1 General

1.1 Theory

Regression model of some property, y, for some system, $\widetilde{\mathbf{X}}$:

$$y\left(\widetilde{\mathbf{X}}\right) = \sum_i \alpha_i \; K\left(\widetilde{\mathbf{X}}, \mathbf{X}_i\right) \tag{1}$$

E.g. Using Gaussian kernel function with Frobenius norm:

$$K_{ij} = K\left(\mathbf{X}_i, \mathbf{X}_j\right) = \exp\left(-\frac{\|\mathbf{X}_i - \mathbf{X}_j\|_2^2}{2\sigma^2}\right) \tag{2}$$

Regression coefficients are obtained through kernel matrix inversion and multiplication with reference labels

$$\boldsymbol{\alpha} = (\mathbf{K} + \lambda\mathbf{I})^{-1}\mathbf{y} \tag{3}$$

1.2 `QML` Code

Before starting the exercises, you will need the `QML` code which resides on github (`https://github.com/qmlcode/qml`). Carry out the following steps to install `QML` and get the data files (and solutions) to the exercises. On most Linux and Mac systems you can install `QML` from the official PyPI Python repository:

```
pip install qml --user -U
```

Follow this link for documentation `https://qmlcode.github.io/qml.html#`

2 Tutorial exercises

Clone the following GIT repository to access the necessary scripts and QM7 dataset (atomization energies and relaxed geometries at PBE0/def2-TZVP level of theory) for ∼7k GDB1-7 molecules.[1][2]

```
git clone https://github.com/qmlcode/tutorial.git
```

This also includes Python scripts with solutions to each exercise.

Exercise 2.1: Generating representations

In this exercise we use `QML` to generate the Coulomb matrix and Bag of bonds (BoB) representations.[3]

In `QML` data can be parsed via the `Compound` class, which stores data and generates representations in Numpy's ndarray format. If you run the code below, you will read in the file "qm7/0001.xyz" (a methane molecule) and generate a coulomb matrix representation (sorted by row-norm) and a BoB representation.

[1] Rupp et al, Phys Rev Letters, 2012.
[2] Ruddigkeit et al, J Chem Inf Model, 2012.
[3] Montavon et al, New J Phys, 2013.

```
import qml

# Create the compound object mol from the file qm7/0001.xyz which happens to be methane
mol = qml.Compound(xyz="qm7/0001.xyz")

# Generate and print a coulomb matrix for compound with 5 atoms
mol.generate_coulomb_matrix(size=5, sorting="row-norm")
print(mol.coulomb_matrix)

# Generate and print BoB bags for compound containing C and H
mol.generate_bob(size=5, asize={"C":2, "H":5})
print(mol.bob)
```

The representations are simply stored as 1D-vectors. Note the keyword `size` which is the largest number of atoms in a molecule occurring in test or training set. Additionally, the coulomb matrix can take a sorting scheme as keyword, and the BoB representations requires the specifications of how many atoms of a certain type to make room for in the representations.

Lastly, you can print the following properties which is read from the XYZ file:

```
# Print other properties stored in the object
print(mol.coordinates)
print(mol.atomtypes)
print(mol.nuclear_charges)
print(mol.name)
print(mol.unit_cell)
```

See the online SOURCE DOCUMENTATION for this and further details `qml.representations`-module here: `https://qmlcode.github.io/qml.html#module-qml.representations`

Exercise 2.2: Generating Kernel matrix

In this exercise we generate a Gaussian kernel matrix, $\mathbf{K}$, using the representations, $\mathbf{X}$, which are generated similarly to the example in the previous exercise:

$$K_{ij} = \exp\left(-\frac{\|\mathbf{X}_i - \mathbf{X}_j\|_2^2}{2\sigma^2}\right) \tag{4}$$

`QML` supplies functions to generate the most basic kernels (E.g. Gaussian, Laplacian). In the exercise below, we calculate a Gaussian kernel for the QM7 dataset.

In order to save time you can import the entire QM7 dataset as `Compound` objects from the file `tutorial_data.py` found in the GitHub repository.

```
# Import QM7, already parsed to QML
from tutorial_data import compounds

from qml.kernels import gaussian_kernel

# For every compound generate a coulomb matrix or BoB
for mol in compounds:

    mol.generate_coulomb_matrix(size=23, sorting="row-norm")
    # mol.generate_bob(size=23, asize={"O":3, "C":7, "N":3, "H":16, "S":1})

# Make a big 2D array with all the representations
X = np.array([mol.coulomb_matrix for mol in compounds])
# X = np.array([mol.bob for mol in compounds])

# Print all representations
print(X)

# Run on only a subset of the first 100 (for speed)
```

```
X = X[:100]

# Define the kernel width
sigma = 1000.0

# K is also a Numpy array
K = gaussian_kernel(X, X, sigma)

# Print the kernel
print K
```

Exercise 2.3: Kernel-Ridge Regression

With the kernel matrix and representations sorted out in the previous two exercise, we can now solve the $\boldsymbol{\alpha}$ regression coefficients (see Eq. 5):

$$\boldsymbol{\alpha} = (\mathbf{K} + \lambda \mathbf{I})^{-1} \mathbf{y} \tag{5}$$

One of the most efficient ways of solving this equation is using a Cholesky-decomposition. `QML` includes a function named `cho_solve` to do this via the math module `qml.math`.

In this step it is convenient to only use a subset of the full dataset as training data (see below). The following builds on the code from the previous step. To save time, you can import the PBE0/def2-TZVP atomization energies for the QM7 dataset from the file `tutorial_data.py`. This has been sorted to match the ordering of the representations generated in the previous exercise.
Extend your code from the previous step with the code below:

```
from qml.math import cho_solve
from tutorial_data import energy_pbe0

# Assign 1000 first molecules to the training set
X_training = X[:1000]
Y_training = energy_pbe0[:1000]

sigma = 4000.0
K = gaussian_kernel(X_training, X_training, sigma)
print(K)

# Add a small lambda to the diagonal of the kernel matrix
K[np.diag_indices_from(K)] += 1e-8

# Use the built-in Cholesky-decomposition to solve
alpha = cho_solve(K, Y_training)

print(alpha)
```

Exercise 2.4: Making predictions

With the $\boldsymbol{\alpha}$ regression coefficients from the previous step, we have (successfully) trained the machine, and we are now ready to do predictions for other compounds. This is done using the following equation:

$$y\left(\widetilde{\mathbf{X}}\right) = \sum_i \alpha_i \; K\left(\widetilde{\mathbf{X}}, \mathbf{X}_i\right) \tag{6}$$

In this step we further divide the dataset into a training and a test set. Try using the last 1000 entries as test set.

```
# Assign 1000 last molecules to the test set
X_test = X[-1000:]
Y_test = energy_pbe0[-1000:]

# calculate a kernel matrix between test and training data, using the same sigma
Ks = gaussian_kernel(X_test, X_training, sigma)

# Make the predictions
Y_predicted = np.dot(Ks, alpha)

# Calculate mean-absolute-error (MAE):
print np.mean(np.abs(Y_predicted - Y_test))
```

Exercise 2.5: Learning curves:

Repeat the prediction from Exercise 2.4 with training set sizes of 1000, 2000, and 4000 molecules. Note the MAE for every training size. Plot a learning curve of the MAE versus the training set size.

Generate a learning curve for the Gaussian and Laplacian kernels, as well using the coulomb matrix and bag-of-bonds representations.

Which combination gives the best learning curve? Note you will have to adjust the kernel width (sigma) underway.

Exercise 2.6: Delta learning:

A powerful technique in machine learning is the delta learning approach. Instead of predicting the PBE0/def2-TZVP atomization energies, we shall try to predict the difference between DFTB3 (a semi-empirical quantum method) and PBE0 atomization energies. Instead of importing the `energy_pbe0` data, you can import the `energy_delta` and use this instead

```
from tutorial_data import energy_delta

Y_training = energy_delta[:1000]
Y_test = energy_delta[-1000:]
```

Finally re-draw one of the learning curves from the previous exercise, and note how the prediction improves.