INTRODUCTION TO ARTIFICIAL INTELLIGENCE

in Python LANGUAGE

Chapter 2: Regression

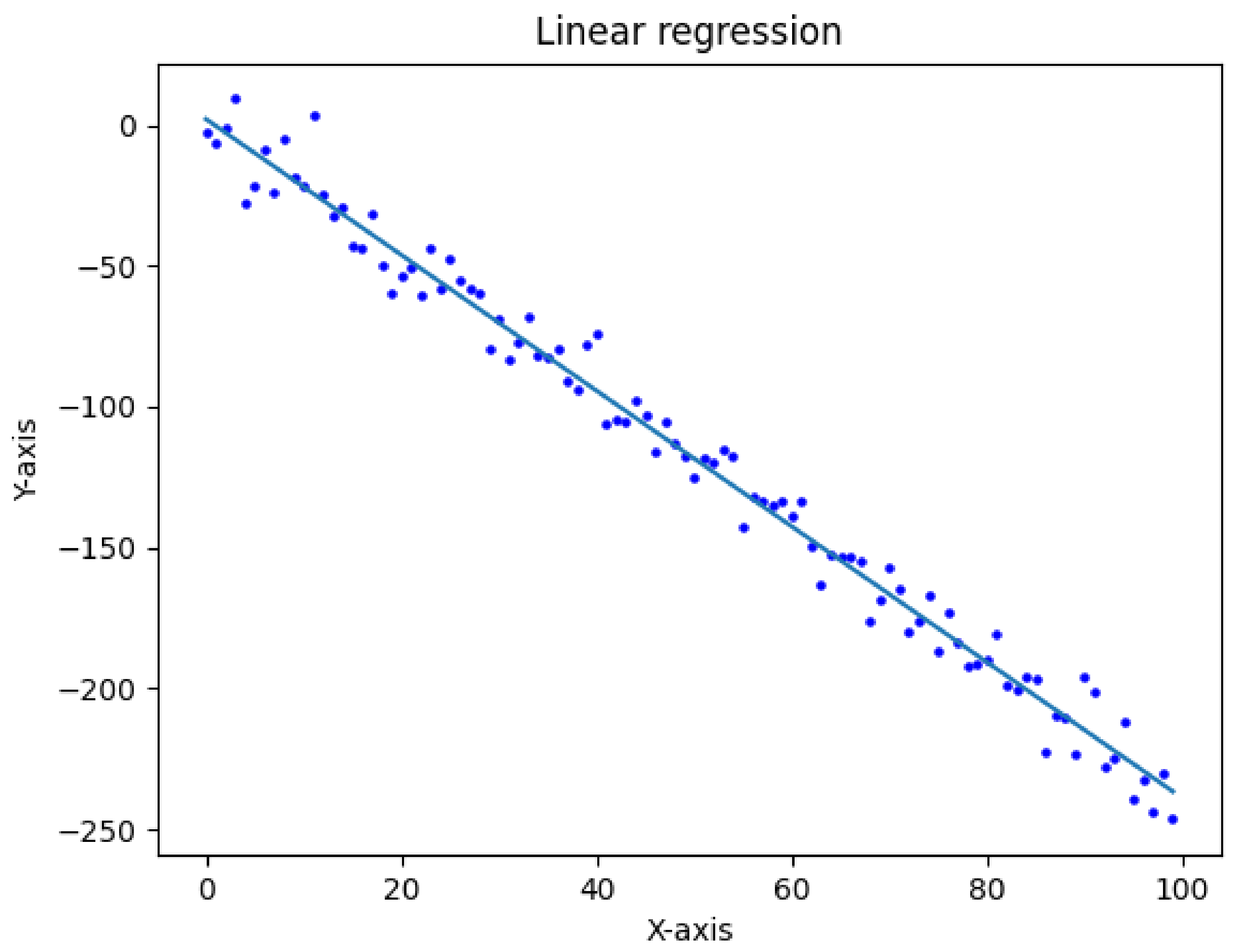
Outline

1. Regression in Machine Learning
2. Mathematical modeling
3. Example of Linear Regression on Python

CS 221 - Reflex-based Models Cheatsheet (stanford.edu)

Regression in machine learning

- In machine learning, regression can be used to solve a task of predicting a continuous quantity.
- In a regression problem: One try to predict the output by proposing a function y_hat=f(x_1,x_2,…,x_n) that best fit all the data points (x_1,x_2,…,x_n,y).
- In the simplest case, in 2D, we apply linear regression (linear fitting / linear least square) algorithm to find the line equation y_hat=f(x) that best fits all the data points (x_i,y_i)



2.2. Mathematical modeling

2.2.1. Input / parameter vectors

$$\mathbf{x} = [x_n, x_{n-1}, \ldots, x_2, x_1, x_0]\text{: } \textit{Input vector}$$
$$\boldsymbol{\omega} = [\omega_n, \omega_{n-1}, \ldots, \omega_2, \omega_1, \omega_0]\text{: } \textit{Parameter vectors (to be optimized)}$$
$$\hat{y} = \mathbf{x}.\boldsymbol{\omega}^{\mathbf{T}}\text{: } \textit{estimated output}$$

- We will try to optimize the parameters vector **w** in such a way that the estimated output will converge to the real output y. One usual and effective technique for that is using **gradient descent algorithm**.

2.2.2. Gradient descent algorithm

Gradient is the vector composed by all partial derivatives of the function. Denotation: Nabla ∇

Example: $f(x,y,z) = 3x^3z - y^2 + 5z + 2yz$

$$\nabla f(x,y,z) = \begin{bmatrix} \frac{\partial}{\partial x} f(x,y,z) \\ \frac{\partial}{\partial y} f(x,y,z) \\ \frac{\partial}{\partial z} f(x,y,z) \end{bmatrix} = \begin{bmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \\ \frac{\partial}{\partial z} \end{bmatrix} f(x,y,z) = \begin{bmatrix} 9x^2z \\ -2y + 2z \\ 3x^3 + 5 + 2y \end{bmatrix}$$

Gradient descent algorithm: An optimal algorithm that help finding the local minimum of a function (in the case of a NN: the loss function) by making converging the model's parameters to optimal values.

2.2.2. Gradient descent algorithm

Gradient descent algorithm: Starting from a point that could be close to the solution, one will use an iterative operation to gradually approach the desired point (local minimum), i.e., when the derivative converge to 0.

In order to converge to the local minimum, one have to move in the inverse sense of the gradient vector. The formula can be as follow:

$$\text{for each } x_i \text{ in } \mathbf{x}: \quad x_i(t+1) = x_i(t) - LR.\frac{\partial f}{\partial x_i}(\mathbf{x})$$

with LR is the ***learning rate***

In the case of a regression algorithm, *f* is the **loss function** i.e. the sum of all quadratic errors

2.2.3. The loss function

In a regression problem, the loss function is defined to be the sum (or the mean value) of the all the quadratic errors of the output data. The formula for the calculation of the loss function is as follow:

$$L(\mathbf{w}) = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \quad with \quad \hat{y}_i = \mathbf{x^i}.\boldsymbol{\omega}^{\mathbf{T}} = (x_n^i \quad \ldots \quad x_1^i \quad x_0^i).(\omega_n \quad \ldots \quad \omega_1 \quad \omega_0)^T$$

In 2D, the formula becomes:

$$L(\mathbf{w}) = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \quad with \quad \hat{y}_i = a.x_i + b$$

2.2.4. How to estimate the best fit equation

In order to estimate the optimal equation that best fit the data point, one should update the parameter vector by using the gradient computed form the loss function. The formula can be expressed as follow:

$$\text{for each } \omega_i \ in \ \mathbf{w}\text{:} \ \omega_i(t+1) = \omega_i(t) - LR.\frac{\partial L}{\partial \omega_i}$$

Once all the parameters were updated, one can re-calculate the loss function and then re-update the parameters again. This process can be stopped after reaching the limit of epochs fixed in advanced or once the loss function has already converged to a stable value.

2.3.1. How to update parameters in a linear regression

In 2D, the formula for updating the 2 parameters of the line can be expressed as follow:

$$\begin{cases} a = a - LR.\dfrac{\partial L}{\partial a} \\ b = b - LR.\dfrac{\partial L}{\partial b} \end{cases} \quad with \quad L = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 = \sum_{i=1}^{n}\big(y_i - (ax_i + b)\big)^2$$

with: $$\frac{\partial L}{\partial a} = \sum_{i=1}^{n} -2x_i\big(y_i - (ax_i + b)\big)\,; \quad \frac{\partial L}{\partial b} = \sum_{i=1}^{n} -2\big(y_i - (ax_i + b)\big)$$

2.3.2. Code Python Example

Read data from .cvs file

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

data = pd.read_csv('points_line.csv')

print(data.head())

studytime = data['studytime'].to_numpy()  # Convert to NumPy array
score = data['score'].to_numpy()          # Convert to NumPy array

print(studytime)
print(score)
```

2.3.2. Code Python Example

Define the loss function

```
def loss(m,b,X,Y):
    total_error = 0
    for i in range(len(X)):
        total_error += (Y[i] - (m*X[i]+b))**2
    total_error = total_error / float(len(X))
    return total_error
```

2.3.2. Code Python Example

Define the gradient descent optimization function

```
def gradient_descent(a_now, b_now, X, Y, LR):
    a_gradient = 0
    b_gradient = 0
    n = len(X)
    for i in range(n):
        a_gradient += -(2/n) * X[i] * (Y[i] - (a_now * X[i] + b_now))
        b_gradient += -(2/n) * (Y[i] - (a_now * X[i] + b_now))
    a = a_now - a_gradient*LR
    b = b_now - b_gradient*LR
    return a,b
```

2.3.2. Code Python Example

Run the algorithm now …

```
# Run the algorithm
a = 0
b = 0
LR = 0.0003
epochs = 40000

for i in range(epochs):
    if (i%1000 ==0):
        loss_val = loss(a,b,studytime,score)
        print(f'epochs: {i}' + f', loss: {loss_val:.2f}')
        #print(f'epochs: {i}')
    a, b = gradient_descent(a,b,studytime,score,LR)

print(f'The estimated line equation is: y = {a:.3f}x + ' + f'{b:.3f}')
```

Introduction to Artificial Intelligence

END OF CHAPTER 2