First and second days: Exercise set 1

Data Analysis and Machine Learning for Nuclear Physics

Jul 1, 2020

Day one and two exercises

*

Exercise 1: Getting started

The first exercise here is of a mere technical art. We want you to have

- git as a version control software and to establish a user account on a provider like GitHub. Other providers like GitLab etc are equally fine.
- Install various Python packages

We will make extensive use of Python as programming language and its myriad of available libraries. You will find IPython/Jupyter notebooks invaluable in your work. You can run **R** codes in the Jupyter/IPython notebooks, with the immediate benefit of visualizing your data. You can also use compiled languages like C++, Rust, Fortran etc if you prefer. The focus in these lectures will be on Python.

If you have Python installed (we recommend Python3) and you feel pretty familiar with installing different packages, we recommend that you install the following Python packages via **pip** as

1. pip install numpy scipy matplotlib ipython scikit-learn sympy pandas pillow

For **Tensorflow**, we recommend following the instructions in the text of Aurelien Geron, Hands-On Machine Learning with Scikit-Learn and TensorFlow, O'Reilly

We will come back to **tensorflow** later.

For Python3, replace **pip** with **pip3**.

For OSX users we recommend, after having installed Xcode, to install **brew**. Brew allows for a seamless installation of additional software via for example

1. brew install python3

For Linux users, with its variety of distributions like for example the widely popular Ubuntu distribution, you can use **pip** as well and simply install Python as

1. sudo apt-get install python3 (or python for Python2.7)

If you don't want to perform these operations separately and venture into the hassle of exploring how to set up dependencies and paths, we recommend two widely used distrubutions which set up all relevant dependencies for Python, namely

- Anaconda,

which is an open source distribution of the Python and R programming languages for large-scale data processing, predictive analytics, and scientific computing, that aims to simplify package management and deployment. Package versions are managed by the package management system **conda**.

- Enthought canopy

is a Python distribution for scientific and analytic computing distribution and analysis environment, available for free and under a commercial license.

We recommend using **Anaconda**.

*

Exercise 2: Our first Python encounter

This exercise has as its aim to write a small program which reads in data from a **csv** file on the equation of state for dense nuclear matter. The file is localized at `https://github.com/mhjensen/MachineLearningMSU-FRIB2020/blob/master/doc/pub/Regression/ipynb/datafiles/EoS.csv`. Thereafter you will have to set up the design matrix $\boldsymbol{X}$ for the n datapoints and a polynomial of degree 3. The steps are:

- Write a Python code which reads the in the above mentioned file.
- Use for example **pandas** to order your data and find out how many data points there are.
- Set thereafter up the design matrix with dimensionality $n \times p$ where $p = 4$ and where you have defined a polynomial of degree $p - 1 = 3$. Print the matrix and check that the numbers are correct.

We recommend looking at the examples in the regression slides.

*

Exercise 3: making your own data and exploring scikit-learn

We will generate our own dataset for a function $y(x)$ where $x \in [0,1]$ and defined by random numbers computed with the uniform distribution. The function y is a quadratic polynomial in x with added stochastic noise according to the normal distribution $\mathcal{N}(\prime, \infty)$. The following simple Python instructions define our x and y values (with 100 data points).

1. Write your own code (following the examples under the regression slides) for computing the parametrization of the data set fitting a second-order polynomial.
2. Use thereafter **scikit-learn** (see again the examples in the regression slides) and compare with your own code.
3. Using scikit-learn, compute also the mean square error, a risk metric corresponding to the expected value of the squared (quadratic) error defined as

$$MSE(\hat{y},\hat{\tilde{y}}) = \frac{1}{n}\sum_{i=0}^{n-1}(y_i - \tilde{y}_i)^2,$$

and the R^2 score function. If $\tilde{\hat{y}}_i$ is the predicted value of the $i-th$ sample and y_i is the corresponding true value, then the score R^2 is defined as

$$R^2(\hat{y},\tilde{\hat{y}}) = 1 - \frac{\sum_{i=0}^{n-1}(y_i - \tilde{y}_i)^2}{\sum_{i=0}^{n-1}(y_i - \bar{y})^2},$$

where we have defined the mean value of $\hat{y}$ as

$$\bar{y} = \frac{1}{n}\sum_{i=0}^{n-1} y_i.$$

You can use the functionality included in scikit-learn. If you feel for it, you can use your own program and define functions which compute the above two functions. Discuss the meaning of these results. Try also to vary the coefficient in front of the added stochastic noise term and discuss the quality of the fits.

Solution. The code here is an example of where we define our own design matrix and fit parameters β.

*

Exercise 4: mean values and variances in linear regression

This exercise deals with various mean values ad variances in linear regression method (here it may be useful to look up chapter 3, equation (3.8) of Trevor Hastie, Robert Tibshirani, Jerome H. Friedman, The Elements of Statistical Learning, Springer).

The assumption we have made is that there exists a function $f(\boldsymbol{x})$ and a normal distributed error $\boldsymbol{\varepsilon} \sim \mathcal{N}(0, \sigma^2)$ which describes our data

$$\boldsymbol{y} = f(\boldsymbol{x}) + \boldsymbol{\varepsilon}$$

We then approximate this function with our model from the solution of the linear regression equations (ordinary least squares OLS), that is our function f is approximated by $\tilde{\boldsymbol{y}}$ where we minimized $(\boldsymbol{y} - \tilde{\boldsymbol{y}})^2$, with

$$\tilde{\boldsymbol{y}} = \boldsymbol{X}\boldsymbol{\beta}.$$

The matrix $\boldsymbol{X}$ is the so-called design matrix.

aragraph!paragraph>paragraph>-0.5em

a) Show that the expectation value of $\boldsymbol{y}$ for a given element i

$$\mathbb{E}(y_i) = \mathbf{X}_{i,*}\,\beta,$$

and that its variance is

$$\mathrm{Var}(y_i) = \sigma^2.$$

Hence, $y_i \sim \mathcal{N}(\mathbf{X}_{i,*}\,\boldsymbol{\beta}, \sigma^2)$, that is $\boldsymbol{y}$ follows a normal distribution with mean value $\boldsymbol{X}\boldsymbol{\beta}$ and variance σ^2.

Solution. We can calculate the expectation value of $\boldsymbol{y}$ for a given element i

$$\mathbb{E}(y_i) = \mathbb{E}(\mathbf{X}_{i,*}\,\boldsymbol{\beta}) + \mathbb{E}(\varepsilon_i) \;\; = \;\; \mathbf{X}_{i,*}\,\beta,$$

while its variance is

$$\begin{aligned}
\mathrm{Var}(y_i) = \mathbb{E}\{[y_i - \mathbb{E}(y_i)]^2\} \;\; &= \;\; \mathbb{E}(y_i^2) - [\mathbb{E}(y_i)]^2 \\
&= \mathbb{E}[(\mathbf{X}_{i,*}\,\beta + \varepsilon_i)^2] - (\mathbf{X}_{i,*}\,\boldsymbol{\beta})^2 \\
&= \mathbb{E}[(\mathbf{X}_{i,*}\,\boldsymbol{\beta})^2 + 2\varepsilon_i \mathbf{X}_{i,*}\,\boldsymbol{\beta} + \varepsilon_i^2] - (\mathbf{X}_{i,*}\,\beta)^2 \\
&= (\mathbf{X}_{i,*}\,\boldsymbol{\beta})^2 + 2\mathbb{E}(\varepsilon_i)\mathbf{X}_{i,*}\,\boldsymbol{\beta} + \mathbb{E}(\varepsilon_i^2) - (\mathbf{X}_{i,*}\,\boldsymbol{\beta})^2 \\
&= \mathbb{E}(\varepsilon_i^2) \;\; = \;\; \mathrm{Var}(\varepsilon_i) \;\; = \;\; \sigma^2.
\end{aligned}$$

Hence, $y_i \sim \mathcal{N}(\mathbf{X}_{i,*}\,\boldsymbol{\beta}, \sigma^2)$, that is $\boldsymbol{y}$ follows a normal distribution with mean value $\boldsymbol{X}\boldsymbol{\beta}$ and variance σ^2 (not be confused with the singular values of the SVD).

aragraph!paragraph>paragraph>-0.5em

b) With the OLS expressions for the parameters $\boldsymbol{\beta}$ show that

$$\mathbb{E}(\boldsymbol{\beta}) = \boldsymbol{\beta}.$$

Solution.

$$\mathbb{E}(\boldsymbol{\beta}) = \mathbb{E}[(\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y}] = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbb{E}[\mathbf{Y}] = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{X}\boldsymbol{\beta} = \boldsymbol{\beta}.$$

This means that the estimator of the regression parameters is unbiased.

aragraph!paragraph>paragraph>-0.5em

c) Show finally that the variance of $\boldsymbol{\beta}$ is

$$\mathrm{Var}(\boldsymbol{\beta}) \;=\; \sigma^2\,(\mathbf{X}^T\mathbf{X})^{-1}.$$

Solution. The variance of $\boldsymbol{\beta}$ is

$$\begin{aligned}
\mathrm{Var}(\boldsymbol{\beta}) &= \mathbb{E}\{[\boldsymbol{\beta} - \mathbb{E}(\boldsymbol{\beta})][\boldsymbol{\beta} - \mathbb{E}(\boldsymbol{\beta})]^T\} \\
&= \mathbb{E}\{[(\mathbf{X}^T\mathbf{X})^{-1}\,\mathbf{X}^T\mathbf{Y} - \boldsymbol{\beta}]\,[(\mathbf{X}^T\mathbf{X})^{-1}\,\mathbf{X}^T\mathbf{Y} - \boldsymbol{\beta}]^T\} \\
&= (\mathbf{X}^T\mathbf{X})^{-1}\,\mathbf{X}^T\,\mathbb{E}\{\mathbf{Y}\,\mathbf{Y}^T\}\,\mathbf{X}\,(\mathbf{X}^T\mathbf{X})^{-1} - \boldsymbol{\beta}\,\boldsymbol{\beta}^T \\
&= (\mathbf{X}^T\mathbf{X})^{-1}\,\mathbf{X}^T\,\{\mathbf{X}\,\boldsymbol{\beta}\,\boldsymbol{\beta}^T\,\mathbf{X}^T + \sigma^2\}\,\mathbf{X}\,(\mathbf{X}^T\mathbf{X})^{-1} - \boldsymbol{\beta}\,\boldsymbol{\beta}^T \\
&= \boldsymbol{\beta}\,\boldsymbol{\beta}^T + \sigma^2\,(\mathbf{X}^T\mathbf{X})^{-1} - \boldsymbol{\beta}\,\boldsymbol{\beta}^T \;=\; \sigma^2\,(\mathbf{X}^T\mathbf{X})^{-1},
\end{aligned}$$

where we have used that $\mathbb{E}(\mathbf{Y}\mathbf{Y}^T) = \mathbf{X}\,\boldsymbol{\beta}\,\boldsymbol{\beta}^T\,\mathbf{X}^T + \sigma^2\,\mathbf{I}_{nn}$. From $\mathrm{Var}(\boldsymbol{\beta}) = \sigma^2\,(\mathbf{X}^T\mathbf{X})^{-1}$, one obtains an estimate of the variance of the estimate of the j-th regression coefficient: $\boldsymbol{\sigma}^2(\hat{\beta}_j) = \boldsymbol{\sigma}^2\sqrt{[(\mathbf{X}^T\mathbf{X})^{-1}]_{jj}}$. This may be used to construct a confidence interval for the estimates.

In a similar way, we can obtain analytical expressions for say the expectation values of the parameters $\boldsymbol{\beta}$ and their variance when we employ Ridge regression, allowing us again to define a confidence interval.

*

Exercise 5: Playing with nuclear masses

Finally, try now to write your own code (you can use the example the nuclear masses in the lecture slides on Regression and Getting started from Day1, that reads in the nuclear masses and compute the proton separation energies, the two-neutron and two-proton separation energies and finally the shell gaps for selected nuclei.

Finally, try to compute the Q-values for $\beta-$ decay for selected nuclei.

Solution. Let us study the Q values associated with the removal of one or two nucleons from a nucleus. These are conventionally defined in terms of the one-nucleon and two-nucleon separation energies. With the functionality in **pandas**, two to three lines of code will allow us to plot the separation energies. The neutron separation energy is defined as

$$S_n = -Q_n = BE(N,Z) - BE(N-1,Z),$$

and the proton separation energy reads

$$S_p = -Q_p = BE(N,Z) - BE(N,Z-1).$$

The two-neutron separation energy is defined as

$$S_{2n} = -Q_{2n} = BE(N, Z) - BE(N - 2, Z),$$

and the two-proton separation energy is given by

$$S_{2p} = -Q_{2p} = BE(N, Z) - BE(N, Z - 2).$$

Using say the neutron separation energies (alternatively the proton separation energies)

$$S_n = -Q_n = BE(N, Z) - BE(N - 1, Z),$$

we can define the so-called energy gap for neutrons (or protons) as

$$\Delta S_n = BE(N, Z) - BE(N - 1, Z) - (BE(N + 1, Z) - BE(N, Z)) ,$$

or

$$\Delta S_n = 2BE(N, Z) - BE(N - 1, Z) - BE(N + 1, Z).$$

This quantity can in turn be used to determine which nuclei could be interpreted as magic or not. For protons we would have

$$\Delta S_p = 2BE(N, Z) - BE(N, Z - 1) - BE(N, Z + 1).$$

To calculate say the neutron separation we need to multiply our masses with the nucleon number A. The example here is for the neutron separation energies for the oxygen isotopes. Note the simple function we use to compute the neutron separation energies

If we want another isotope we need simply to change the Z value. For isotones, we fix simply the neutron number. Furthermore, if we wish to compute say the two-neutron separation energies of the oyxgen isotopes we need simply to write Note the +2 in the function **diff(+2)**! Easy, isn't it? It is easy to change to two-proton separation energies. The full example here is for the neutron separation energies.

Expectation value and variance for β

We can also calculate the variance

Resampling methods

Resampling methods, basic overview

With all these analytical equations for both the OLS and Ridge regression, we will now outline how to assess a given model. This will lead us to a discussion of the so-called bias-variance tradeoff (see below) and so-called resampling methods.

One of the quantities we have discussed as a way to measure errors is the mean-squared error (MSE), mainly used for fitting of continuous functions. Another choice is the absolute error.

In the discussions below we will focus on the MSE and in particular since we will split the data into test and training data, we discuss the

1. prediction error or simply the **test error**, where we have a fixed training set and the test error is the MSE arising from the data reserved for testing. We discuss also the

2. training error $\mathrm{Err}_{\mathrm{Train}}$, which is the average loss over the training data.

As our model becomes more and more complex, more of the training data tends to used. The training may thence adapt to more complicated structures in the data. This may lead to a decrease in the bias (see below for code example) and a slight increase of the variance for the test error. For a certain level of complexity the test error will reach a minimum, before starting to increase again. The training error reaches a saturation.

Train and test, an example

Bringing back our Equation of State data

Resampling methods: Jackknife and Bootstrap

Two famous resampling methods are the **independent bootstrap** and **the jackknife**.

The jackknife is a special case of the independent bootstrap. Still, the jackknife was made popular prior to the independent bootstrap. And as the popularity of the independent bootstrap soared, new variants, such as **the dependent bootstrap**.

The Jackknife and independent bootstrap work for independent, identically distributed random variables. If these conditions are not satisfied, the methods will fail.

Resampling methods: Bootstrap

Bootstrapping is a nonparametric approach to statistical inference that substitutes computation for more traditional distributional assumptions and asymptotic results. Bootstrapping offers a number of advantages:

1. The bootstrap is quite general, although there are some cases in which it fails.

2. Because it does not require distributional assumptions (such as normally distributed errors), the bootstrap can provide more accurate inferences when the data are not well behaved or when the sample size is small.

3. It is possible to apply the bootstrap to statistics with sampling distributions that are difficult to derive, even asymptotically.

4. It is relatively simple to apply the bootstrap.

Resampling methods: Bootstrap steps

The independent bootstrap works like this:

1. Draw with replacement n numbers for the observed variables $\boldsymbol{x} = (x_1, x_2, \cdots, x_n)$.
2. Define a vector $\boldsymbol{x}^*$ containing the values which were drawn from $\boldsymbol{x}$.
3. Using the vector $\boldsymbol{x}^*$ compute the estimate (parameter) $\widehat{\theta}^*$ by evaluating $\widehat{\theta}$ under the observations $\boldsymbol{x}^*$.
4. Repeat this process k times.

When you are done, you can draw a histogram of the relative frequency of the estimator/parameter $\widehat{\theta}^*$. This is your estimate of the probability distribution $p(x)$. Using this probability distribution you can estimate any statistics thereof. In principle you never draw the histogram of the relative frequency of $\widehat{\theta}^*$. Instead you use the estimators corresponding to the statistic of interest. For example, if you are interested in estimating the variance of $\widehat{\theta}$, apply the etsimator $\widehat{\sigma}^2$ to the values $\widehat{\theta}^*$.

Code example for the Bootstrap method

The following code starts with a Gaussian distribution with mean value $\mu = 100$ and variance $\sigma = 15$. We use this to generate the data used in the bootstrap analysis. The bootstrap analysis returns a data set after a given number of bootstrap operations (as many as we have data points). This data set consists of estimated mean values for each bootstrap operation. The histogram generated by the bootstrap method shows that the distribution for these mean values is also a Gaussian, centered around the mean value $\mu = 100$ but with standard deviation $\sigma/\sqrt{n}$, where n is the number of bootstrap samples (in this case the same as the number of original data points). The value of the standard deviation is what we expect from the central limit theorem.

Various steps in cross-validation

When the repetitive splitting of the data set is done randomly, samples may accidently end up in a fast majority of the splits in either training or test set. Such samples may have an unbalanced influence on either model building or prediction evaluation. To avoid this k-fold cross-validation structures the data splitting. The samples are divided into k more or less equally sized exhaustive and mutually exclusive subsets. In turn (at each split) one of these subsets plays the role of the test set while the union of the remaining subsets constitutes the training set. Such a splitting warrants a balanced representation of each sample in both training and test set over the splits. Still the division into the k subsets involves a degree of randomness. This may be fully excluded when choosing $k = n$. This particular case is referred to as leave-one-out cross-validation (LOOCV).

How to set up the cross-validation for Ridge and/or Lasso

- Define a range of interest for the penalty parameter.
- Divide the data set into training and test set comprising samples $\{1, \ldots, n\} \setminus i$ and $\{i\}$, respectively.
- Fit the linear regression model by means of OLS, or Ridge or Lasso estimations for each λ in the grid using the training set, and the corresponding estimate of the error variance $\boldsymbol{\sigma}^2_{-i}(\lambda)$, as

$$\boldsymbol{\beta}_{-i}(\lambda) = (\boldsymbol{X}^T_{-i,*}\boldsymbol{X}_{-i,*} + \lambda \boldsymbol{I}_{pp})^{-1}\boldsymbol{X}^T_{-i,*}\boldsymbol{y}_{-i}$$

- Evaluate the prediction performance of these models on the test set by $\log\{L[y_i, \boldsymbol{X}_{i,*}; \boldsymbol{\beta}_{-i}(\lambda), \boldsymbol{\sigma}^2_{-i}(\lambda)]\}$. Or, by the prediction error $|y_i - \boldsymbol{X}_{i,*}\boldsymbol{\beta}_{-i}(\lambda)|$, the relative error, the error squared or the R2 score function.
- Repeat the first three steps such that each sample plays the role of the test set once.
- Average the prediction performances of the test sets at each grid point of the penalty bias/parameter. It is an estimate of the prediction performance of the model corresponding to this value of the penalty parameter on novel data. It is defined as

$$\frac{1}{n}\sum_{i=1}^{n} \log\{L[y_i, \mathbf{X}_{i,*}; \boldsymbol{\beta}_{-i}(\lambda), \boldsymbol{\sigma}^2_{-i}(\lambda)]\}.$$

Cross-validation in brief

For the various values of k

1. shuffle the dataset randomly.
2. Split the dataset into k groups.
3. For each unique group:
   (a) Decide which group to use as set for test data
   (b) Take the remaining groups as a training data set
   (c) Fit a model on the training set and evaluate it on the test set
   (d) Retain the evaluation score and discard the model
4. Summarize the model using the sample of model evaluation scores

Code Example for Cross-validation and k-fold Cross-validation

The code here uses Ridge regression with cross-validation (CV) resampling and k-fold CV in order to fit a specific polynomial.

How to decide upon the best model?

The bias-variance tradeoff

We will discuss the bias-variance tradeoff in the context of continuous predictions such as regression. However, many of the intuitions and ideas discussed here also carry over to classification tasks. Consider a dataset $\mathcal{L}$ consisting of the data $\mathbf{X}_{\mathcal{L}} = \{(y_j, \boldsymbol{x}_j), j = 0 \dots n-1\}$.

Let us assume that the true data is generated from a noisy model

$$\boldsymbol{y} = f(\boldsymbol{x}) + \boldsymbol{\epsilon}$$

where ϵ is normally distributed with mean zero and standard deviation σ^2.

In our derivation of the ordinary least squares method we defined then an approximation to the function f in terms of the parameters $\boldsymbol{\beta}$ and the design matrix $\boldsymbol{X}$ which embody our model, that is $\tilde{\boldsymbol{y}} = \boldsymbol{X}\boldsymbol{\beta}$.

Thereafter we found the parameters $\boldsymbol{\beta}$ by optimizing the mean-squared error via the so-called cost function

$$C(\boldsymbol{X}, \boldsymbol{\beta}) = \frac{1}{n}\sum_{i=0}^{n-1}(y_i - \tilde{y}_i)^2 = \mathbb{E}\left[(\boldsymbol{y} - \tilde{\boldsymbol{y}})^2\right].$$

We can rewrite this as

$$\mathbb{E}\left[(\boldsymbol{y} - \tilde{\boldsymbol{y}})^2\right] = \frac{1}{n}\sum_{i}(f_i - \mathbb{E}\left[\tilde{\boldsymbol{y}}\right])^2 + \frac{1}{n}\sum_{i}(\tilde{y}_i - \mathbb{E}\left[\tilde{\boldsymbol{y}}\right])^2 + \sigma^2.$$

The three terms represent the square of the bias of the learning method, which can be thought of as the error caused by the simplifying assumptions built into the method. The second term represents the variance of the chosen model and finally the last terms is variance of the error $\boldsymbol{\epsilon}$.

To derive this equation, we need to recall that the variance of $\boldsymbol{y}$ and $\boldsymbol{\epsilon}$ are both equal to σ^2. The mean value of $\boldsymbol{\epsilon}$ is by definition equal to zero. Furthermore, the function f is not a stochastics variable, idem for $\tilde{\boldsymbol{y}}$. We use a more compact notation in terms of the expectation value

$$\mathbb{E}\left[(\boldsymbol{y} - \tilde{\boldsymbol{y}})^2\right] = \mathbb{E}\left[(\boldsymbol{f} + \boldsymbol{\epsilon} - \tilde{\boldsymbol{y}})^2\right],$$

and adding and subtracting $\mathbb{E}\left[\tilde{\boldsymbol{y}}\right]$ we get

$$\mathbb{E}\left[(\boldsymbol{y} - \tilde{\boldsymbol{y}})^2\right] = \mathbb{E}\left[(\boldsymbol{f} + \boldsymbol{\epsilon} - \tilde{\boldsymbol{y}} + \mathbb{E}\left[\tilde{\boldsymbol{y}}\right] - \mathbb{E}\left[\tilde{\boldsymbol{y}}\right])^2\right],$$

which, using the abovementioned expectation values can be rewritten as

$$\mathbb{E}\left[(\boldsymbol{y} - \tilde{\boldsymbol{y}})^2\right] = \mathbb{E}\left[(\boldsymbol{y} - \mathbb{E}\left[\tilde{\boldsymbol{y}}\right])^2\right] + \mathrm{Var}\left[\tilde{\boldsymbol{y}}\right] + \sigma^2,$$

that is the rewriting in terms of the so-called bias, the variance of the model $\tilde{\boldsymbol{y}}$ and the variance of $\boldsymbol{\epsilon}$.

Understanding what happens

Summing up

The bias-variance tradeoff summarizes the fundamental tension in machine learning, particularly supervised learning, between the complexity of a model and the amount of training data needed to train it. Since data is often limited, in practice it is often useful to use a less-complex model with higher bias, that is a model whose asymptotic performance is worse than another model because it is easier to train and less sensitive to sampling noise arising from having a finite-sized training dataset (smaller variance).

The above equations tell us that in order to minimize the expected test error, we need to select a statistical learning method that simultaneously achieves low variance and low bias. Note that variance is inherently a nonnegative quantity, and squared bias is also nonnegative. Hence, we see that the expected test MSE can never lie below $Var(\epsilon)$, the irreducible error.

What do we mean by the variance and bias of a statistical learning method? The variance refers to the amount by which our model would change if we estimated it using a different training data set. Since the training data are used to fit the statistical learning method, different training data sets will result in a different estimate. But ideally the estimate for our model should not vary too much between training sets. However, if a method has high variance then small changes in the training data can result in large changes in the model. In general, more flexible statistical methods have higher variance.

You may also find this recent article of interest.

The same example but now with cross-validation

Cross-validation with Ridge

Applying Regression Analysis to the Ising Model

The Ising model

The one-dimensional Ising model with nearest neighbor interaction, no external field and a constant coupling constant J is given by

$$H = -J \sum_{k}^{L} s_k s_{k+1}, \tag{1}$$

where $s_i \in \{-1, 1\}$ and $s_{N+1} = s_1$. The number of spins in the system is determined by L. For the one-dimensional system there is no phase transition.

We will look at a system of $L = 40$ spins with a coupling constant of $J = 1$. To get enough training data we will generate 10000 states with their respective energies.

Here we use ordinary least squares regression to predict the energy for the nearest neighbor one-dimensional Ising model on a ring, i.e., the endpoints wrap

around. We will use linear regression to fit a value for the coupling constant to achieve this.

Reformulating the problem to suit regression

A more general form for the one-dimensional Ising model is

$$H = -\sum_{j}^{L}\sum_{k}^{L} s_j s_k J_{jk}. \tag{2}$$

Here we allow for interactions beyond the nearest neighbors and a state dependent coupling constant. This latter expression can be formulated as a matrix-product

$$\boldsymbol{H} = \boldsymbol{X}J, \tag{3}$$

where $X_{jk} = s_j s_k$ and J is a matrix which consists of the elements $-J_{jk}$. This form of writing the energy fits perfectly with the form utilized in linear regression, that is

$$\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}, \tag{4}$$

We split the data in training and test data as discussed in the previous example

Linear regression

In the ordinary least squares method we choose the cost function

$$C(\boldsymbol{X}, \boldsymbol{\beta}) = \frac{1}{n}\left\{(\boldsymbol{X}\boldsymbol{\beta} - \boldsymbol{y})^T(\boldsymbol{X}\boldsymbol{\beta} - \boldsymbol{y})\right\}. \tag{5}$$

We then find the extremal point of C by taking the derivative with respect to $\boldsymbol{\beta}$ as discussed above. This yields the expression for $\boldsymbol{\beta}$ to be

$$\boldsymbol{\beta} = \frac{\boldsymbol{X}^T\boldsymbol{y}}{\boldsymbol{X}^T\boldsymbol{X}},$$

which immediately imposes some requirements on $\boldsymbol{X}$ as there must exist an inverse of $\boldsymbol{X}^T\boldsymbol{X}$. If the expression we are modeling contains an intercept, i.e., a constant term, we must make sure that the first column of $\boldsymbol{X}$ consists of 1. We do this here

Singular Value decomposition

Doing the inversion directly turns out to be a bad idea since the matrix $\boldsymbol{X}^T\boldsymbol{X}$ is singular. An alternative approach is to use the **singular value decomposition**. Using the definition of the Moore-Penrose pseudoinverse we can write the equation for $\boldsymbol{\beta}$ as

$$\boldsymbol{\beta} = \boldsymbol{X}^+\boldsymbol{y},$$

where the pseudoinverse of $\boldsymbol{X}$ is given by

$$\boldsymbol{X}^+ = \frac{\boldsymbol{X}^T}{\boldsymbol{X}^T\boldsymbol{X}}.$$

Using singular value decomposition we can decompose the matrix $\boldsymbol{X} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$, where $\boldsymbol{U}$ and $\boldsymbol{V}$ are orthogonal(unitary) matrices and $\boldsymbol{\Sigma}$ contains the singular values (more details below). where $X^+ = V\Sigma^+U^T$. This reduces the equation for ω to

$$\boldsymbol{\beta} = \boldsymbol{V}\boldsymbol{\Sigma}^+\boldsymbol{U}^T\boldsymbol{y}. \tag{6}$$

Note that solving this equation by actually doing the pseudoinverse (which is what we will do) is not a good idea as this operation scales as $\mathcal{O}(n^3)$, where n is the number of elements in a general matrix. Instead, doing QR-factorization and solving the linear system as an equation would reduce this down to $\mathcal{O}(n^2)$ operations.

When extracting the J-matrix we need to make sure that we remove the intercept, as is done here

A way of looking at the coefficients in J is to plot the matrices as images.

It is interesting to note that OLS considers both $J_{j,j+1} = -0.5$ and $J_{j,j-1} = -0.5$ as valid matrix elements for J. In our discussion below on hyperparameters and Ridge and Lasso regression we will see that this problem can be removed, partly and only with Lasso regression.

In this case our matrix inversion was actually possible. The obvious question now is what is the mathematics behind the SVD?

The one-dimensional Ising model

Let us bring back the Ising model again, but now with an additional focus on Ridge and Lasso regression as well. We repeat some of the basic parts of the Ising model and the setup of the training and test data. The one-dimensional Ising model with nearest neighbor interaction, no external field and a constant coupling constant J is given by

$$H = -J\sum_{k}^{L} s_k s_{k+1}, \tag{7}$$

where $s_i \in \{-1, 1\}$ and $s_{N+1} = s_1$. The number of spins in the system is determined by L. For the one-dimensional system there is no phase transition.

We will look at a system of $L = 40$ spins with a coupling constant of $J = 1$. To get enough training data we will generate 10000 states with their respective energies.

A more general form for the one-dimensional Ising model is

$$H = -\sum_{j}^{L}\sum_{k}^{L} s_j s_k J_{jk}. \tag{8}$$

Here we allow for interactions beyond the nearest neighbors and a more adaptive coupling matrix. This latter expression can be formulated as a matrix-product on the form

$$H = XJ, \tag{9}$$

where $X_{jk} = s_j s_k$ and J is the matrix consisting of the elements $-J_{jk}$. This form of writing the energy fits perfectly with the form utilized in linear regression, viz.

$$\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}. \tag{10}$$

We organize the data as we did above

We will do all fitting with **Scikit-Learn**,

When extracting the J-matrix we make sure to remove the intercept And then we plot the results The results perfectly with our previous discussion where we used our own code.

Ridge regression

Having explored the ordinary least squares we move on to ridge regression. In ridge regression we include a **regularizer**. This involves a new cost function which leads to a new estimate for the weights $\boldsymbol{\beta}$. This results in a penalized regression problem. The cost function is given by

$$C(\boldsymbol{X}, \boldsymbol{\beta}; \lambda) = (\boldsymbol{X}\boldsymbol{\beta} - \boldsymbol{y})^T(\boldsymbol{X}\boldsymbol{\beta} - \boldsymbol{y}) + \lambda\boldsymbol{\beta}^T\boldsymbol{\beta}. \tag{11}$$

LASSO regression

In the **Least Absolute Shrinkage and Selection Operator** (LASSO)-method we get a third cost function.

$$C(\boldsymbol{X}, \boldsymbol{\beta}; \lambda) = (\boldsymbol{X}\boldsymbol{\beta} - \boldsymbol{y})^T(\boldsymbol{X}\boldsymbol{\beta} - \boldsymbol{y}) + \lambda\sqrt{\boldsymbol{\beta}^T\boldsymbol{\beta}}. \tag{12}$$

Finding the extremal point of this cost function is not so straight-forward as in least squares and ridge. We will therefore rely solely on the function “Lasso“ from **Scikit-Learn**.

It is quite striking how LASSO breaks the symmetry of the coupling constant as opposed to ridge and OLS. We get a sparse solution with $J_{j,j+1} = -1$.

Performance as function of the regularization parameter

We see how the different models perform for a different set of values for λ.

We see that LASSO reaches a good solution for low values of λ, but will "wither" when we increase λ too much. Ridge is more stable over a larger range of values for λ, but eventually also fades away.

Finding the optimal value of λ

To determine which value of λ is best we plot the accuracy of the models when predicting the training and the testing set. We expect the accuracy of the training set to be quite good, but if the accuracy of the testing set is much lower this tells us that we might be subject to an overfit model. The ideal scenario is an accuracy on the testing set that is close to the accuracy of the training set.

From the above figure we can see that LASSO with $\lambda = 10^{-2}$ achieves a very good accuracy on the test set. This by far surpasses the other models for all values of λ.