Simple Linear Regression

Lecture 14

Centre for Data Science, ITER
Siksha 'O' Anusandhan (Deemed to be University), Bhubaneswar, Odisha, India.

Contents

1. Introduction
2. Simple Linear Regression model
3. Gradient Descent
4. Maximum Likelihood Estimation

Correlation:

- Correlation function to measure the strength and direction of the linear linear relationship between two variables.
- The value of the correlation between two variable lies between -1 to 1.
- If the two variables move in the same direction, then those variables are said to have a positive correlation. If they move in opposite directions, then they have a negative correlation.

What is the need of linear regression ?

Simple Linear Regression model

- The model for linear regression with one predictor variable can be stated as follows:

$$y_i = \beta x_i + \alpha + \epsilon_i \ , \qquad \text{where}$$

- y_i is the value of the response variable in the ith trial.
- $\alpha and \beta$ are parameters.
- x_i is known, it is the value of the predictor variable in the ith trial.
- ϵ_i is a random error term ,normally distributed with mean,E(ϵ_i) = 0 and variance, V(ϵ_i) = σ^2
- Cov(ϵ_i, ϵ_j) =0 if i $\neq$ j for i=1,...,n and j=1,...,n.

The above regression model is said to be simple linear regression because it is linear in terms of parameters.

Examples of Simple Linear Regression model:

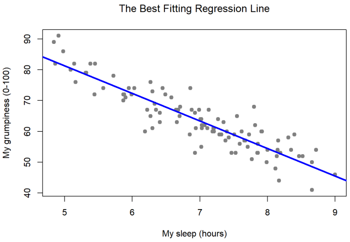

Figure 1: first figure

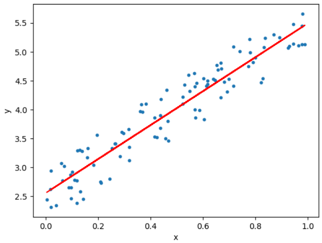

Figure 2: second figure

Simple Linear Regression (Contd.)

Program to plot regression line and scatter plot :

```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

x=[1,2,3,4,9]
y=[5,2,9,7,3]

plt.figure()
sns.regplot(x,y,fit_reg = True)
plt.scatter(np.mean(x),np.mean(y),color='green')
```

Simple Linear Regression (Contd.)

Least square Estimation of α and β :

$$\widehat{\beta} = \frac{Correlation(x, y) * \sigma_y}{\sigma_x}$$

$$\widehat{\alpha} = \overline{y} - \widehat{\beta}\overline{x}\ , \text{ where}$$

$\overline{y}$= mean of y
$\overline{x}$= mean of x
σ_y =standard deviation of y
σ_x =standard deviation of x

Simple Linear Regression (Cont.)

Program to find the least square estimates of α and β :
1.By Formula:

```
import numpy as np
x = np.array([5, 15, 25, 35, 45, 55])
y = np.array([5, 20, 14, 32, 22, 38])

from typing import Tuple
from scratch.linear_algebra import Vector
from scratch.statistics import correlation, standard_deviation,
    mean
"""Given two vectors x and y,
 find the least-squares values of alpha and beta"""

def least_squares_fit(x: Vector, y: Vector) -> Tuple[float,
    float]:
    beta = correlation(x, y) * standard_deviation(y) /
    standard_deviation(x)
    alpha = mean(y) - beta * mean(x)
    return alpha, beta
```

Program to predict for any value of x and to find the error :

```
def predict(alpha: float, beta: float, x_i: float) -> float:
    return beta * x_i + alpha
""" The error from predicting beta * x_i + alpha
when the actual value is y_i"""
def error(alpha: float, beta: float, x_i: float, y_i: float) ->
     float:
     return predict(alpha, beta, x_i) - y_i

from scratch.linear_algebra import Vector
def sum_of_sqerrors(alpha: float, beta: float, x: Vector, y:
     Vector) -> float:
     return sum(error(alpha, beta, x_i, y_i) ** 2
     for x_i, y_i in zip(x, y))

m1=least_squares_fit(x,y)
predict(m1[0], m1[1], 20)
E=error(m1[0], m1[1],x,y)
```

Simple Linear Regression (Cont.)

Program to find the least square estimates of α and β :
2.By Inbuilt module of python:

```
import statsmodels.api as s
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import statsmodels.formula.api as sm

x = np.array([5, 15, 25, 35, 45, 55]).reshape((-1, 1))
y = np.array([5, 20, 14, 32, 22, 38])

x= s.add_constant(x)

model1=s.OLS(y, x)
result1=model1.fit()
print(result1.summary())
```

Simple Linear Regression (Cont.)

Test the least square module:

```
x = [i for i in range(-100, 110, 10)]
y = [3 * i - 5 for i in x]
# Should find that y = 3x - 5
p= least_squares_fit(x, y)
assert least_squares_fit(x, y) == (-5, 3)
```

Use of assert statement in least square method:

```
from scratch.statistics import num_friends_good,
    daily_minutes_good
alpha, beta = least_squares_fit(num_friends_good,
    daily_minutes_good)
assert 22.9 < alpha < 23.0
assert 0.9 < beta < 0.905
```

Simple Linear Regression (Cont.)

Calculation of R-square:

```
from scratch.statistics import de_mean
def total_sum_of_squares(y: Vector) -> float:
#"""the total squared variation of y_i's from their mean"""
    return sum(v ** 2 for v in de_mean(y))
def r_squared(alpha: float, beta: float, x: Vector, y: Vector) -> float:
    return 1.0 - (sum_of_sqerrors(alpha, beta, x, y) / total_sum_of_squares(y))

#"""the fraction of variation in y captured by the model, which equals
#1 - the fraction of variation in y not captured by the model
"""

rsq = r_squared(alpha, beta, num_friends_good, daily_minutes_good)
assert 0.328 < rsq < 0.330
```

Simple Linear Regression (Cont.)

R-square:

- R-square value lies between 0 to 1
- The model whose R-squared value close to 1 indicates that the model is a maximum variance can be explained using the model. If the R-square value close to 0 indicates that is the variance of response can't be explained using x.

Simple Linear Regression (Cont.)

R-square comparison between two plots:

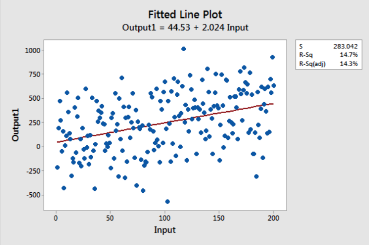

Figure 3: R^2 close to 0

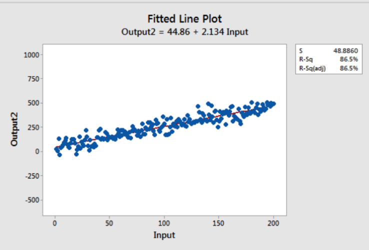

Figure 4: R^2 close to 1

Gradient Descent :

Program to estimate the parameters using gradient and Descent:

```
import random
import tqdm
from scratch.gradient_descent import gradient_step
num_epochs = 10000
random.seed(0)
guess = [random.random(), random.random()] # choose random
    value to start
learning_rate = 0.00001
with tqdm.trange(num_epochs) as t:
    for _ in t:
        alpha, beta = guess
    # Partial derivative of loss with respect to alpha
        grad_a = sum(2 * error(alpha, beta, x_i, y_i)
                     for x_i, y_i in zip(num_friends_good,
    daily_minutes_good))
```

Gradient Descent (Cont...) :

Program to estimate the parameters using gradient and Descent:

```
# Partial derivative of loss with respect to beta
        grad_b = sum(2 * error(alpha, beta, x_i, y_i) * x_i
                     for x_i, y_i in zip(num_friends_good,
                                         daily_minutes_good))
# Compute loss to stick in the tqdm description
        loss = sum_of_sqerrors(alpha, beta,
                               num_friends_good, daily_minutes_good)
        t.set_description(f"loss: {loss:.3f}")
# Finally, update the guess
    guess = gradient_step(guess, [grad_a, grad_b], -learning_rate)
# We should get pretty much the same results:
alpha, beta
assert 22.9 < alpha < 23.0
assert 0.9 < beta < 0.905
```

Maximum Likelihood Estimation :

- Imagine that we have a sample of data v_1 , v_2, ..., v_n that comes from a distribution that depends on some unknown parameter θ .
- If θ is unknown then we can consider the likelihood of θ given the sample that is

  $$L\left(\theta | v_1, ..., v_n\right)$$

  Under this approach, the most likely is the value that maximizes this likelihood function.
- As per the assumption of the Linear regression model, the errora are normally distributed with mean 0 and standard deviation σ. So the likelihood of α and β based on x and y data set is :

$$L\left(\alpha, \beta | x_i, y_i, \sigma\right) = \frac{1}{\sqrt{2\pi}\sigma} exp\left(-\left(y_i - \alpha - \beta x_i\right)^2 / 2\sigma^2 \right)$$

Thank You
Any Questions?