1 Simple Linear Regression

A way to analyze an available data set is to perform linear regression. The most common form of linear regression is known as **least squares fitting**, whose aim is to fit a polynomial curve to the data such that the sum of the squares of the distance from the data points to the line is minimized.

1.1 Short Theory

Simple linear regression is the least squares estimator of a linear regression model with a single explanatory variable. In other words, simple linear regression fits a straight line through the set of n points in such a way that makes the sum of squared residuals of the model (that is, vertical distances between the points of the data set and the fitted line) as small as possible. The adjective simple refers to the fact that this regression is one of the simplest in statistics.

Suppose there are n training samples $\underline{x}_i = (1, x_{i1})^T$ and y_i, where $i = 1, 2, \ldots, n$. These samples are assumed to be instantiations of the input random vector $\underline{X} = (1, X_1)^T$ and the output random variable Y, respectively. The function that describes $\underline{x}_i$ and y_i is:

$$y_i = \underline{\theta}^T \underline{x}_i + \epsilon_i = \theta_0 + \theta_1 x_1 + \epsilon_i,$$

where $\underline{\theta} = (\theta_0, \theta_1)^T$ is the parameters' vector and ϵ_i is the error for a tuple $(\underline{x}_i, y_i)$. The goal is to find the equation of the straight line:

$$Y = \theta_0 + \theta_1 X_1$$

that would provide a "best" fit for the data points. Here the "best" will be understood as in the least-squares approach: a line that minimizes the sum of squared residuals of the linear regression model. In other words, θ_0 (the y-intercept) and θ_1 (the slope) solve the following minimization problem:

$$[\hat{\theta}_0, \hat{\theta}_1] = \arg\min_{\theta_0, \theta_1} \sum_{i=1}^{n} (y_i - \theta_0 - \theta_1 x_{i1})^2 \tag{1}$$

By using either calculus (the geometry of inner product spaces) or simply expanding to get a quadratic expression in θ_0 and θ_1, it can be shown

that the values of θ_0 and θ_1 that minimize the expression above are

$$\hat{\theta}_1 = \frac{\sum_{i=1}^{n}(x_{1i} - \bar{x_1})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_{1i} - \bar{x_1})^2} = \frac{\mathrm{Cov}[\mathbf{x}_1, \mathbf{y}]}{\mathrm{Var}[\mathbf{x}_1]}, \tag{2}$$

$$\hat{\theta}_0 = \bar{\mathbf{y}} - \hat{\theta}_1 \bar{\mathbf{x}}_1, \tag{3}$$

where $\mathbf{x}_1 = (x_{11}, \ldots, x_{i1}, \ldots, x_{n1})$ and $\mathbf{y} = (y_{11}, \ldots, y_{i1}, \ldots, y_{n1})$ are the row vectors (of dimension n) that contain all sample values of the variables X_1 and Y, respectively. The horizontal bar over a quantity indicates the sample-average of that quantity. A more thorough description of the simple linear regression theory can be found in any of the following sources:

- The lecture notes and slides.
- Section 3.1, Chapter 3 from the book "Pattern recognition and machine learning" of Bishop and Nasrabadi.
- `https://en.wikipedia.org/wiki/Simple_linear_regression`
- `http://mathworld.wolfram.com/LeastSquaresFitting.html`

1.2 Python Exercise

Often mathematical models of experimental or physiological systems are developed to form the basis of a measurement technique. For example, it is not possible to measure cardiac output directly, but we may be able to infer it from analysis of a mathematical model of respiratory gas exchange. For such models we are often solving an inverse problem of parameter estimation (the cardiac output), where the parameter of interest is embedded (somewhere) within the mathematical model. In these circumstances, it is often advisable to introduce simulated experimental error into the system to test the robustness of the recovery procedure. The following is one of the simplest possible examples, where the parameters form part of a linear model.

The following activity will lead you through generating some experimental data adding artificial noise and then performing least squares estimation to try to elucidate the underlying model parameters.

1. Generate and plot the data (values in $\mathbf{x}_1$ and $\mathbf{y}$) for a simple straight line of the form $y = \theta_0 + \theta_1 x_1$ where $\theta_0 = 1$ and $\theta_1 = 2$ are constants and $x_1 \in [0, 1]$. You should calculate the value of y at 21 evenly spaced points between 0 and 1.

2. Generate random errors from a normal distribution, with zero mean $\mu = 0$ and standard deviation $\sigma = 0.1$, using Numpy's built-in commands, and add these to each of the y coordinates in step 1. Plot these values as points on the same graph as in step 1. What happens if μ is non-zero?

3. In this step, you will calculate the parameters of the simple linear regression model using the formulas (2), (3). Then, using this model to fit the data generated in step 2, and plot the regression line on the same plot.

4. Repeat steps 2 and 3 for differing values of the standard deviation, from $\sigma = 0$ to $\sigma = 1$, in steps of 0.2, and plot the regression lines on the same graph. *You should use a loop to do this*.

5. For a suitable value of the standard deviation ($\sigma = 0.1$), repeat steps 2 and 3 1000 times to investigate the statistical properties of the regression coefficients. To do this you should calculate the mean and standard deviation of the regression coefficients and compare them to the theoretical values of $\mu = 2$, $\sigma \approx 0.042$ for θ_0, and $\mu = 1$, $\sigma \approx 0.072$ for θ_1. The estimators should also be normally distributed. You do not need to know how these were calculated, but note that the estimators are unbiased, as the mean values of the estimators of the parameters are the parameters themselves.

6. Explain why the above can be used to simulate the effects of random experimental error.

2 Multivariate Linear Regression

In this exercise, you will investigate multivariate linear regression using the normal equation, as introduced in Lecture 2.

2.1 Short Theory

Given a set of samples $x_i \in \mathbb{R}^m$, and their corresponding scores $y_i \in \mathbb{R}^1$, with $1 \leq i \leq n$, we want to predict the score y_i from a sample x_i. Each values in x_i represent a feature of the sample, e.g. intelligence score or extroversion score.

In linear regression, we form a hypothesis that y is a linear function of the features. Particularly,

$$y_i = \beta_0 + \beta^1 * x_i^1 + \beta^2 * x_i^2 + ... + \beta^m * x_i^m, \tag{4}$$

where β^i with $0 \leq \beta \leq m$ are the parameters.

From all the samples and scores, we form two matrices, $X \in \mathbb{R}^{n \times (m+1)}$ by stacking all samples into a big matrix. We also add 1 as the last feature of each sample, to account for the intercept term. Analogously we form a matrix $Y \in \mathbb{R}^{n \times 1}$ by stacking all scores together.

The parameter β can be found using the **normal equation**:

$$\beta = \left(X^T X\right)^{-1} X^T Y \tag{5}$$

After estimating β, the score predictions can be calculate by:

$$\mathtt{pred} = X\beta \tag{6}$$

2.2 Python Exercise

This exercise is contained in the file *multivairate_linear_reg.ipynb*. You need to proceed through this file and add some missing code sections.

Step 1: Load data

The *load_dat* funciton is provided for you. This function open a .dat file, read all samples line by line, form and return a big matrix. In order to load the data used in this exercise, you need to call *load_dat* function to load X from *ex1x.dat*, and Y from **ex1y.dat**

Step 2: Preprocess data

For solving linear regression using the **normal equation**, no special data preprocessing is needed. We only need to put an additional 1 to all rows of X (i.e. all the samples) to account for the intercept term. The code for this step is provided for you.

Step 3: Fit the data

Use the **normal equation** to estimate the parameters β. You need to:

- Implement the *pseudo_inverse* function which calculates the pseudo-inverse of a given matrix
- Implement the *sse* function, which calculates the sum of square errors (SSE) between a prediction vector and a reference vector. This function is used to evaluate the model.
- Call the *pseudo_inverse* function you implemented to estimate the parameters β.

Note: you can use the following Numpy functions in your implementation:

- *numpy.matmul*: Calculate multiplication of two matrices
- *numpy.linalg.inv*: Calculate the inverse of a given matrix

You may need to check the online references of these functions for more details.

Step 4: Evaluate the model

After estimating β, you are ready to make predictions and evaluate the model as well as the hypothesis.

- Calculate the predictions according to Eq. (6).
- Calculate the sum of square error of your prediction w.r.t the original scores by calling the equation you implemented in step 3.

Question: What do you think about the prediction, the error and the hypothesis?

Extra step: Fitting on synthetic data

After step 4, you have finished implementing the multivariate linear regression algorithm. Let's apply it to a synthetic data generated with a linear function to see if we can fit the well the data using the algorithm.

The new scores *Y*s are synthetic scores, generated using a linear function of the samples' features. Applying steps 3 and 4 to *X* and *Y*s and evaluate the model/results.