

Project 2 on Machine Learning, deadline November 8

Data Analysis and Machine Learning FYS-STK3155/FYS4155

Department of Physics, University of Oslo, Norway

Oct 17, 2019

Classification and Regression, from linear and logistic regression to neural networks

The main aim of this project is to study both classification and regression problems, where we can reuse the regression algorithms studied in project 1. We will include logistic regression for classification problems and write our own multilayer perceptron code for studying both regression and classification problems. The codes developed in project 1, including bootstrap and/or cross-validation as well as the computation of the mean-squared error and/or the R^2 or the accuracy score (classification problems) functions can also be utilized in the present analysis.

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

- Regression (fitting a continuous function). In this part you will need to bring up your results from project 1 and compare these with what you get from your Neural Network code to be developed here. The data sets could be
 1. Either the Franke function or the terrain data from project 1, or data sets your propose.
- Classification. Here you will also need to develop a Logistic regression code that you will use to compare with the Neural Network code. The data set we propose are
 1. The credit card data set from [UCI](#). This data set links to a [recent scientific article](#). Furthermore, in the lecture slides on [Logistic Regression](#), you will find an example code for the Credit Card data. You could use this code as an example on how to read the data and use **Scikit-Learn** to run a classification problem.

However, if you would like to study other data sets, feel free to propose other sets. What we listed 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. It is also an essential elements of the scientific discussion.

In particular, when developing your own Logistic Regression code for classification problems, the so-called Wisconsin Cancer data (which is a binary problem, benign or malignant tumors) may be studied. You find more information about this at the [Scikit-Learn site](#) or at the [University of California at Irvine](#). The [lecture slides on dimensionality reduction](#) have several code examples on this data set.

Part a): Write your Logistic Regression code, first step. If you opt for the credit card data, your first task is to familiarize yourself with the data set and the scientific article. We recommend also that you study the code example in the [Logistic Regression](#).

Write the part of the code which reads in the data and sets up the relevant data sets.

Part b): Write your Logistic Regression code, second step. Now you should write your Logistic Regression code with the aim to reproduce the Logistic Regression analysis of the [scientific article](#).

Define your cost function and the design matrix before you start writing your code.

In order to find the optimal parameters of your logistic regressor you should include a gradient descent solver, as discussed in the [gradient descent lectures](#). Since we don't have so many data points, you may just code the standard gradient descent with a given learning rate, or even attempt to use the Newton-Raphson method. Alternatively, it may be useful for the next part on neural networks to implement a stochastic gradient descent with and without mini-batches. Stochastic gradient with mini-batches may give the best results. You could finally compare your code with the output from **scikit-learn**'s toolbox for optimization methods applied to logistic regression.

To measure the performance of our classification problem we use the so-called *accuracy* score. The accuracy is as you would expect just the number of correctly guessed targets t_i divided by the total number of targets. A perfect classifier will have an accuracy score of 1.

$$\text{Accuracy} = \frac{\sum_{i=1}^n I(t_i = y_i)}{n},$$

where I is the indicator function, 1 if $t_i = y_i$ and 0 otherwise if we have a binary classification problem. Here t_i represents the target and y_i the outputs of your Logistic Regression code.

You can compare your own results with those obtained using **scikit-learn**.

As stated in the introduction, it can also be useful to study other datasets. In particular, when developing your own Logistic Regression code for classification problems, the so-called Wisconsin Cancer data (which is a binary problem, benign or malignant tumors) may be studied. You find more information about this at the [Scikit-Learn site](https://archive.ics.uci.edu/ml/datasets/breast+cancer+wisconsin+(original)) or at the "University of California at Irvine": [https://archive.ics.uci.edu/ml/datasets/breast+cancer+wisconsin+\(original\)](https://archive.ics.uci.edu/ml/datasets/breast+cancer+wisconsin+(original)). The [lecture slides on dimensionality reduction](#) have several code examples on this data set.

Part c): Writing your own Neural Network code. Your aim now, and this is the central part of this project, is to write your own Feed Forward Neural Network code implementing the back propagation algorithm discussed in the [lecture slides](#). We start with the Logistic Regression case and the data set discussed in parts a) and b) but train now the network to find the optimal weights and biases. You are free to use the codes in the above lecture slides as starting points.

Discuss again your choice of cost function.

Train your network and compare the results with those from your Logistic Regression code. You should test your results against a similar code using **Scikit-Learn** (see the examples in the above lecture notes) or **tensorflow/keras**.

Comment your results and give a critical discussion of the results obtained with the Logistic Regression code and your own Neural Network code. Make an analysis of the regularization parameters and the learning rates employed to find the optimal accuracy score.

A useful reference on the back propagation algorithm is [Nielsen's book](#). It is an excellent read.

Part d): Regression analysis using neural networks. Here we will change the cost function for our neural network code developed in part c) in order to perform a regression (fitting a function or some data set) analysis. As stated above, our default data sets could be either the Franke function or the terrain data from project 1.

Compare your results from the neural network regression analysis (with a discussion of learning rates and regularization parameters) with those you obtained in project 1. Alternatively, if you opt for other data sets, you would need to run your standard ordinary least squares, Ridge and Lasso calculations using your codes from project 1.

Again, we strongly recommend that you compare your own neural Network code and results against a similar code using **Scikit-Learn** (see the examples in the above lecture notes) or **tensorflow/keras**.

Part e) 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. The text of Michael Nielsen is highly recommended, see [Nielsen's book](#). It is an excellent read.
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. [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 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:

- Use Devilry to hand in your projects, log in at <http://devilry.ifi.uio.no> with your normal UiO username and password and choose either 'fysstk3155' or 'fysstk4155'. There you can load up the files within the deadline.
- Upload **only** the report file! For the source code file(s) you have developed please provide us with your link to your github 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 git 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.
- In this and all later projects, you should include tests (for example unit tests) of your code(s).
- Comments from us on your projects, approval or not, corrections to be made etc can be found under your Devilry domain and are only visible to you and the teachers of the course.

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.

1. For project 3, you should feel free to use your own codes from projects 1 and 2, eventually write your own for SVMs and/or Decision trees and random forests' or use the available functionality of **scikit-learn**, **tensorflow**, etc.
2. The estimates you used and tested in projects 1 and 2 should also be included, that is the *R*²-score, **MSE**, cross-validation and/or bootstrap if these are relevant.

3. If possible, you should link the data sets with existing research and analyses thereof. Scientific articles which have used Machine Learning algorithms to analyze the data are highly welcome. Perhaps you can improve previous analyses and even publish a new article?
4. A critical assessment of the methods with ditto perspectives and recommendations is also something you need to include.