01.112 Machine Learning

Regression

Liang Zheng ft. Rafe Tey

1 Introduction

There are different types of data to be processed in Machine Learning, in this module, you will be learning about **Linear**, **Binary** and **Non-Linear Data**.

Regardless of the kind of data we are dealing with, we will always separate data into Training Data and Test Data (and Validation Data - if we are unable to get an exact solution for our model). **Reason** for this separation is because we want to test how well the model we have created with the training data will perform against the test data and whether we have **under-fit** or **over-fit** our model.

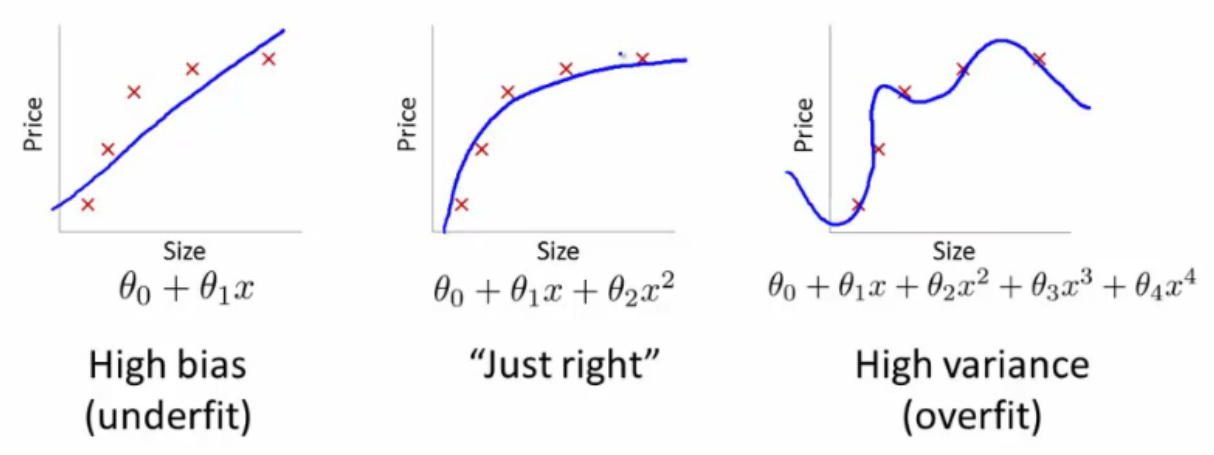

Figure 1: Types of model fit

2 Parameters

Parameters are simply weightages for each data feature: $[X_i]$.
(Note!) Each training data can have multiple features.

3 Generalization

Basically the ultimate **goal** of Machine Learning: finding a predictor that generalizes well.

Finding a θ that optimizes the model: $f(x;\theta,\theta_0) = \theta_d x_d + ... + \theta_1 x_1 + \theta_0 = \theta^T x + \theta_0$
Objective: minimize loss using the loss function $= \mathcal{R}(\hat{f}; S_*) = \frac{1}{n}\Sigma(x,y)\epsilon_{S_*}\frac{1}{2}(y - \hat{f}(x))^2$
(Note!) Least squares is just a way to formulate "loss", you can quantify it with any other method.
(Note2!) The loss in this method simply means prediction loss.

4 Definitions

Training Data: $\{S_n = (x^{(i)}, y^{(i)}) \mid i = 1, ..., n\}$

Simply means $\{S_n\}$ is the set of data used for Training. Where X represents the features of the data and Y represents the output.
(Note: Training Data is also usually written as S')

Training Loss: $\mathcal{R}(\hat{\theta}; S_n) \frac{1}{n} \Sigma_{(x,y)\epsilon S_n} \frac{1}{2}(y - \theta^T x)^2 + \frac{\lambda}{2}\|\theta\|^2$

Test Loss: $\mathcal{R}(\hat{\theta}; S_*) = \frac{1}{n} \Sigma_{(x,y)\epsilon S_*} \frac{1}{2}(y - \hat{\theta}^T x)^2$

Learning Rate, η: a hyper-parameter that controls how much we are adjusting the weights of our network with respect the loss gradient. (A constant given in the question, usually 0.05.)

5 Regression

5.1 Definition

Regression analyses how the typical value of a dependent variable changes when any one of the independent variables are varied. It is used to predict the outcome of an event based on the relationship between variables obtained from the data set.

Regression - height and weight: e.g. the taller the person, the heavier he/she is.

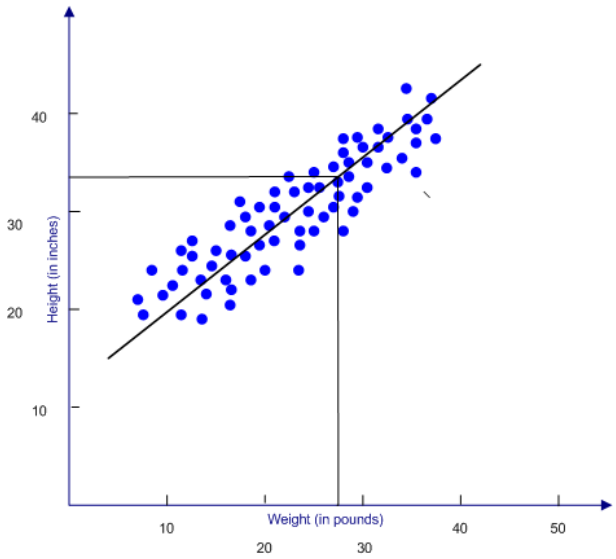

Figure 2: Linear Regression Graph

5.2 Methodology

For all kinds of Regression (Simple and Multivariate) - there are different formulas to formulate your loss function such as **Least Squares**, **Mean Square Error** or even **Mean Absolute Error**. The **Constant Trick Feature** is used to "fold" your model so it is easier to code (it is easier to code a dot product $y = \hat{w} \cdot x$ compared to an equation y = mx + c)

1. Identify your parameters: Gradient, Y-intercept.
For Linear Regression - 1 feature, 1 output.

2. Find your test/training loss function in terms of parameters.

3. Minimize loss: differentiate with respect to the parameters, using the following methods of optimization:
- **Exact Solution**: Basically just finding the gradient and equating it to 0 - the differentiation variable dE/dQ (only when you have an exact solution).
- **Gradient Descent**: used when you have or do not have an exact solution.
- **Stochastic Gradient Descent**: average of point gradients - used from you have no time and dealing with large data.

(Note: Test loss can be different from training loss)

5.2.1 Exact Solution

If there are no constraints on the parameters:
1. Set the gradient to zero, and solve for the parameters.
2. Run through all the solutions to find the parameter that has the smallest training loss.

5.2.2 Gradient Descent

It is a minimization algorithm that minimizes a given function.

1. Initialize θ randomly.
2. Update $\theta \leftarrow \theta$ - $\eta_k \nabla \mathcal{L}_n(\theta)$ where k is the number of iterations you have to do. By substrating $\eta_k \nabla \mathcal{L}_n(\theta)$, we are moving against the gradient towards the minimum.
(Note: Go from θ into the direction of a negative gradient e.g. $-\nabla \mathcal{L}(\theta)$
3. Repeat step (2) until it converges to a local minimum.
Example:

```
# Step 1: Import required libraries.
import numpy as np
import matplotlib.pyplot as plt

# Step 2: Define functions required.
def est_coeff(x, y):
    # number of observations/points
    n = np.size(x)

    # mean of x and y vector
    m_x, m_y = np.mean(x), np.mean(y)

    # calculating cross-deviation and deviation about x
    SS_xy = np.sum(y*x - n*m_y*m_x)
    SS_xx = np.sum(x*x - n*m_x*m_x)
```

```
    # calculating regression coefficients
    b_1 = SS_xy / SS_xx # gradient
    b_0 = m_y - b_1*m_x # y-intercept

    return(b_0, b_1)

# Step 3: Plot graph based on calculated values.
def plot_regression_line(x, y, b):
    # plotting the actual points as scatter plot
    plt.scatter(x, y, color = "m",
                marker = "o", s = 30)

    # predicted response vector
    y_pred = b[0] + b[1]*x

    # plotting the regression line
    plt.plot(x, y_pred, color = "g")

    # putting labels
    plt.xlabel('x')
    plt.ylabel('y')

    # function to show plot
    plt.show()

def main():
    # observations
    x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
    y = np.array([1, 3, 2, 5, 7, 8, 8, 9, 10, 12])

    # estimating coefficients
    b = est_coeff(x, y)
    print("Estimated coefficients:\nb_0 = {}  \
          \nb_1 = {}".format(b[0], b[1]))

    # plotting regression line
    plot_regression_line(x, y, b)

# Step 4: Write and call main function
if __name__ == "__main__":
    main()
```

5.2.3 Stochastic Gradient Descent

The same steps as Gradient Descent but instead of using the average of point gradients, we will be using only a mini-set (subset of the training set).

1. Initialize θ randomly.
2. Update $\theta \leftarrow \theta$ - $\eta_k \nabla \mathcal{L}_m(\theta; \mathcal{B}_m)$ where k is the number of iterations you have to do and B_m is the mini batch of data selected from S_n randomly.
3. Repeat step (2) until it converges to a local minimum.

6 Multivariate Linear Regression

6.1 Definition

Multivariate Linear Regression is a technique that estimates a single regression model with more than one feature variable. When there is more than one predictor variable in a multivariate regression model, the model is a multivariate multiple regression. (thanks Google!)

Model to optimize: $f(x;\theta,\theta_0) = \theta_1 x_1 + ... + \theta_1 x_1 + \theta_0 = \tilde{\theta}^T \tilde{x}$
Objective: minimize loss using the loss function $= \mathcal{R}(\hat{f}; S_*) = \frac{1}{n}\Sigma_(x,y)\epsilon_{S_*}\frac{1}{2}(y - \hat{f}(x))^2$

$$\mathcal{L}_n(\tilde{\theta}; S_n) = \frac{1}{n}\Sigma_{(x,y)\epsilon S_n}\mathcal{L}_1(\tilde{\theta}; \tilde{x}, y)$$

6.2 Methodology

Refer to 5.2 as it is literally the same method, logic and steps but for data sets with more than 1 feature: meaning dimension of X is more than 1.
(Recall: $\nabla\mathcal{L}(\theta; S_n) = \frac{1}{n}\Sigma_{(x,y)\epsilon S_n}\nabla\mathcal{L}(\theta; x, y)$)

6.3 Exact Solution

Use this when optimization problem is convex, so the minimum is attained when the gradient is zero.
Where: y is your output and x is your features along with their dimensions.

$$\nabla\mathcal{L}_n(\hat{\theta}) = 0$$
$$\frac{1}{n}(X^T X) = \frac{1}{n}X^T Y$$
$$\hat{\theta} = (X^T X)^{-1} X^T Y$$

$$\begin{bmatrix} y0 \\ y1 \\ ... \\ yd \end{bmatrix} \begin{bmatrix} x_0^0 & x_0^1 & ... & x_0^d \\ x_1^0 & x_1^1 & ... & x_1^d \\ x_2^0 & x_2^1 & ... & x_2^d \end{bmatrix}$$

Things to note:
1. $X^T X$ needs to be invertible with more data than features.
(Note: You can use regularization if it is not invertible.)
2. Use SGD if $X^T X$ is too large.

6.4 Gradient Descent

Same method, refer to 5.2.2;

$$\theta \leftarrow \theta - \eta_k[\tfrac{1}{n}(X^T X)\theta - \tfrac{1}{n}X^T Y]$$

7 Regularization

Regularization is to simplify your parameters so that its magnitude does not go too high.

7.1 Definition

Ridge Regression: Adds a penalty to the model of theta when it becomes too big, hence, helping to simplify the theta value, resulting in less memory space taken up during training.

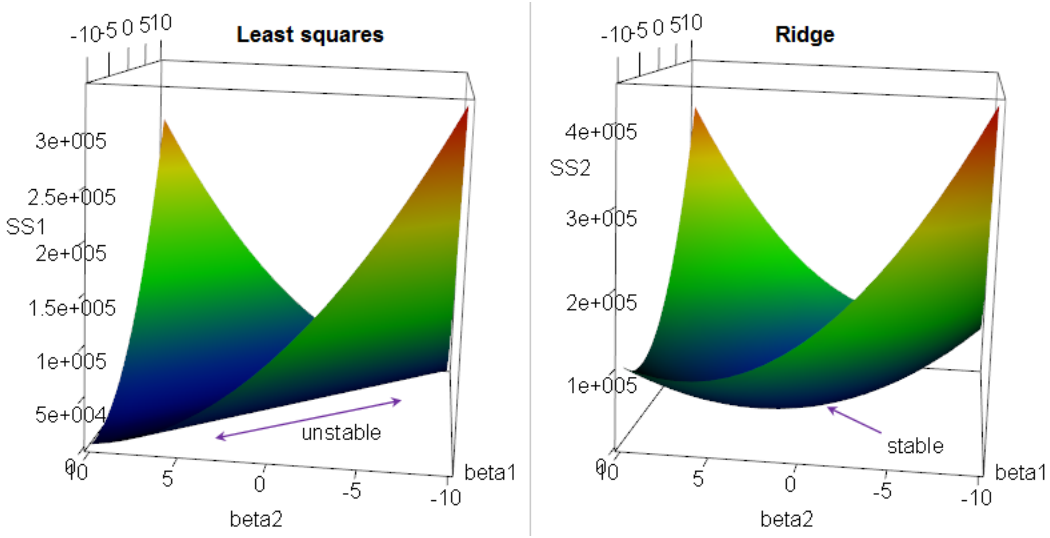

Figure 3: Ridge Regression

Training Error: The training error is the test loss applied to the training set, and it may be different from the training loss.

$$\mathcal{R}(\hat{\theta}; S_n) = \tfrac{1}{n}\Sigma_(x, y)\epsilon S_n \tfrac{1}{2}(y - \hat{\theta}^T x)^2$$

Validation Set: For model selection, e.g. picking λ in ridge regression. Acts as a proxy for test set, $\mathcal{S}_{val}$.

Validation Loss: The validation loss is the test loss applied to the validation set.

$$\mathcal{R}(\hat{\theta}; S_{val}) = \frac{1}{n}\Sigma_{(x,y)\epsilon S_{val}}\frac{1}{2}(y - \hat{\theta}^T x)^2$$

Hyperparameters: They are often used in processes to help estimate model parameters, for Ridge Regression, it is your Regularization parameter = λ.

7.2 Exact Solution

$\nabla\mathcal{L}_{n,\lambda}(\theta) = \lambda\theta + \frac{1}{n}(X^T X)\theta - \frac{1}{n}X^T Y$

7.3 Gradient Descent

$\theta \leftarrow$(1 - $\eta_k\lambda)\theta - \eta_k[\frac{1}{n}(X^T X)\theta - \frac{1}{n}X^T Y]$
Without regularization, e.g. $= 0$, this shrinkage factor $(1 - \eta_k\lambda)\theta$ equals 1.

7.4 Effects of Regularization

Regularization solves the problem of overfitting in statistical models as it discourages complexity by disregarding stochastic noise, even if it means picking a less-accurate rule according to the training data.

That's it for Week 1!