install the the following:

- Python 3
- Jupyter
- Numpy
- Scipy
- Pandas
- Scikit-learn
- Matplotlib

You can install the remaining packges be opening the Anaconda prompt:



and executing: conda install -c conda-forge numpy scipy pandas scikit-learn matplotlib

Then clone the git repository by executing:

```
git clone https://github.com/avanteijlingen/Tutorial-SB06
```

2 Parameter generation

There are many tools available for generating chemical parameters, to various levels of abstraction. These include Mordred, PaDEL & PyMOLSAR for chemical descriptors. Coulomb matrices for an electrotopological description, atomic environment vectors for local chemical environments and many others.

For this tutorial we will be using the Judred parameters as these are not very computationally expensive to generate for large datasets, and are rooted in basic chemistry.

Table 1: The 10 parameters generated by the Judred program with a brief description of each parameter as well as the physical mechanisms it is approximating.

Name	Description	Physical Mechanism
SP2	Number of SP2 carbon atoms	Entropic loss
NH2	Number of NH2/NH3 groups on the side chain(s)	Hydrogen bonding
MW	Molecular weight	Size
S	Number of sulphur atoms	Hydrogen bonding
log P WW	Wimley-White log P	Solubility
Z	Charge	Electrostatic interactions
RotRatio	Ratio of SP2 to SP3 carbon atoms	Relative Entropic loss
MaxASA	Maximum solvent accessible surface	Hydrophobic effect
Bulkiness	Sum of amino acid bulkiness	Size
OH	Number of OH groups (excluding backbone)	Hydrogen bonding

To generate these parameters open the Machine Learning/Judred.ipynb script and set L to 2 to generate parameters for Dipeptides. This will generate the file Dipeptides_Judred.csv which we will use in the next section. Make sure to run every cell of the notebook or the dataset will note be saved to a file!

3 Linear regression

Firstly we will look at fitting Equation 2 to Y = AP score. The Jupyter notebook for this section is in Machine Learning/MachineLearing_01.ipynb

$$y = m_1 x_1 + m_2 x_2 ... + b \tag{2}$$

Try changing the proportion of data in the train/testing/validation set by modifying test_size=X and seeing what affect this has on the accuracy of the predictions, think about why this is? If we have more training data the model should be able to make a better fit, however that also results in few validation samples which could artificially skew the result to appear better or worse than the model really is.

```
train_test_split(parameters, targets, test_size=0.33, shuffle=True)
```

Also try turning on/off shuffling of the data, since the dataset is ordered alphabetically there is order in the data (all FX dipeptides are bunched together, for example). By turning off shuffling we may lose many of our aggregators from the training/testing/validation set, skewing the result.

When you have ran this linear regression try projecting the data into non-linear space. This can be done by removing a=""" from code block 4. This will perform the following function on all of our Judred parameters, where X is the parameter upon which the function is operating:

```
def rbf(X, epsilon):
    return np.e ** -(epsilon*X)**2
```

We have chosen 0.2 as the value for ϵ, re-run the notebook to see what affect this has on the results. Finally, add epsilon to the hyperparameter optimization step. To do this we will need to add an addition for loop in code block 5:

```
#Scan values of epsilon between 0.1 and 1.0
#with 0.1 increments
for fit_intercept in [True, False]:
    for positive in [True, False]:
        for epsilon in np.arange(0.1, 1.1, 0.1):
```

We will also need to add this as a column in our BestHyperparameters dataframe, and store the epsilon value on each iteration:

```
BestHyperparameters = pandas.DataFrame(columns = [..., "epsilon", ...])
BestHyperparameters.loc[iteration] = [..., epsilon, ...]
```

3.1 Additional challenges

A gaussian radial basis function is just one way we can project from linear into non-linear space, try replacing the function 'rbf(X, epsilon)' with another non-linear projection such as a logarithm. What results do you get from the regression algorithm, better or worse?

```
def projection(X, p0, p1, ...):
    newXvalues = X .. p0 .. p1
    return newXvalues
```

4 More advanced machine learning algorithms

In this section we will look at implementing more advanced machine learning algorithms, namely Random Forest (RF), Multi-layer Perceptron (MLP, aka deep neural network (DNN)) and support vector machines (SVM). The Jupyter notebook for this section is in Machine Learning/Machine-Learing_02.ipynb

This time we will be using scikit-learn's GridSearchCV to fit hyperparameters now that we have an understanding of how this can be performed. Run through this notebook and attempt to modify the hyperparameter ranges for each of the three models, however, be aware that including too many values will take a very long time to process.
You will notice that apart from the great number and sensitivity of hyperparameters, more advanced machine learning models are still relatively easy to implement using the scikit-learn framework.

4.1 Additional challenges

In the re-weighting section, you will notice the lines:

```
SVMmodel.fit(X_train, y_train, sample_weight = y_train-0.5)
RFmodel.fit(X_train, y_train, sample_weight = y_train-0.5)
```

This assigns a greater degree of importance (weight) to peptides with higher AP scores in the fitting algorithm. We do this because there are less examples of these peptides and also that they are the most interesting. This particular implementation subtracts 0.5 from the AP, such that a non-assembler with AP of 1.0 has a weight of 0.5 and good aggregator with an AP of 2.5 has a weight 4 times larger of 2.0. How else might we define weights to favour different peptides? Try to implement another method in the notebook.

You may also want to try to implement other machine learning models, this can be done by the same process as each of the three we have gone through already. Import the model, do a hyperparameter optimization and test the RMSE of the model on the validation set.

5 Active learning

So far we have been looking at an example of having complete results for a dataset (dipeptides), however this can be extended to chemical spaces with far too many potential datapoints to reasonably be computed. We have prepared a partially complete AP score dataset of tetrapeptides which we will use to simulate searching the whole space. The Jupyter notebook for this section is in Machine Learning/MachineLearing_03.ipynb.

Firstly we will need to go back to Machine Learning/Judred.ipynb and set L to 4 to generate the parameters for all tetrapeptides. This algorithm operates by taking an initial starting point of a single peptide (GGGG for example) and iteratively making predictions of the dataset and taking the top 10 results back into the training set, this simulates actually running these peptide simulations as that is outside of the time frame for this workshop.

In this setup we are performing five iterations with the GGGG starting peptide, equal sample weighting, scoring by RMSE and using a support vector machine AI model. Each of these things can be modified to improve results, and I encourage you to play around with them to vary the results.

5.1 Additional challenges

You will notice the model tends to converge by always selecting for highly hydrophobic residues, try removing peptides from the dataset that have a log P greater than the mean

```
parameters = parameters[parameters["Judred_LogP WW"] < parameters["Judred_LogP WW"].mean()]
keep_index = [x for x in parameters.index if x in targets.index]
parameters = parameters.reindex(keep_index)
parameters = parameters.reindex(targets.index)
print(parameters)
print(targets)
```

To increase chemical diversity we can also slightly modify the predict AP scores (though do include the unmodified AP scores in the training set if they are selected) using some kind of a Monte Carlo function, try to implement this within the iterative loop.

```
def MC(AP):
    modified_AP = AP * (np.random.random()/10)
    return modified_AP
```