Project 2 on Machine Learning, deadline November 12

Data Analysis and Machine Learning FYS-STK3155/FYS4155

Department of Physics, University of Oslo, Norway

Nov 2, 2018

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, starting with the regression algorithms studied in project 1. We will include logistic regresion 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 the R2 score function can also be utilized (and included in logistic regression and the neural network codes) in the present analysis.

We will use the so-called Ising model for our training data and will focus on supervised training. We will follow closely the recent article of Mehta et al, arXiv 1803.08823. This article stands out as an excellent review on machine learning (ML) algorithms. The added benefit is that each figure and model presented in this article is accompanied by its jupyter notebook. This means that we can start using these and compare with our own results. They provide also the data set for the regression and classification analysis that we will explore. In this sense, with their available notebooks, it makes life easier since we can compare our own codes with their codes.

With the abovementioned configurations we will determine, using first various regression methods, the value of the coupling constant for the energy of the one-dimensional Ising model. Thereafter, we will use the two-dimensional data, but now computed at different temperatures, in order to classify the phase of the Ising model. Below the critical temperature, the system will be in a so-called ferromagnetic phase. Close to the critical temperature, the final magnetization becomes smaller and smaller in absolute value while above the critical temperature, the net magnetization is zero. This classification case, that is the two-dimensional Ising model, will be studied using logistic regression and deep neural networks. The aim is to develop your own logistic regression code

for the classification of the phases (this is a binary model) and your multilayer perceptron code for the classification and regression case. You can compare your own results with those obtained using **scikit-learn** or **tensorflow** or other Python packages such as **keras** or other.

Feel free to use the notebooks to benchmark your code. If you wish to write your own C++ or Fortran program for say a multilayer neural network model and a logistic regression model, please feel free to do so.

Part a): Producing the data for the one-dimensional Ising model. The model we will employ in our studies is the so-called Ising model. Together with models like the Potts model and similar so-called lattice models, the Ising model has been widely studied in mathematics (in statistics in particular), physics, life science, chemistry and even in the social sciences in order to model social behavior. It is a simple binary value system where the variables of the model (spins often in physics) can take two values only, for example ± 1 or 0 and 1. The system exhibits a phase transition in two or higher dimensions and the first person to find the analytical expressions for various expectation values was the Norwegian chemist Lars Onsager (Nobel prize in chemistry) after a tour de force mathematics exercise.

In our discussions here we will stay with a physicist's approach and call the variables for spin. You could replace this with any other type of binary variables, ranging from a two political parties to blue and red spheres. In its simplest form we define the energy of the system as

$$E = -J \sum_{<kl>}^{N} s_k s_l,$$

with $s_k = \pm 1$, N is the total number of spins, J is a coupling constant expressing the strength of the interaction between neighboring spins.

The symbol $< kl >$ indicates that we sum over nearest neighbors only. Notice that for $J > 0$ it is energetically favorable for neighboring spins to be aligned. This feature leads to, at low enough temperatures, a cooperative phenomenon called spontaneous magnetization. That is, through interactions between nearest neighbors, a given magnetic moment can influence the alignment of spins that are separated from the given spin by a macroscopic distance. These long range correlations between spins are associated with a long-range order in which the lattice has a net magnetization in the absence of a magnetic field.

We start by considering the one-dimensional Ising model with nearest neighbor interactions. This model does not exhibit any phase transition.

Consider the 1D Ising model with nearest-neighbor interactions

$$E[\hat{s}] = -J \sum_{j=1}^{N} s_j s_{j+1},$$

on a chain of length N with so-called periodic boundary conditions and $S_j = \pm 1$ Ising spin variables. In one dimension, this model has no phase transition at finite temperature.

In the Python code below we generate, with a coupling coefficient set to $J = 1$, a large number of spin configurations say 10000 as shown in the code below. It means that our data will be a set of $i = 1 \dots n$ points of the form $\{(E[\boldsymbol{s}^i], \boldsymbol{s}^i)\}$. Our task is to find the value of J from the data set using linear regression.

Here is the Python code you need to generate the training data, see also the notebook of Mehta et al.

```
import numpy as np
import scipy.sparse as sp
np.random.seed(12)

import warnings
#Comment this to turn on warnings
warnings.filterwarnings('ignore')

### define Ising model aprams
# system size
L=40

# create 10000 random Ising states
states=np.random.choice([-1, 1], size=(10000,L))

def ising_energies(states,L):
    """
    This function calculates the energies of the states in the nn Ising Hamiltonian
    """
    J=np.zeros((L,L),)
    for i in range(L):
        J[i,(i+1)%L]-=1.0
    # compute energies
    E = np.einsum('...i,ij,...j->...',states,J,states)

    return E
# calculate Ising energies
energies=ising_energies(states,L)
```

We can now recast the problem as a linear regression model using our codes from project 1. The way we are going to build our model mimicks the way we could think of finding say the gravitional constant for the graviational force between two planets. In the absence of any prior knowledge, one sensible choice is the all-to-all Ising model

$$E_{\text{model}}[\boldsymbol{s}^i] = -\sum_{j=1}^{N}\sum_{k=1}^{N} J_{j,k} s_j^i s_k^i.$$

Here i represents a particular spin configuration (one of the possible n configurations we generated with the code above).

This model is uniquely defined by the non-local coupling strengths J_{jk} which we want to learn. The model is linear in $\mathbf{J}$ which makes it possible to use linear regression.

To apply linear regression, we recast this model in the form

$$E_{\text{model}}^i \equiv \mathbf{X}^i \cdot \mathbf{J},$$

where the vectors $\mathbf{X}^i$ represent all two-body interactions $\{s_j^i s_k^i\}_{j,k=1}^N$, and the index i runs over the samples in the data set. To make the analogy complete, we can also represent the dot product by a single index $p = \{j,k\}$, i.e. $\mathbf{X}^i \cdot \mathbf{J} = X_p^i J_p$. Note that the regression model does not include the minus sign, so we expect to learn negative J's.

With these preliminaries, we are now ready to reutilize our codes from project 1.

Part b): Estimating the coupling constant of the one-dimensional Ising model using linear regression. We start with the one-dimensional Ising model and use the data we have generated with $J = 1$ in the previous point. Use linear regression, Lasso and Ridge regression as done in project 1. You can compare your results with those of Mehta et al.. Make sure it is the 1D data which is used.

Discuss the methods and how they perform in computing the coupling constant J and include a bias-variance analysis using either cross-validation or bootstrap. Discuss also the mean squared error and the $R2$ score as measures to assess your model.

Give a critical analysis of your results.

Part c): Determine the phase of the two-dimensional Ising model. We switch now to binary classification methods and use logistic regression to define the phases of the Ising model. This means that we switch to the two-dimensional Ising model and use the data sets generated by Mehta et al These energies and their corresponding spin orientation configurations represent then your data. We will use a fixed lattice of $L \times L = 40 \times 40$ spins in two dimensions. The link above contains data for several temperatures. The theoretical critical temperature for a phase transition is $T_C \approx 2.269$ in units of energy. However, for a finite lattice the results representing the critical temperature are slightly higher ($T_C \approx 2.3$).

Our goal here, using logistic regression, is to train our model to predict the phase of a sample given the spin configuration, whether it represents a state above the critical temperature or below. The configurations representing states

below the critical temperature are called ordered states (the spins tend to point in one direction, resulting in a net magnetic moment) while those above the critical temperature are called disordered. Since a finite lattice like this does not exhibit a clear sign of a phase transition we will mainly stay with either orderer or disordered phases. You could include the critical phase if you want.

Your aim here is thus to read in these data (use the examples from Mehta et al) and write your own code for doing logistic regression, see the lecture notes on logistic regression.

In this case, to evaluate the model, we will use the so-called accuracy score instead of the bootstrap or cross-validation as done in the standard linear regression part discussed in b). Examples of how to define the accuracy score can be found under the neural network slides, see for example the slides here.

To measure the performance of our network we evaluate how well it does it data it has never seen before, i.e. the test data. We measure the performance of the network using the *accuracy* score. The accuracy is as you would expect just the number of images correctly labeled divided by the total number of images. 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, where t_i represents the target and y_i the outputs.

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.

The notebook of Mehta et al is highly recommended in order to benchmark your code and results.

Part d): Regression analysis of the one-dimensional Ising model using neural networks. Your aim now, and this is the central part of this project, is to write to your own multilayer perceptron model implementing the back propagation algorithm discussed in the lecture slides. We start with the regression case 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.

Train your network and compare the results with those from your linear regression code. You can test your results against a similar code using $_s cikit_l earn_($ *seetheexamplesintheabovelecturenotes*$)or$**tensorflow/keras**.

A useful reference on the back progagation algorithm is Nielsen's book. It is an excellent read.

Part e): Classifying the Ising model phase using neural networks. Finally, change now your cost function to the *log* cross-entropy classification cost function for the case discussed in part c). Train your network again and compare the results with those from your logistic regression code i c). Here again you can compare your results with those of Mehta et al. There they used **tensorflow** to classify the phases.

Part f) 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 will 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.

If you wish to read more about the Ising model and statistical physics here are three suggestions.

1. M. Plischke and B. Bergersen, *Equilibrium Statistical Physics*, World Scientific, see chapters 5 and 6.

2. D. P. Landau and K. Binder, *A Guide to Monte Carlo Simulations in Statistical Physics*, Cambridge, see chapters 2,3 and 4.

3. M. E. J. Newman and T. Barkema, *Monte Carlo Methods in Statistical Physics*, Oxford, see chapters 3 and 4.

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 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 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.