Computational Methods - Lesson 11 (Luxembourg, 09.12.2020)

Machine Learning II

Valentin Vassilev Galindo*[,a], Matteo Barborini*[,b], Prof. Alexandre Tkatchenko*

Theoretical Chemical Physics group

Faculté des Sciences, des Technologies et de Médecine - Département Physique et sciences des matériaux

*Campus Limpertsberg, Université du Luxembourg - 162 A, avenue de la Faïencerie, L-1511 Luxembourg

[a]valentin.vassilev@uni.lu, [b]matteo.barborini@uni.lu

1. Kernel Ridge Regression

1.1. Linear regression. Linear regression is given by the following equation

$$y = \sum_{i=1}^{N} w_i X_i \tag{1}$$

where N is equal to the number of features, w_i is the i-th coefficient (or parameter), and X_i is the i-th feature. This equation gives the predicted value, provided we know the values of all w_i coefficients.
To obtain the value of the w_i coefficients one needs to minimize the value of the loss function, which, for this case it will be defined with the equation

$$\mathcal{L}(\mathbf{w}) = \sum_{j=0}^{M} (y_j - \mathbf{X}_j \mathbf{w})^2 \tag{2}$$

where M is equal to the number of training points, y_j is the target of the j-th observation, $\boldsymbol{w}$ is the vector containing all coefficients, and $\boldsymbol{X}_j$ is a vector containing the features of the j-th observation.
In a linear regression model the objective is to minimize the value of the loss function. Lower the loss function value, better the linear regression model. Usually, to decrease the value of loss function, we could increase the number of features in our model. If we increase the number of features each observation in our training set has, our model will start fitting the training data set well and the loss function value starts decreasing. But, with increase in number of features, the hypothesis becomes a higher order polynomial equation. This will lead to overfitting of the data.
For avoiding overfitting, one can control the value of the weights $\boldsymbol{w}$ since when overfitting happens, the values of the coefficients tend to become huge. For controlling the values of $\boldsymbol{w}$, we need to introduce a new term into the loss function: the so-called regularization term. One example of a regularization process is the L2 regularized linear regression, or ridge regression, which will be discussed below.

1.2. Ridge regression. Ridge regression is used to quantify the overfitting of the data through measuring the magnitude of coefficients. In ridge regression one needs to find the best trade-off between i) how well the model fits data, and ii) the magnitude of the coefficients. Here we are going to discuss primal and dual ridge regression.

1.2.1. Primal ridge regression.

In primal ridge regression our loss function in Eq. 2 becomes

$$\mathcal{L}(\boldsymbol{w}) = \sum_{j=0}^{M} (y_j - \boldsymbol{X}_j \boldsymbol{w})^2 + \lambda ||\boldsymbol{w}||^2 \tag{3}$$

where the first term measures the fit of the model and the second term, the magnitude of the coefficients. $||\boldsymbol{w}||^2$ is the euclidean norm of $\boldsymbol{w}$ and λ is a regularization parameter that allows balancing the fit of data and the magnitude of coefficients.
Eq. 3 in matrix notation takes the form

$$\mathcal{L}(\mathbf{w}) = ||\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}||^2 + \lambda ||\boldsymbol{w}||^2 \tag{4}$$

Taking the gradient of the above equation

$$\frac{\partial \mathcal{L}(\boldsymbol{w})}{\partial \boldsymbol{w}} = 2\lambda \boldsymbol{w} - 2\boldsymbol{X}^T\boldsymbol{y} + 2\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{w} \tag{5}$$

where $\boldsymbol{X}^T$ is the transpose of $\boldsymbol{X}$. The optimality condition is then met when the gradient of the loss function is equal to zero. So, Eq. 5 leads to

$$(\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I})\boldsymbol{w} = \boldsymbol{X}^T\boldsymbol{y} \tag{6}$$

where $\boldsymbol{I}$ is the identity matrix. The $\boldsymbol{w}$ coefficients can be obtained with

$$\boldsymbol{w} = (\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I})^{-1}\boldsymbol{X}^T\boldsymbol{y} \tag{7}$$

1.2.2. Dual ridge regression. The mathematical procedure to perform dual ridge regression is the same as in primal ridge regression until Eq. 6. Now, instead of Eq. 7, one solves for $\boldsymbol{w}$ with

$$\boldsymbol{w} = \lambda^{-1}(\boldsymbol{X}^T\boldsymbol{y} - \boldsymbol{X}^T\boldsymbol{X}\boldsymbol{w}) = \lambda^{-1}\boldsymbol{X}^T(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}) \tag{8}$$

If we define

$$\boldsymbol{\alpha} = \lambda^{-1}(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}) \tag{9}$$

and then solve for $\boldsymbol{\alpha}$ considering that $\boldsymbol{w} = \boldsymbol{X}^T\boldsymbol{\alpha}$

$$\boldsymbol{\alpha} = (\boldsymbol{X}\boldsymbol{X}^T + \lambda\boldsymbol{I})^{-1}\boldsymbol{y} \tag{10}$$

where $\boldsymbol{X}\boldsymbol{X}^T$ is the inner product between data points ($< \boldsymbol{X_i}, \boldsymbol{X_j} >$).
The new coefficients found with Eq. 10 lead to a hypothesis different to the one in linear regression. Now, predicted targets are not obtained considering only the features of the observations we want to predict, but we use an inner product of the features of the new datum with the features of all the training points. Thus, our predictor function becomes

$$y' = \sum_{i=0}^{M} \alpha_i < \boldsymbol{X}_i, \boldsymbol{X'} > \tag{11}$$

where y' and $\boldsymbol{X'}$ are the predicted target and the features of the observation we want to predict, respectively.

1.3. Kernel Ridge Regression. A linear hypothesis is often not enough for obtaining a robust ML model. So, most of the times we would prefer to apply a certain non-linear function ϕ to our features $\boldsymbol{X}$. In such a case, Eq. 11 takes the form

$$y' = \sum_{i=0}^{M} \alpha_i < \phi(\boldsymbol{X_i}), \phi(\boldsymbol{X'}) > \tag{12}$$

The non-linear mapping ϕ to a higher dimension is computationally very expensive if one performs it explicitly. However, there are certain type of functions, called kernels, that can implicitly map our data. A Kernel K is related to a mathematical space, $\mathcal{H}_k$, such that applying K to two feature vectors, $\boldsymbol{X}_1$ and $\boldsymbol{X}_2$, is equivalent to projecting these feature vectors into $\mathcal{H}_k$ by some projection function, ϕ and taking their inner product there

$$K(\boldsymbol{X_1}, \boldsymbol{X_2}) =< \phi(\boldsymbol{X_1}), \phi(\boldsymbol{X_2}) >_{\mathcal{H}_k} \tag{13}$$

The above is the basis for the famous "kernel trick": If the input features are involved in the equation of a statistical model only in the form of inner products then we can replace the inner products in the equation with calls to the kernel function and the result is as if we had projected the input features into a higher dimensional space and taken their inner product there. But we never need to perform the actual projection.
So, by substituting Eq. 13 into Eq. 12 and Eq. 10 we end up with the equations for doing kernel ridge regression

$$y' = \sum_{i=0}^{M} \alpha_i K(\boldsymbol{X_i}, \boldsymbol{X'}) \tag{14}$$

$$\boldsymbol{\alpha} = (\boldsymbol{K} + \lambda \boldsymbol{I})^{-1} \boldsymbol{y} \tag{15}$$

1.3.1. Kernel functions. There are several types/families of kernel functions. Here, however, we will only introduce the most widely used.

1. **Gaussian Kernel**. They are an example of radial basis function kernels and are sometimes called radial basis kernels. The value of a radial basis function kernel depends only on the distance between the argument vectors, rather than their location. Such kernels are also termed stationary. Gaussian kernels take a hyperparameter γ as input. Their functional form is

$$K(\boldsymbol{X_1}, \boldsymbol{X_2}) = e^{-\gamma ||\boldsymbol{X_1} - \boldsymbol{X_2}||^2} \tag{16}$$

   where $||\boldsymbol{X_1} - \boldsymbol{X_2}||^2$ is the Euclidean distance between two given feature vectors.

2. **Laplacian Kernel**. They are also radial basis functions. They take a hyperparameter σ as input. Their functional form is

$$K(\boldsymbol{X_1}, \boldsymbol{X_2}) = e^{-\frac{||\boldsymbol{X_1} - \boldsymbol{X_2}||}{\sigma}} \tag{17}$$

   where $||\boldsymbol{X_1} - \boldsymbol{X_2}||$ is the Manhattan distance between two given feature vectors.

3. **Polynomial Kernel**. They are an example of non-stationary kernels. So these kernels will assign different values to pairs of points that share the same distance, based on their values. Hyperparameter values must be non-negative. They take hyperparameters α,*c*, and *d* as input. Their functional form is

$$K(\boldsymbol{X_1}, \boldsymbol{X_2}) = (\alpha \boldsymbol{X_1^T} \boldsymbol{X_2} + c)^d \tag{18}$$

2. Scikit-learn Module

Scikit-learn is a library for doing Machine Learning (ML) in Python. It is built on NumPy, SciPy, and matplotlib. It has all sort of simple and efficient tools for predictive data analysis.
To import the library one simply needs to write ***import sklearn*** in the python script. However, since you do not need all its modules and methods at a single time, a more specific import command is preferred. For instance, if you are interested only in the ***cluster*** module, you should only write ***from sklearn import cluster***.
A complete documentation can be found at `https://scikit-learn.org/stable/index.html`.

2.1. Important Basic Tools.

2.1.1. Feature Scaling. We saw in our last lecture that feature scaling is an important preprocessing step in most ML algorithms. It is performed to handle highly varying magnitudes, values or units. Scikit-learn has a useful class within its ***preprocessing*** module, ***StandardScaler***. This class performs a standardization (i.e. removes the mean and sets to unity the variance) on the data to scale all features.
The class includes the methods ***fit*** (for computing the mean and standard deviation to be used for later scaling), ***transform*** (to perform standardization by centering and scaling), and ***fit_transform*** (to both fit and transform).
To use it one should add to the python script the line ***from sklearn.preprocessing import StandardScaler***.

2.1.2. Cross-validation. It is important to avoid biases towards certain types of samples within our datasets. One crucial procedure to reliably assess the performance of a given model is cross-validation. In cross-validation, we train/test the ML models several times using different training/testing samples from our dataset. Particularly, the so-called k-fold cross-validation splits the dataset into k different subsets, where usually k-1 subsets are used for training and the remaining one is used for testing. This training/testing procedure is repeated k times, so that each subset has been used once as test set. Scikit-learn can perform this procedure using the class ***KFold*** within its ***model_selection*** module.
One only needs to set the number of splits and select between random and non-random splits. Then, one can get the indexes of the samples that will be used for training and testing by applying the ***split*** method. To use it one needs to write ***from sklearn.model_selection import KFold***.

2.1.3. Grid search. Last lecture we introduced the concept of hyperparameter in ML. The hyperparameters are all the parameters which can be arbitrarily set by the user before starting training and they are not directly learnt within the estimators. Hyperparameters define higher level concepts about the model such as complexity, or capacity to learn. Although they are not directly linked to the learning process, it is important to find an optimal set of hyperparameters. With such an optimal set, our model can improve its performance.
A way to select the best set of hyperparameters is to first test different models on a subset of our dataset (normally referred to as validation set) and then select the model that presents the lowest error. Scikit-learn contains a class called ***GridSearchCV***, which allows to automatically perform the search of hyperparameters. One only needs to provide the estimator (i.e. the function or method for which we want to optimize the hyperparameters) and a dictionary containing the names of the hyperparameters as keys and the list of the parameters to try as values. The class contains the methods ***fit*** (to fit the estimator with the training data) and ***predict*** (to predict on the estimator with the best found parameters).To use it one needs to write ***from sklearn.model_selection import GridSearchCV***.

2.1.4. Learning curve. In ML, we call a learning curve the plot of errors as a function of training set size. Usually, the performance of different ML models is compared by means of this curve. One can, of course, iteratively increase the amount of training points and retrain the estimator at each step. Other option is to use the class ***learning_curve*** within the ***model_selection*** module. The class takes as input the array of features, the array of targets and an array with the different training set sizes that we want to evaluate. To use it one needs to write ***from sklearn.model_selection import learning_curve***.

2.1.5. Regression metrics. Metrics are functions that are used to measure the performance of our ML models. Specifically, regression metrics are employed for regression tasks. The mean squared error (MSE; introduced in our previous lecture) and the mean absolute error (MAE) are examples of such metrics. Scikit-learn contains these functions within its ***metrics*** module. For instance, one can use the MSE and MAE with ***from sklearn.metrics import mean_squared_error, mean_absolute_error***. Both classes take as input the arrays of true and predicted targets.

2.2. Kernel Ridge Regression in Scikit-learn.

One can perform Kernel Ridge Regression (KRR) in Scikit-learn following two approaches:

1. Using the Kernel functions available within the module ***sklearn.metrics.pairwise***. In this module, one can find the classes for the gaussian kernel (***rbf_kernel***), the laplacian kernel (***laplacian_kernel***), and the polynomial kernel (***polynomial_kernel***). All these classes take as arguments the two arrays for which one wants to compute the kernel, and the hyperparameters one would like to use. This approach requires the user to code by himself the routines for obtaining the coefficients α (training), any hyperparameter search (validation), and the prediction of the unknown observations (testing).

2. Using the ***KernelRidge*** class within the ***kernel_ridge*** module. This class takes as argument the name of the kernel we would like to use, as well as all the hyperparameters that are needed for the selected kernel. This approach does not need much input from the user because the ***KernelRidge*** class contains the methods ***fit*** and ***predict***, which allow for an "automatic" training and testing, respectively. Moreover, this approach can be easily coupled with other utilities inside scikit-learn for doing grid searchs of hyperparameters or plotting learning curves.

3. References

1. Singhal, A. *Machine Learning: Ridge Regression in Detail*. Recovered from: `https://towardsdatascience.com/machine-learning-ridge-regression-in-detail-76787a2f8e2d`

2. Ashcroft, M. *Basics & Kernel Regression*. Recovered from: `https://www.futurelearn.com/info/courses/advanced-machine-learning/0/steps/49560`

3. Bennet, K. *Kernel Ridge Regression*. Recovered from: `http://homepages.rpi.edu/~bennek/class/mds/lecture/lecture6-06.pdf`