Project 1 on Computational Physics and Machine Learning, deadline January 31, 2023

Data Analysis and Machine Learning FYS-STK3155/FYS4155

Department of Physics, University of Oslo, Norway

Dec 11, 2022

Regression analysis and resampling methods

The main aim of this project is to study in more detail various regression methods, including the Ordinary Least Squares (OLS) method, Ridge regression and finally Lasso regression and eventually kernel regression and/or Bayesian Linear Regression as well as Logistic Regression for classification problems. The project includes also a discussion of support vector machines for classification problems.

The numerical methods include matrix inversion, singular value decomposition, convex optimization methods (gradient descent, steepest descent, stochastic gradient descent, ie iterative solvers) and several central (deterministic) ML methods.

The methods are in turn combined with resampling techniques like the bootstrap method and cross validation.

We will first study how to fit polynomials to a specific two-dimensional function called Franke's function. This is a function which has been widely used when testing various interpolation and fitting algorithms. Furthermore, after having established the model and the method, we will employ resamling techniques such as cross-validation and/or bootstrap in order to perform a proper assessment of our models. We will also study in detail the so-called Bias-Variance trade off.

The Franke function, which is a weighted sum of four exponentials reads as follows

$$f(x,y) = \frac{3}{4}\exp\left(-\frac{(9x-2)^2}{4} - \frac{(9y-2)^2}{4}\right) + \frac{3}{4}\exp\left(-\frac{(9x+1)^2}{49} - \frac{(9y+1)}{10}\right)$$
$$+ \frac{1}{2}\exp\left(-\frac{(9x-7)^2}{4} - \frac{(9y-3)^2}{4}\right) - \frac{1}{5}\exp\left(-(9x-4)^2 - (9y-7)^2\right).$$

The function is defined for $x, y \in [0,1]$. Our first step will be to perform an OLS regression analysis of this function, trying out a polynomial fit with an x and y dependence of the form $[x, y, x^2, y^2, xy, \dots]$. We will also include bootstrap first as a resampling technique. After that we will include the cross-validation technique.

We can use a uniform distribution to set up the arrays of values for x and y, or as in the example below just a set of fixed values for x and y with a given step size. We will fit a function (for example a polynomial) of x and y. Thereafter we will repeat much of the same procedure using Ridge and Lasso regression, introducing thus a dependence on the regularization parameter (also called penalty) λ.

The Python code for the Franke function is included here (it performs also a three-dimensional plot of it)

```
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
from matplotlib import cm
from matplotlib.ticker import LinearLocator, FormatStrFormatter
import numpy as np
from random import random, seed

fig = plt.figure()
ax = fig.gca(projection='3d')

# Make data. Use alternatively the uniform distribution
x = np.arange(0, 1, 0.05)
y = np.arange(0, 1, 0.05)
# Using numpy's meshgrid (why?)
x, y = np.meshgrid(x,y)


def FrankeFunction(x,y):
    term1 = 0.75*np.exp(-(0.25*(9*x-2)**2) - 0.25*((9*y-2)**2))
    term2 = 0.75*np.exp(-((9*x+1)**2)/49.0 - 0.1*(9*y+1))
    term3 = 0.5*np.exp(-(9*x-7)**2/4.0 - 0.25*((9*y-3)**2))
    term4 = -0.2*np.exp(-(9*x-4)**2 - (9*y-7)**2)
    return term1 + term2 + term3 + term4


z = FrankeFunction(x, y)

# Plot the surface.
surf = ax.plot_surface(x, y, z, cmap=cm.coolwarm,
                       linewidth=0, antialiased=False)

# Customize the z axis.
```

```
ax.set_zlim(-0.10, 1.40)
ax.zaxis.set_major_locator(LinearLocator(10))
ax.zaxis.set_major_formatter(FormatStrFormatter('%.02f'))

# Add a color bar which maps values to colors.
fig.colorbar(surf, shrink=0.5, aspect=5)

plt.show()
```

Discussion of data sets and report format. Before we proceed however, we would like to add some words about data sets and how to prepare your reports.

The data sets that we propose here are (the default sets)

- Regression (fitting a continuous function).
  1. Either the Franke function proposed by us or data sets your propose.

- Classification. Here you will also need to develop a Logistic regression code. The data set we propose are the so-called Wisconsin Breat Cancer Data data set of images representing various features of tumors. A longer explanation with links to the scientific literature can be found at the Machine Learning repository of the University of California at Irvine. Feel free to consult this site and the pertinent literature.

You can find more information about this at the Scikit-Learn site or at the University of California at Irvine.

However, if you would like to study other data sets, feel free to propose other sets. What we list 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.

Your answers to the projects have to be presented as a standard scientific report. The instructions on how to do this and how we grade are available at `https://github.com/CompPhysics/MachineLearning/blob/master/doc/Projects/EvaluationGrading/EvaluationForm.md`. Please do spend some time to read our guidelines. For PhD students the grades are either passed or not passed, with a threshold of at least the grade B.

Part a): Ordinary Least Square (OLS) on the Franke function. We will generate our own dataset for a function FrankeFunction(x, y) with $x, y \in [0, 1]$. The function $f(x, y)$ is the Franke function. 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 either a matrix inversion or a singular value decomposition from e.g., **numpy**) and perform a standard least square regression analysis using polynomials in x and y up to fifth order. You can use all the functionality of Numpy.

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

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

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

You should consider and discuss ways to scale your data and split the data in training and test data. For this part 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*. Similarly, you can use the data normalization/scaling functionality of **Scikit-Learn**.

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.

Part b): Bias-variance trade-off and resamplng techniques. Our aim here is to study the bias-variance trade-off by implementing the **bootstrap** resampling technique.

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!

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

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

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

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

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

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

The mean squared error can be rewritten as

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

Explain what the terms mean, which one is the bias and which one is the variance and discuss their interpretations.

Perform then a bias-variance analysis of the Franke 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.

Part c) Cross-validation as resampling techniques, adding more complexity. The aim here is to write your own code for another widely popular resampling technique, the so-called cross-validation method. Again, before you start with cross-validation approach, you should consider to scale your data. You can use also folding splitting provided by **Scikit-Learn**.

Implement the k-fold cross-validation algorithm and evaluate again the MSE function resulting from the test folds. You can compare your own code with that from **Scikit-Learn** if needed.

Compare the MSE you get from your cross-validation code with the one you got from your **bootstrap** code. Comment your results. Try $5-10$ folds. You can also compare your own cross-validation code with the one provided by **Scikit-Learn**.

Part d): Ridge Regression on the Franke function with resampling. Write your own code for the Ridge method, either using matrix inversion or the singular value decomposition as done in the previous exercise or howework 2 (see also chapter 3.4 of Hastie *et al.*, equations (3.43) and (3.44)). Perform the same bootstrap analysis as in the part b) (for the same polynomials) and the cross-validation part in part c) but now for different values of λ. Compare and analyze your results with those obtained in parts a-c). Study the dependence on λ.

Study also the bias-variance trade-off as function of various values of the parameter λ. For the bias-variance trade-off, use the **bootstrap** resampling method. Comment your results.

Part e): Lasso Regression on the Franke function with resampling. This part is essentially a repeat of the previous two ones, but now with Lasso regression. Write either your own code (difficult and optional) or, in this case, you can also use the functionalities of **Scikit-Learn** (recommended). Give a critical discussion of the three methods and a judgement of which model fits the data best. Perform here as well an analysis of the bias-variance trade-off using the **bootstrap** resampling technique and an analysis of the mean squared error using cross-validation.

Part f): Write your own Stochastic Gradient Descent code, first step. In order to get started, we will now replace in our standard ordinary least squares (OLS) and Ridge regression codes (from project 1) the matrix inversion algorithm with our own gradient descent (GD) and SGD codes. You can use the Franke function or other data of your choice. However, we recommend using a simpler function like $f(x) = a_0 + a_1x + a_2x^2$ or higher-order one-dimensional polynomials. You can obviously test your final codes against for example the Franke function.

You should include in your analysis of the GD and SGD codes the following elements

1. A plain gradient descent with a fixed learning rate (you will need to tune it).
2. Add momentum to the plain GD code and compare convergence with a fixed learning rate (you may need to tune the learning rate).
3. Repeat these steps for stochastic gradient descent with mini batches and a given number of epochs. Use a tunable learning rate as discussed in the lectures from week 50. Discuss the results as functions of the various parameters (size of batches, number of epochs etc)
4. Implement the Adagrad method in order to tune the learning rate. Do this with and without momentum for plain gradient descent and SGD.
5. Add RMSprop and Adam to your library of methods for tuning the learning rate.

In summary, you should perform an analysis of the results for OLS and Ridge regression as function of the chosen learning rates, the number of mini-batches and epochs as well as algorithm for scaling the learning rate. You can also compare your own results with those that can be obtained using for example **Scikit-Learn**'s various SGD options. Discuss your results. For Ridge regression you need now to study the results as functions of the hyper-parameter λ and the learning rate η. Discuss your results.

You will need your SGD code for the setup of the and Logistic Regression code. You will find the Python Seaborn package useful when plotting the results as function of the learning rate η and the hyper-parameter λ when you use Ridge regression.

We recommend reading chapter 8 on optimization from the textbook of Goodfellow, Bengio and Courville. This chapter contains many useful insights and discussions on the optimization part of machine learning.

Part g): Write your Logistic Regression code, final step. We will now study a classification problem (using the Wisconsin breast cancer data as possible data set) with Logistic regression.

Define your cost function and the design matrix before you start writing your code. Write thereafter a Logistic regression code using your SGD algorithm. You can also use standard gradient descent in this case, with a learning rate as hyper-parameter. Study the results as functions of the chosen learning rates. Add also an l_2 regularization parameter λ. Compare your results with those obtained using **Scikit-Learn**'s logistic regression functionality. You should consider scaling/normalizing your data and include cross-validation.

Part h): Support Vector Machines (optional). For the classification problem, we include now support vector machines. Feel free to use **Scikit-Learn's** functionality here and compare your results with those obtained from logistic regression. This part will be discussed during our first lectures in January.

Part i) Critical evaluation of the various algorithms. After all these glorious calculations, you should now summarize the various algorithms and come with a critical evaluation of their pros and cons. Which algorithm works best for the regression case and which is best for the classification case. These codes can also be part of your final project 3, but now applied to other data sets.

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 here.

(a) Mehta et al, arXiv 1803.08823, *A high-bias, low-variance introduction to Machine Learning for physicists*, ArXiv:1803.08823.

Introduction to numerical projects

Here follows a brief recipe and recommendation on how to write a report for each project.

- 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.
- 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, Rust or Python. For the machine learning topics, Python is recommended. The following prescription should be followed when preparing the report:

- 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.

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.