Advanced Aspects of Nature Inspired Search and Optimisation 2019/2020

Lab 2 MSc: Time Series Prediction with GP

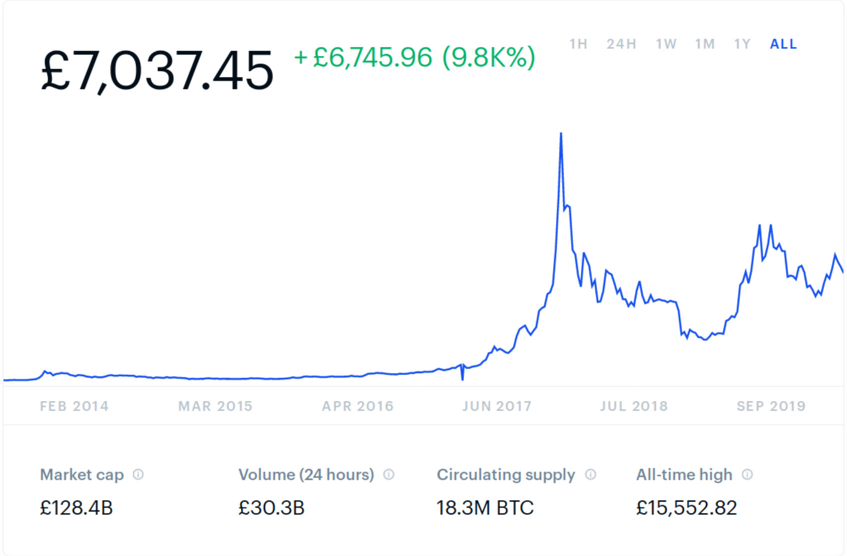
Released: March 6, 2020
Deadline: March 27, 2020
Weight: 25 %

NB! This coursework is only compulsory for MSc students taking the 20cr module. We released a different Lab 2 with an earlier deadline for UG students taking the 20cr module.

You need to implement one program that solves Exercises 1-3 using any programming language. In Exercise 5, you will run a set of experiments and describe the result using plots and a short discussion.

(In the following, replace `abc123` with your username.) You need to submit one zip file with the name `niso3-abc123.zip`. The zip file should contain one directory named `niso3-abc123` containing the following files:

- the source code for your program
- a Dockerfile (see the appendix for instructions)
- a PDF file for Exercises 4 and 5



£7,037.45 +£6,745.96 (9.8K%)

Market cap	Volume (24 hours)	Circulating supply	All-time high
£128.4B	£30.3B	18.3M BTC	£15,552.82

In this lab, we will do a simple form of time series prediction. We assume that we are given some historical data, (e.g. bitcoin prices for each day over a year), and need to predict the next value in the time series (e.g., tomorrow's bitcoin value).

We formulate the problem as a regression problem. The training data consists of a set of m input vectors $\mathcal{X} = (x^{(0)}, \ldots, x^{(m-1)})$ representing historical data, and a set of m output values $\mathcal{Y} = (x^{(0)}, \ldots, x^{(m-1)})$, where for each $0 \leq j \leq m-1$, $x^{(j)} \in \mathbb{R}^n$ and $y^{(j)} \in \mathbb{R}$. We will use genetic programming to evolve a prediction model $f : \mathbb{R}^n \rightarrow \mathbb{R}$, such that $f(x^{(j)}) \approx y^{(j)}$.

Candidate solutions, i.e. programs, will be represented as *expressions*, where each expression *evaluates* to a value, which is considered the output of the program. When evaluating an expression, we assume that we are given a current input vector $x = (x_0, \ldots, x_{n-1}) \in \mathbb{R}^n$. Expressions and evaluations are defined recursively. Any floating number is an expression which evaluates to the value of the number. If e_1, e_2, e_3, and e_4 are expressions which evaluate to v_1, v_2, v_3 and v_4 respectively, then the following are also expressions

- `(add` e_1 e_2`)` is addition which evaluates to $v_1 + v_2$, e.g. `(add 1 2)`$\equiv 3$
- `(sub` e_1 e_2`)` is subtraction which evaluates to $v_1 - v_2$, e.g. `(sub 2 1)`$\equiv 1$
- `(mul` e_1 e_2`)` is multiplication which evaluates to $v_1 v_2$, e.g. `(mul 2 1)`$\equiv 2$
- `(div` e_1 e_2`)` is division which evaluates to v_1/v_2 if $v_2 \neq 0$ and 0 otherwise, e.g., `(div 4 2)`$\equiv 2$, and `(div 4 0)`$\equiv 0$,
- `(pow` e_1 e_2`)` is power which evaluates to $v_1^{v_2}$, e.g., `(pow 2 3)`$\equiv 8$
- `(sqrt` e_1`)` is the square root which evaluates to $\sqrt{v_1}$, e.g.`(sqrt 4)`$\equiv 2$
- `(log` e_1`)` is the logarithm base 2 which evaluates to $\log(v_1)$, e.g. `(log 8)`$\equiv 3$
- `(exp` e_1`)` is the exponential function which evaluates to e^{v_1}, e.g. `(exp 2)`$\equiv e^2 \approx 7.39$
- `(max` e_1 e_2`)` is the maximum which evaluates to $\max(v_1, v_2)$, e.g., `(max 1 2)`$\equiv 2$
- `(ifleq` e_1 e_2 e_3 e_4`)` is a branching statement which evaluates to v_3 if $v_1 \leq v_2$, otherwise the expression evaluates to v_4 e.g. `(ifleq 1 2 3 4)`$\equiv 3$ and `(ifleq 2 1 3 4)`$\equiv 4$
- `(data` e_1`)` is the j-th element x_j of the input, where $j \equiv |\lfloor v_1 \rfloor| \mod n$.
- `(diff` e_1 e_2`)` is the difference $x_k - x_\ell$ where $k \equiv |\lfloor v_1 \rfloor| \mod n$ and $\ell \equiv |\lfloor v_2 \rfloor| \mod n$
- `(avg` e_1 e_2`)` is the average $\frac{1}{|k-\ell|} \sum_{t=\min(k,\ell)}^{\max(k,\ell)-1} x_t$ where $k \equiv |\lfloor v_1 \rfloor| \mod n$ and $\ell \equiv |\lfloor v_2 \rfloor| \mod n$

In all cases where the mathematical value of an expression is undefined or not a real number (e.g., $\sqrt{-1}$, $1/0$ or `(avg 1 1)`), the expression should evaluate to 0.

We can build large expressions from the recursive definitions. For example, the expression

```
(add (mul 2 3) (log 4))
```

evaluates to

$$2 \cdot 3 + \log(4) = 6 + 2 = 8.$$

To evaluate the fitness of an expression e on a training data $(\mathcal{X}, \mathcal{Y})$ of size m, we use the mean square error

$$f(e) = \frac{1}{m} \sum_{j=0}^{m-1} \left(y^{(j)} - e(x^{(j)}) \right)^2 ,$$

where $e(x^{(j)})$ is the value of the expression e when evaluated on the input vector $x^{(j)}$.

Exercise 1. *(30 % of the marks)*

Implement a routine to parse and evaluate expressions. You can assume that the input describes a syntactically correct expression. Hint: Make use of a library for parsing s-expressions[1]*, and ensure that you evaluate expressions exactly as specified on page 2.*

Input arguments:

- `-expr` *an expression*
- `-n` *the dimension of the input vector n*
- `-x` *the input vector*

Output:

- *the value of the expression*

Example:

```
[pkl@phi ocamlec]$ niso_lab3 -question 1 -n 1 -x "1.0" \
       -expr "(mul (add 1 2) (log 8))"
9.0
[pkl@phi ocamlec]$ niso_lab3 -question 1 -n 2 -x "1.0 2.0" \
       -expr "(max (data 0) (data 1))"
2.0
```

Exercise 2. *(10 % of the marks) Implement a routine which computes the fitness of an expression given a training data set.*

Input arguments:

- `-expr` *an expression*
- `-n` *the dimension of the input vector*
- `-m` *the size of the training data $(\mathcal{X}, \mathcal{Y})$*
- `-data` *the name of a file containing the training data in the form of m lines, where each line contains $n + 1$ values separated by tab characters. The first n elements in a line represents an input vector x, and the last element in a line represents the output value y.*

Output:

- *The fitness of the expression, given the data.*

[1] See e.g. implementations here `http://rosettacode.org/wiki/S-Expressions`

Exercise 3. *(30 % of the marks)*

Design a genetic programming algorithm to do time series forecasting. You can use any genetic operators and selection mechanism you find suitable.

Input arguments:

- `-lambda` *population size*
- `-n` *the dimension of the input vector*
- `-m` *the size of the training data* $(\mathcal{X}, \mathcal{Y})$
- `-data` *the name of a file containing training data in the form of* m *lines, where each line contains* $n+1$ *values separated by tab characters. The first* n *elements in a line represents an input vector* x*, and the last element in a line represents the output value* y*.*
- `-time_budget` *the number of seconds to run the algorithm*

Output:

- *The fittest expression found within the time budget.*

Exercise 4. *(10 % of the marks)*

Describe your algorithm from Exercise 3 in the form of pseudo-code. The pseudo-code should be sufficiently detailed to allow an exact re-implementation.

Exercise 5. *(20 % of the marks)*

In this final task, you should try to determine parameter settings for your algorithm which lead to as fit expressions as possible.

Your algorithm is likely to have several parameters, such as the population size, mutation rates, selection mechanism, and other mechanisms components, such as diversity mechanisms.

Choose parameters which you think are essential for the behaviour of your algorithm. Run a set of experiments to determine the impact of these parameters on the solution quality. For each parameter setting, run 100 repetitions, and plot box plots of the fittest solution found within the time budget.

A. Docker Howto

Follow these steps exactly to build, test, save, and submit your Docker image. Please replace *abc123* in the text below with your username.

1. Install Docker CE on your machine from the following website:
   `https://www.docker.com/community-edition`

2. Copy the PDF file from Exercises 4 and 5 all required source files, and/or bytecode to an empty directory named niso2-abc123 (where you replace abc123 with your username).

```
mkdir niso2-abc123
cd niso2-abc123/
cp ../exercise.pdf .
cp ../abc123.py .
```

3. Create a text file `Dockerfile` file in the same directory, following the instructions below.

```
# Do not change the following line. It specifies the base image which
# will be downloaded when you build your image.

FROM pklehre/niso2020-lab2-msc

# Add all the files you need for your submission into the Docker image,
# e.g. source code, Java bytecode, etc. In this example, we assume your
# program is the Python code in the file abc123.py. For simplicity, we
# copy the file to the /bin directory in the Docker image. You can add
# multiple files if needed.

ADD abc123.py /bin

# Install all the software required to run your code. The Docker image
# is derived from the Debian Linux distribution. You therefore need to
# use the apt-get package manager to install software. You can install
# e.g. java, python, ghc or whatever you need. You can also
# compile your code if needed.

# Note that Java and Python are already installed in the base image.

# RUN apt-get update
# RUN apt-get -y install python-numpy

# The final line specifies your username and how to start your program.
# Replace abc123 with your real username and python /bin/abc123.py
# with what is required to start your program.

CMD ["-username", "abc123", "-submission", "python /bin/abc123.py"]
```

4. Build the Docker image as shown below. The base image `pklehre/niso2019-lab3` will be downloaded from Docker Hub

   ```
   docker build . -t niso2-abc123
   ```

5. Run the docker image to test that your program starts. A battery of test cases will be executed to check your solution.

   ```
   docker run niso2-abc123
   ```

6. Once you are happy with your solution, compress the directory containing the Dockerfile as a zip-file. The directory should contain the source code, the Dockerfile, and the PDF file for Exercise 4 and 5. The name of the zip-file should be `niso2-abc123.zip` (again, replace the abc123 with your username).

   Following the example above, the directory structure contained in the zip file should be as follows:

   ```
   niso2-abc123/exercise.pdf
   niso2-abc123/abc123.py
   niso2-abc123/Dockerfile
   ```

   Submissions which do not adhere to this directory structure will be rejected!

7. Submit the zip file `niso2-abc123.zip` on Canvas.