

Project 1, deadline January 31, 2024

Erasmus+ Data Analysis and Machine Learning course, Fall semester
2023

Nov 11, 2023

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 and Ridge regression.

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 resampling 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.

Note: you can replace the above function with more realistic data from your own research. Feel free to do so. The Franke function is just an example of data sets where we can produce as many data points for training as we want.

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

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

The function will be 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. As in homeworks 1 and 2, 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 regression, introducing thus a dependence on the bias (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.
x = np.arange(0, 1, 0.05)
y = np.arange(0, 1, 0.05)
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()
```

Part a): Ordinary Least Square (OLS) on the Franke function. We will generate our own dataset for a function $\text{FrankeFunction}(x, y)$ 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**) or use your code from the exercise sets and perform a standard least square regression analysis using polynomials in x and

y up to fifth order. Find the [confidence intervals](#) of the parameters (estimators) β by computing their variances, evaluate the Mean Squared error (MSE)

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

and the R^2 score function. If \tilde{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

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

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

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

Your code should consider a scaling of the data (for example by subtracting the mean value) and a split of 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 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 resampling 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 $\mathbf{X}_{\mathcal{L}} = \{(y_j, \mathbf{x}_j), j = 0 \dots n-1\}$. Let us assume that the true data is generated from a noisy model

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

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

The parameters β are in turn found by optimizing the means squared error via the so-called cost function

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

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

Show that you can rewrite this as

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

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. Plot the bias and variance trade-off, and evaluate how it depends on your model complexity, the number of data points, and possibly also the noise parameter.

Note also that when you calculate the bias, in all applications you don't know the function values f_i . You would hence replace them with the actual data points y_i .

Part c) Cross-validation as resampling techniques, adding more complexity. The aim here is to study another widely popular resampling technique, the so-called cross-validation method.

Before you start with the cross-validation approach, you should assess whether you should scale your data before the whole procedure, or during the procedure inbetween each split. This issue is relevant to the topic of [data leakage](#).

Implement the k -fold cross-validation algorithm and evaluate again the MSE function resulting from the test folds. You can use the functionality of **Scikit-Learn** or even write your own code. Try 5 – 10 folds, comment on your results.

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

Part d): Ridge Regression on the Franke function with resampling. We will now use Ridge regression. You can use the **Scikit-Learn** functionality or write your own code.

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): Introducing Real Data. With our codes functioning and having been tested properly on a simpler function we are now ready to look at real data. **Feel free to propose other data sets!** This part can be discussed in more detail when we meet during November 12 and 13.

Part f): Presentation. At the end, you should present a critical evaluation of your results and discuss the applicability of these regression methods to the type of data presented here.

Part g, optional with additional score of 30 points): Write your own Stochastic Gradient Descent code. This exercise is optional. In order to get started, we will now replace in our standard ordinary least squares (OLS) and Ridge regression codes with the matrix inversion algorithm with our own gradient descent (GD) and SGD codes. You can use the Franke function as data set. 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. Having written your own gradient descent code gives you a deeper insight about what is at the heart of most machine learning methods, namely an optimization problem with the calculation of derivatives.

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. You can use a tunable learning rate as discussed in the lectures. 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.

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.

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.
3. For discussions on the mathematics of deep learning and machine learning, we recommend to recent textbooks:
 - (a) [Mathematical Introduction to Deep Learning: Methods, Implementations, and Theory](#), by Arnulf Jentzen, Benno Kuckuck, Philippe von Wurstemberger
 - (b) [Mathematics for deep learning](#) by Marc Peter Deisenroth, A. Aldo Faisal, and Cheng Soon Ong

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 reliability 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 your 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 or Python. The following prescription should be followed when preparing the report:

- Send us by email **only** the report file or the link to your GitHub/GitLab or similar repos! Make sure it is public or if not, give us access. 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.
- 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 distributions 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.