P02 – Machine Learning from Scratch

1. Univariate linear regression with gradient descent

1.1 Become familiar with the environment

Consider the Python script **`linear_regression.py`**. It contains skeleton implementations of

- the linear regression method as well as…
- a batch gradient descent optimizer for its training together with…
- facilities to handle data and visualize training progress.

Thus, it implements a complete machine learning solution from scratch without the help of optimized learning libraries (they are just used for data handling).

Make yourself familiar with the script in its current form:

- What is the control flow?
- What do you have to complete in order for the script to work (look out for the "**`TODO`**" comments)?
- Which functions do you need to understand in detail in order to implement your parts? Which functions do you not need to consider?

The script makes heavy use of functions as parameters of other functions (and as their return values as well). Compare this to the math on the lecture slides. Does it make the program flow more comprehensible?

Hint: If you need to refresh your Python programming skills, you might want to have a look at http://opentechschool.github.io/python-data-intro/

1.2 Implement univariate linear regression

Implement the missing parts for the linear regression method:

- The regression function **`h_lr()`**
- The cost function **`J()`**

Use the mathematical descriptions from the lecture slides and the given API (application programming interface to the function). Check the given formulas for the partial derivatives – are they ok?

Hint: The sample solution needs only 7 additional lines of code.

1.3 Implement gradient descent

Implement the missing parts for the **`batch_gradient_descent()`** method. Some lines of code are already given to start you off:

- Initializations and unpacking of parameters
- The main loop that already controls convergence (by the use of the hyper parameter **`CONVERGENCE_DELTA`**)

Please fill in the main code of the gradient descent algorithm as discussed in the lecture. Take care to do simultaneous updates correctly.

Hint: The sample solution needs only 7 additional lines of code.

1.4 Experiment with the two hyper parameters

Experiment with the hyper parameters **`LEARNING_RATE`** and **`CONVERGENCE_DELTA`** in the script: Are they already well set?

Next, change to another problem and data set, e.g., the height-weight data in **`htwt.xlsx`** (here the task is to predict people's weight from their body height). Therefore, adapt the 4 lines on loading the training- and creating the test data in the **`main()`** function. Do your optimal hyper parameters still work well?

What is the role of the z-transform pre-processing (step 2 in **`main`**) in this? Experiment with removing the pre-processing and then trying different learning rates with both data sets (mind the re-transformation in **`main`**'s step 4). How is the portability affected? Why?

1.5 Reflect on the implementation

Take a moment to think about your script. Let your thoughts be guided by following questions:

- How difficult/easy has it been for you to implement a complete machine learning approach "from math"?
- Which factor played the already given skeleton and the thus pre-sketched program flow?
- What share of complexity is introduced by the actual machine learning mechanisms, as compared to the bare programming?
- What is your estimate of time you would need to program the whole solution really from scratch (i.e., without a given skeleton)?
- What value then has the knowledge of the API of a well-known ML library (e.g., scikit-learn) for you?

2. [Optional] Learning to play tic-tac-toe

Revisit chapter one of [Mitchell, 1997] and its description of a reinforcement learning solution to checkers. Then, solve exercise 1.5 therein with a Python script:

"Implement an algorithm similar to that discussed for the checkers problem, but use the simpler game of tic-tac-toe. Represent the learned function $\hat{V}$ as a linear combination of board features of your choice. To train your program, play it repeatedly against a second copy of the program that uses a fixed evaluation function you created by hand [i.e., no learning in the second program]. Plot the percent of games won by your system, versus the number of training games played."