Exercise 0.2 — Python Tutorial
Artificial Intelligence For Robotics

Fadri Furrer and Mark Pfeiffer

Spring Semester 2017

During this exercise you should be able to become familiar with Python functions, classes, numpy structures and coding in general. The provided examples should introduce you to basic Python application development and give you an introduction to linear regression. An incomplete framework and the required data for this exercise can be found in the lecture's git repository: `https://github.com/ethz-asl/ai_for_robotics`. There we will also add the working solutions after today's exercise session.

1 Linear Regression

The goal of linear regression is to find a regression function for a given data set. The basic example is a linear (1st order polynomial) 2D curve fitting problem as shown in Figure 1.

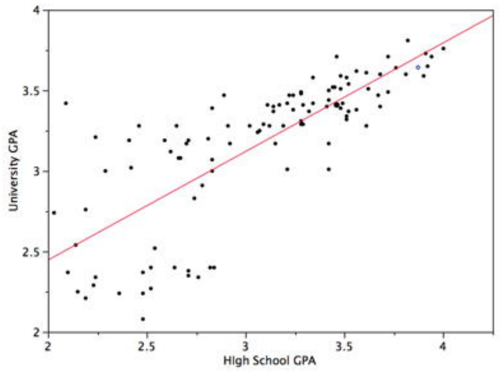


Figure 1: Example for a 2D 1st order linear regression fit (taken from `http://onlinestatbook.com/2/regression/intro.html`).

The linear regression model for a single sample is given by

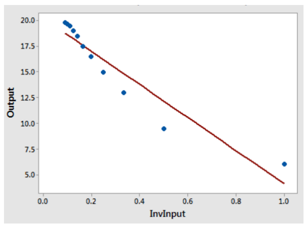
$$y_i = \mathcal{F}(\mathbf{x}_i) * \beta, \tag{1}$$

where $\mathcal{F}(\mathbf{x}_i)$ is the so called feature matrix. However, using non-linear features, the model also allows for a non-linear regression, although the optimization is linear in the feature weights β.

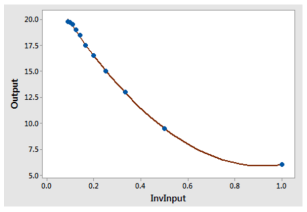
With multiple data samples, the model looks like the following:

$$\underbrace{\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}}_{\mathbf{y}} = \underbrace{\begin{pmatrix} f_1(\mathbf{x}_1) & f_2(\mathbf{x}_1) & \cdots & f_k(\mathbf{x}_1) \\ f_1(\mathbf{x}_2) & f_2(\mathbf{x}_2) & \cdots & f_k(\mathbf{x}_2) \\ \vdots & \vdots & \vdots & \vdots \\ f_1(\mathbf{x}_n) & f_2(\mathbf{x}_n) & \cdots & f_k(\mathbf{x}_n) \end{pmatrix}}_{\mathcal{F}(X)} \cdot \begin{pmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_k \end{pmatrix} \tag{2}$$

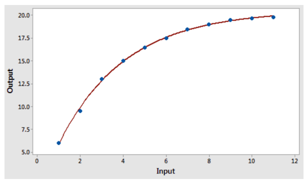
where n is the number of data samples, k is the number of features and $f_i(\mathbf{x_j})$ is the i-th feature evaluated on the j-th data sample. Two examples for a non-linear relationship between input and output data are given in Figure 2. The examples show that the feature selection is crucial for a good regression fit.



(a)



(b)



(c)

Figure 2: The examples show (a) a dataset with a quadratic relationship fitted with a 1st order polynomial, (b) the same dataset fitted with a 2nd order polynomial and (c) an exponential dataset fitted with an exponential function (taken from `http://blog.minitab.com/blog/adventures-in-statistics-2/curve-fitting-with-linear-and-nonlinear-regression`).

In a typical regression task, where input $X = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n]$ and output $\mathbf{y} = [y_1, y_2, \ldots, y_n]$ data is given the parameters β can be computed by a standard least squares fit

$$\beta = \left(\mathcal{F}^T(X) \cdot \mathcal{F}(X)\right)^{-1} \cdot \mathcal{F}^T(X) \cdot \mathbf{y} \tag{3}$$

If you have any questions, just let us know.

1.1 Classes

Brief overview of the partially implemented classes in this exercise and what they are used for.

1.1.1 DataGenerator

This class will only be provided after the exercise, since by knowing how the data is generated, it would be too easy to find a good solution.

- generate raw regression data with specified features
- in this example, the data is only 2D in order to allow for 3D plotting
- uses the noise model for data generation
- save and load the data using an instance of the data saver

1.1.2 DataSaver

- save and load input / output data using pickle[1] serialization
- binary serialization

1.1.3 Features

- (nonlinear) features defined on the input data
- although the input data is only 2D, more than two features can be used for linear regression
- each feature inherits from the feature base class and is defined on the 2D input: $f_i(x_1, x_2)$

[1]https://docs.python.org/2/library/pickle.html

1.1.4 LinearRegressionModel

Model wrapper class for the linear regression.

The functionality of the class is mainly given by the methods

- `fit`(X, y): find the model parameters β given the *training data* X and the function values y using least-squares optimization
- `predict`(X): given the model parameters β, the features and the input data X, predict the regression values $\mathbf{y}$
- `validate`(X,y): compute the validation score of the model on the validation dataset

1.1.5 NoiseModel

- noise model that can be used in the data generation process
- the `sample`() function provides a random noise sample

1.2 Tasks

In general, fill in the sections marked with #`TODO` in order to get the best possible linear regression. Feel free to extend this example or develop new features.

You have to implement the following functionalities to make this example work:

1. `DataSaver.py`: data loader to get the dataset from the binary file
2. `Features.py`: `linearX1` feature and extend the list of features with the ones that you consider to be useful when looking at the data
3. `LinearRegressionModel.py`:
   - loop to compute the feature matrix $\mathcal{F}(X)$
   - `fit` method to find the least-squares fit for the feature weights

   - `predict` method to find the prediction values given the input data (assert that the model parameters were found when executing this method)

4. `NoiseModel.py`: the sample function of the noise model
5. implement the linear regression using sklearn to compare it against the *raw* version

2 SciPy Optimization

This part of the exercise should just give you an impression, how the SciPy optimization toolbox[2] works. This toolbox allows you use different optimizers for constrained and unconstrained optimization problems.

2.1 Tasks

To implement:

1. the call for the optimizer
2. the function to get the Jacobian matrix of the function `f(x)`

3 OpenAI Gym

In this exercise, we will also show you how to use the OpenAI Gym[3] toolbox. You don't have to implement anything here, just sit back and relax.

[2]https://docs.scipy.org/doc/scipy-0.18.1/reference/optimize.html
[3]https://gym.openai.com/