8.2 Linear Regression

Linear regression is a predictive analysis task of modeling the relationship between predictor variables (*X*) and real number response variable (*Y*). Using this model, we can predict the value of *Y* using *X*.

In regression each data vector which is represented by $\boldsymbol{X^i} \in \mathbb{R}^n$

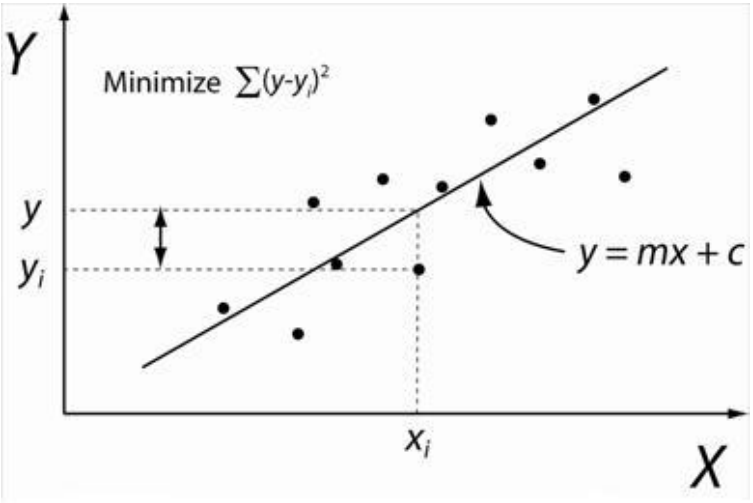
$$X^i = (X_1^i, X_2^i ..... X_n^i) \; is\, n\, dimensional\, data\, vector$$

is associated with real value response variable $\boldsymbol{Y_i} \in \mathbb{R}$ as shown below

$$\left(\boldsymbol{X^1}, \boldsymbol{Y_1}\right), \left(\boldsymbol{X^2}, \boldsymbol{Y_2}\right), ..., \left(\boldsymbol{X^n}, \boldsymbol{Y_n}\right),$$

here $\boldsymbol{X^i}$ represents i^{th} data point; $\boldsymbol{X_n^i}$ represents n^{th} attribute of i^{th} data point.

Following diagram shows what actually linear regression is. Line represented by the equation ***y=mx+c*** is called a regression line



Task: is to predict *Y* as a linear function of *X*, in equation it can be written as

$$\widehat{Y} = h_w(X) = W^T X = w_1 x_1 + w_2 x_2 + ... + w_n x_n$$

$$\widehat{Y}\, is\, predicted\, value$$

$$W = \begin{bmatrix} w_1 \\ w_2 \\ . \\ . \\ w_n \end{bmatrix} \; is\, weight\, matrix$$

$$X = \begin{bmatrix} x_1^1 & . & . & x_n^1 \\ x_1^2 & . & . & x_n^2 \\ . & . & . & . \\ . & . & . & . \\ x_1^m & . & . & x_n^m \end{bmatrix} \; is\, data\, matrix\, each\, row\, represents\, a\, data\, vector$$

$$For\, simplicity\, I\, have\, removed\, w_0\, and\, x_0^i$$

Note: Above equation is same as the equation in **7.7 Curve Fitting and Error Function (https://iu.instructure.com/courses/1714526/pages/7-dot-7-curve-fitting-and-error-function)** , but here I have removed x_0 and w_0 for simplicity.

Goal: Is to find the optimal weight $\widehat{W}$ which minimize the Sum of Squared Error which is represented by following equation

$$S.E = \frac{1}{2}\sum_{i=1}^{i=m}(y_i - \widehat{y_i})^2$$

$$\widehat{y_i} = predicted\,value = W^T X^i$$

$$W^T\,is\,transpose\,of\,weight\,vector$$

$$\text{Object: } \widehat{W} = avg\,min\frac{1}{2}\sum_{i=1}^{i=m}(y_i - W^T X^i)^2$$

To find $\widehat{W}$ either we can use gradient descent, as discussed in unit 7, or we can use the following normal equation

$$X^T X\widehat{W} = X^T Y\,[normal\,equation]$$

$$\widehat{W} = (X^T X)^{-1} X^T Y$$

$$In\,the\,above\,normal\,equation\,Y\,is\,response\,variable$$

$$Y = \begin{bmatrix} y_1 \\ y_2 \\ . \\ . \\ y_m \end{bmatrix} \quad \widehat{W} = \begin{bmatrix} \widehat{w_1} \\ \widehat{w_2} \\ . \\ . \\ \widehat{w_n} \end{bmatrix}$$

$$Y\,is\,(m \times 1)\,matrix$$

$$\widehat{W}\,is\,(n \times 1)\,matrix$$

From $\widehat{W}$ we can calculate $\widehat{Y}$ using following equation

$$\widehat{Y} = h_{\widehat{w}}(X) = \widehat{W}^T X = \widehat{w_1}x_1 + \widehat{w_2}x_2 + ... + \widehat{w_n}x_n$$

Note: Derivation of normal equation can be seen at following web page

http://eli.thegreenplace.net/2014/derivation-of-the-normal-equation-for-linear-regression/ (http://eli.thegreenplace.net/2014/derivation-of-the-normal-equation-for-linear-regression/)

Problem Statement: We have been given a data file related to concrete compressive strength. It has following information about concrete

1. Cement (component 1) -- quantitative -- Input Variable
2. Blast Furnace Slag (component 2) -- quantitative -- Input Variable
3. Fly Ash (component 3) -- quantitative -- Input Variable

4. Water (component 4) -- quantitative -- Input Variable
5. Superplasticizer (component 5) -- quantitative -- Input Variable
6. Coarse Aggregate (component 6) -- quantitative -- Input Variable
7. Fine Aggregate (component 7) -- quantitative -- Input Variable
8. Age -- quantitative -- Day (1~365) -- Input Variable

Our task is to predict “Concrete compressive strength – quantitative” which is the first column in the concrete.csv file and is named “ccs”. In a nutshell we have 8 predictor variables or attributes using that we have to find “ccs”

The data has been taken from
https://archive.ics.uci.edu/ml/datasets/Concrete+Compressive+Strength
(https://archive.ics.uci.edu/ml/datasets/Concrete+Compressive+Strength)

We can solve this problem by breaking it in following steps :

1. Normalize the data to bring all the attributes on the same scale.
2. Filter the data and target variable from the input file. In this step, we separate the 8 predictor variables from the target variable.
3. Then we calculate optimal weight using normal equation.
4. By putting the optimal weight in hypothesis function we calculate the predicted value of target variable
5. At the end, we calculate S.S.E to find how good our prediction was.

```
import pandas
import os
from numpy.linalg import inv
import numpy as np
relativePath = os.getcwd()
dataFilePath = relativePath + "/Resources/concrete.csv"
data = pandas.read_csv(dataFilePath)

# http://archive.ics.uci.edu/ml/datasets/Bike+Sharing+Dataset
# http://archive.ics.uci.edu/ml/datasets/Wine+Quality


def Z_ScoreNormalization(data, targetColName):  # we need to normalize the data so that each attribute is brought to same scale
    for i in data.columns:
        if i == targetColName:  # do not modify target values
            continue
        mean = data[i].mean()
        std = data[i].std()
        data[i] = data[i].apply(lambda d: float(d - mean) / float(std))  # perform z-score normalization


def filterData(dataFrame, targetVarName):  # we need to remove target value from the data frame and need to put x0 values.
    targetValues = dataFrame[targetVarName]
    colum = range(1, len(dataFrame.columns))  # all the column except 0th which is target
    data = dataFrame[dataFrame.columns[colum]]  # keep all column except 0th
    data.insert(0, "x0", [1] * len(dataFrame))  # in 0th column set all 1 and name the column as X0 this is to accommodate biases weights i.e. w0
    return targetValues, data  # return target value and modified dataframe
```

```
def calOptimalWeight(dataFrame, target):  # this function calculates W'=(X^TX)^-1X^TY
    innerProduct = dataFrame.T.dot(dataFrame)  # calculating X^TX
    inverse = inv(innerProduct)  # calculate (X^TX)^-1
    product = inverse.dot(dataFrame.T)  # (X^TX)^-1X^T
    weight = product.dot(target)  # (X^TX)^-1X^TY
    print weight.shape
    return weight  # return W'


def predict(weights, X):  # this function calculates Y'=W^TX or y'=XW
    predictedValue = X.dot(weights)
    return predictedValue


def calSSE(target, predicted):  # calculate sum of squared error
    m = len(target)
    SSE = ((np.asarray(target) - np.asarray(predicted)) ** 2) / float(2 * m)
    return sum(SSE)


Z_ScoreNormalization(data, 'ccs')  # step 1 Normalize
Y, X = filterData(data, 'ccs')  # Step 2 Filter the data i.e. seperate data from target value
optimalWeight = calOptimalWeight(X, Y)  # Step 3 calculate W'
predictedValue = predict(optimalWeight, X)  # Step 4 predict value using W'
print calSSE(Y, predictedValue)  # calculate S.S.E using actual target value Y and predicted target value Y'
```