Project1

September 2, 2025

Project 1 on Machine Learning, deadline October 6 (midnight), 2025

Data Analysis and Machine Learning FYS-STK3155/FYS4155, University of Oslo, Norway

Date: **September 2**

Preamble: Note on writing reports, using reference material, AI and other tools

We want you to answer the three different projects by handing in reports written like a standard scientific/technical report. The links at https://github.com/CompPhysics/MachineLearning/tree/master/doc/Projects contain more information. There you can find examples of previous reports, the projects themselves, how we rade reports etc. How to write reports will also be discussed during the various lab sessions. Please do ask us if you are in doubt.

When using codes and material from other sources, you should refer to these in the bibliography of your report, indicating wherefrom you for example got the code, whether this is from the lecture notes, softwares like Scikit-Learn, TensorFlow, PyTorch or other sources. These sources should always be cited correctly. How to cite some of the libraries is often indicated from their corresponding GitHub sites or websites, see for example how to cite Scikit-Learn at https://scikit-learn.org/dev/about.html.

We enocurage you to use tools like ChatGPT or similar in writing the report. If you use for example ChatGPT, please do cite it properly and include (if possible) your questions and answers as an addition to the report. This can be uploaded to for example your website, GitHub/GitLab or similar as supplemental material.

If you would like to study other data sets, feel free to propose other sets. What we have proposed here are mere suggestions from our side. If you opt for another data set, consider using a set which has been studied in the scientific literature. This makes it easier for you to compare and analyze your results. Comparing with existing results from the scientific literature is also an essential element of the scientific discussion. The University of California at Irvine with its Machine Learning repository at https://archive.ics.uci.edu/ml/index.php is an excellent site to look up for examples and inspiration. Kaggle.com is an equally interesting site. Feel free to explore these sites. When selecting other data sets, make sure these are sets used for regression problems (not classification).

Regression analysis and resampling methods

The main aim of this project is to study in more detail various regression methods, including Ordinary Least Squares (OLS) reegression, Ridge regression and LASSO regression. In addition to the scientific part, in this course we want also to give you an experience in writing scientific reports.

We will study how to fit polynomials to specific one-dimensional functions (feel free to replace the suggested function with more complicated ones).

We will use Runge's function (see https://en.wikipedia.org/wiki/Runge%27s_phenomenon for a discussion). The one-dimensional function we will study is

$$f(x) = \frac{1}{1+25x^2}.$$

Our first step will be to perform an OLS regression analysis of this function, trying out a polynomial fit with an x dependence of the form $[x, x^2, \dots]$. You can use a uniform distribution to set up the arrays of values for $x \in [-1, 1]$, or alternatively use a fixed step size. Thereafter we will repeat many of the same steps when using the Ridge and Lasso regression methods, introducing thereby a dependence on the hyperparameter (penalty) λ.

We will also include bootstrap as a resampling technique in order to study the so-called **bias-variance tradeoff**. After that we will include the so-called cross-validation technique.

Part a : Ordinary Least Square (OLS) for the Runge function

We will generate our own dataset for abovementioned function Runge(x) function with $x \in [-1, 1]$. You should explore also the addition of an added stochastic noise to this function using the normal distribution $N(0, 1)$.

Write your own code (using for example the pseudoinverse function **pinv** from **Numpy**) and perform a standard **ordinary least square regression** analysis using polynomials in x up to order 15 or higher. Explore the dependence on the number of data points and the polynomial degree.

Evaluate the mean Squared error (MSE)

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

and the R^2 score function. If $\tilde{\boldsymbol{y}}_i$ 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

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

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

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

Plot the resulting scores (MSE and R^2) as functions of the polynomial degree (here up to polymial degree 15). Plot also the parameters θ as you increase the order of the polynomial. Comment your results.

Your code has to include a scaling/centering of the data (for example by subtracting the mean value), and a split of the data in training and test data. For the scaling you can either write your own code or use for example the function for splitting training data provided by the library **Scikit-Learn** (make sure you have installed it). This function is called *train_test_split*. **You should present a critical discussion of why and how you have scaled or not scaled the data**.

It is normal in essentially all Machine Learning studies to split the data in a training set and a test set (eventually also an additional validation set). There is no explicit recipe for how much data should be included as training data and say test data. An accepted rule of thumb is to use approximately 2/3 to 4/5 of the data as training data.

You can easily reuse the solutions to your exercises from week 35. See also the lecture slides from week 35 and week 36.

On scaling, we recommend reading the following section from the scikit-learn software description, see https://scikit-learn.org/stable/auto_examples/preprocessing/plot_all_scaling.html#plot-all-scaling-standard-scaler-section.

Part b: Adding Ridge regression for the Runge function

Write your own code for the Ridge method as done in the previous exercise. The lecture notes from week 35 and 36 contain more information. Furthermore, the results from the exercise set from week 36 is something you can reuse here.

Perform the same analysis as you did in the previous exercise but now for different values of λ. Compare and analyze your results with those obtained in part a) with the OLS method. Study the dependence on λ.

Part c: Writing your own gradient descent code

Replace now the analytical expressions for the optimal parameters $\boldsymbol{\theta}$ with your own gradient descent code. In this exercise we focus only on the simplest gradient descent approach with a fixed learning rate (see the exercises from week 37 and the lecture notes from week 36).

Study and compare your results from parts a) and b) with your gradient descent approch. Discuss in particular the role of the learning rate.

Part d: Including momentum and more advanced ways to update the learning the rate

We keep our focus on OLS and Ridge regression and update our code for the gradient descent method by including **momentum**, **ADAgrad**, **RMSprop** and **ADAM** as methods fro iteratively updating your learning rate. Discuss the results and compare the different methods applied to the one-dimensional Runge function. The lecture notes from week 37 contain several examples on how to implement these methods.

Part e: Writing our own code for Lasso regression

LASSO regression (see lecture slides from week 36 and week 37) represents our first encounter with a machine learning method which cannot be solved through analytical expressions (as in OLS and Ridge regression). Use the gradient descent methods you developed in parts c) and d) to solve the LASSO optimization problem. You can compare your results with the functionalities of **Scikit-Learn**.

Discuss (critically) your results for the Runge function from OLS, Ridge and LASSO regression using the various gradient descent approaches.

Part f: Stochastic gradient descent

Our last gradient step is to include stochastic gradient descent using the same methods to update the learning rates as in parts c-e). Compare and discuss your results with and without stochastic gradient and give a critical assessment of the various methods.

Part g: Bias-variance trade-off and resampling techniques

Our aim here is to study the bias-variance trade-off by implementing the **bootstrap** resampling technique. **We will only use the simpler ordinary least squares here**.

With a code which does OLS and includes resampling techniques, we will now discuss the bias-variance trade-off in the context of continuous predictions such as regression. However, many of the intuitions and ideas discussed here also carry over to classification tasks and basically all Machine Learning algorithms.

Before you perform an analysis of the bias-variance trade-off on your test data, make first a figure similar to Fig. 2.11 of Hastie, Tibshirani, and Friedman. Figure 2.11 of this reference displays only the test and training MSEs. The test MSE can be used to indicate possible regions of low/high bias and variance. You will most likely not get an equally smooth curve! You may also need to increase the polynomial order and play around with the number of data points as well (see also the exercise set from week 35).

With this result we move on to the bias-variance trade-off analysis.

Consider a dataset $\mathcal{L}$ consisting of the data $\mathbf{X}_{\mathcal{L}} = \{(y_j, \boldsymbol{x}_j), j = 0 \dots n-1\}$.

We assume that the true data is generated from a noisy model

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

Here 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 $\boldsymbol{\theta}$ and the design matrix $\boldsymbol{X}$ which embody our model, that is $\tilde{\boldsymbol{y}} = \boldsymbol{X}\boldsymbol{\theta}$.

The parameters $\boldsymbol{\theta}$ are in turn found by optimizing the mean squared error via the so-called cost function

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

Here the expected value $\mathbb{E}$ is the sample value.

Show that you can rewrite this in terms of a term which contains the variance of the model itself (the so-called variance term), a term which measures the deviation from the true data and the mean value of the model (the bias term) and finally the variance of the noise.

That is, show that

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

with (we approximate $f(\boldsymbol{x}) \approx \boldsymbol{y}$)

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

and

$$\text{var}[\tilde{y}] = \mathbb{E}\left[\left(\tilde{\boldsymbol{y}}-\mathbb{E}\left[\tilde{\boldsymbol{y}}\right]\right)^2\right] = \frac{1}{n}\sum_i(\tilde{y}_i-\mathbb{E}\left[\tilde{\boldsymbol{y}}\right])^2.$$

Important note: Since the function $f(x)$ is unknown, in order to be able to evalute the bias, we replace $f(\boldsymbol{x})$ in the expression for the bias with $\boldsymbol{y}$.

The answer to this exercise should be included in the theory part of the report. This exercise is also part of the weekly exercises of week 38. Explain what the terms mean and discuss their interpretations.

Perform then a bias-variance analysis of the Runge function by studying the MSE value as function of the complexity of your model.

Discuss the bias and variance trade-off as function of your model complexity (the degree of the polynomial) and the number of data points, and possibly also your training and test data using the **bootstrap** resampling method. You can follow the code example in the jupyter-book at https://compphysics.github.io/MachineLearning/doc/LectureNotes/_build/html/chapter3.html#the-bias-variance-tradeoff.

Part h): Cross-validation as resampling techniques, adding more complexity

The aim here is to implement another widely popular resampling technique, the so-called cross-validation method.

Implement the k-fold cross-validation algorithm (feel free to use the functionality of **Scikit-Learn** or write your own code) and evaluate again the MSE function resulting from the test folds.

Compare the MSE you get from your cross-validation code with the one you got from your **bootstrap** code from the previous exercise. Comment and interpret your results.

In addition to using the ordinary least squares method, you should include both Ridge and Lasso regression in the final analysis.

Background literature

1. For a discussion and derivation of the variances and mean squared errors using linear regression, see the Lecture notes on ridge regression by Wessel N. van Wieringen
2. The textbook of Trevor Hastie, Robert Tibshirani, Jerome H. Friedman, The Elements of Statistical Learning, Springer, chapters 3 and 7 are the most relevant ones for the analysis of parts g) and h).

Introduction to numerical projects

Here follows a brief recipe and recommendation on how to answer the various questions when preparing your answers.

- Give a short description of the nature of the problem and the eventual numerical methods you have used.
- Describe the algorithm you have used and/or developed. Here you may find it convenient to use pseudocoding. In many cases you can describe the algorithm in the program itself.
- Include the source code of your program. Comment your program properly. You should have the code at your GitHub/GitLab link. You can also place the code in an appendix of your report.
- If possible, try to find analytic solutions, or known limits in order to test your program when developing the code.
- Include your results either in figure form or in a table. Remember to label your results. All tables and figures should have relevant captions and labels on the axes.
- Try to evaluate the reliabilty and numerical stability/precision of your results. If possible, include a qualitative and/or quantitative discussion of the numerical stability, eventual loss of precision etc.
- Try to give an interpretation of you results in your answers to the problems.
- Critique: if possible include your comments and reflections about the exercise, whether you felt you learnt something, ideas for improvements and other thoughts you've made when solving the exercise. We wish to keep this course at the interactive level and your comments can help us improve it.
- Try to establish a practice where you log your work at the computerlab. You may find such a logbook very handy at later stages in your work, especially when you don't properly remember what a previous test version of your program did. Here you could also record the time spent on solving the exercise, various algorithms you may have tested or other topics which you feel worthy of mentioning.

Format for electronic delivery of report and programs

The preferred format for the report is a PDF file. You can also use DOC or postscript formats or as an ipython notebook file. As programming language we prefer that you choose between C/C++, Fortran2008, Julia or Python. The following prescription should be followed when preparing the report:

- Use Canvas to hand in your projects, log in at https://www.uio.no/english/services/it/education/canvas/ with your normal UiO username and password.
- Upload **only** the report file or the link to your GitHub/GitLab or similar typo of repos! For the source code file(s) you have developed please provide us with your link to your GitHub/GitLab or similar domain. The report file should include all of your discussions and a list of the codes you have developed. Do not include library files which are available at the course homepage, unless you have made specific changes to them.

- In your GitHub/GitLab or similar repository, please include a folder which contains selected results. These can be in the form of output from your code for a selected set of runs and input parameters.

Finally, we encourage you to collaborate. Optimal working groups consist of 2-3 students. You can then hand in a common report.

Software and needed installations

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 tensorflow sympy pandas pillow

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

See below for a discussion of **tensorflow** and **scikit-learn**.

For OSX users we recommend also, 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)

etc etc.

If you don't want to install various Python packages with their dependencies separately, we recommend two widely used distrubutions which set up all relevant dependencies for Python, namely 1. Anaconda Anaconda 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**

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

Popular software packages written in Python for ML are

- Scikit-learn,
- Tensorflow,
- PyTorch and
- Keras.

These are all freely available at their respective GitHub sites. They encompass communities of developers in the thousands or more. And the number of code developers and contributors keeps increasing.