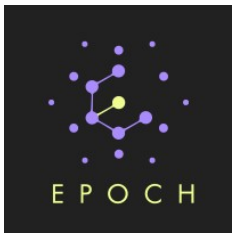

భారతీయ సాంకేతిక విజ్ఞాన సంస్థ హైదరాబాద్
भारतीय प्रौद्योगिकी संस्थान हैदराबाद
Indian Institute of Technology Hyderabad

Learning phase in EPOCH

REPORT

Report by

Jayachandra Naidu Rajapu

Student ID CS21BTECH11050

Academic Year
2022-2023

Abstract

Sommario

Contents

List of Figures

List of Tables

List of Algorithms

List of Code Snippets

List of Acronyms

CSV Comma Separated Values

1 Linear Regression

1.1 Introduction

Machine learning is predictive modelling. We will create models which will predict outcome based on the model's learning from the training data. Sometimes we have to predict the outcome which is a continous variable. Linear regression is a simple machine learning model which establishes a linear relation between data and outcome. We will discuss the details of establishing the relation based on training data with detailed mathematics and python code.

1.2 Model

The model is constructed by simply taking some arbitrary coefficients for creating a linear formula and updating it repeatedly to reach the required state to predict outcome. Everything is explained further.

1.2.1 Terms used frequently

The following are the terms which we will be frequently using throughout this discussion:

- Features
  The input data for the model might have more than one parameter which will decide the outcome. Each parameter is called a feature.

- Instance
  The model is mostly trained with a huge set of data. Each individual part of the data with its features and outcome is called an instance.

1.2.2 Mathematical understanding

Let the dataset have n features. The training dataset contains input X, with m instances. We will consider X as a set of vectors of n dimensions. So $\vec{X}^{(i)}$ represents i^{th} instance of the data.

$$\vec{X}^{(i)} = \sum_{j=1}^{n} \vec{x}_j^{(i)} \tag{1.1}$$

Here $\vec{x}_j^{(i)}$ is value of the j^{th} feature at i^{th} instance. We consider vectors here for easy mathematics. The outcome of the X is y which is a set of real numbers. While training, we compare the calculated outcome y' and update the model accordingly.

We consider $\vec{W}$ as a vector with dimension n. b is a real number.

$$y' = \vec{W} \cdot X + b \tag{1.2}$$

where

$$\vec{W} \cdot X = \left[\vec{W} \cdot \vec{X}^{(i)}\right] = \begin{bmatrix} \vec{W} \cdot \vec{X^{(1)}} \\ \vec{W} \cdot \vec{X^{(2)}} \\ \vdots \\ \vec{W} \cdot \vec{X^{(m)}} \end{bmatrix} \tag{1.3}$$

Therefore equation (1.2) can be rewritten as

$$y' = \begin{bmatrix} \vec{W} \cdot \vec{X^{(1)}} + b \\ \vec{W} \cdot \vec{X^{(2)}} + b \\ \vdots \\ \vec{W} \cdot \vec{X^{(m)}} + b \end{bmatrix} \tag{1.4}$$

Our mission here is to find the $\vec{W}$ and b from the given data and use them in

predicting outcomes for any given input data. For that we should have a deeper and clear understanding of the equations.

$$\vec{W} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix} \tag{1.5}$$

$$\begin{aligned} y'_i = \vec{W} \cdot \vec{X^{(i)}} + b &= \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix} \cdot \begin{bmatrix} x_1^{(i)} & x_2^{(i)} & \cdots & x_n^{(i)} \end{bmatrix} + b \\ &= w_1 x_1^{(i)} + w_2 x_2^{(i)} + \cdots + w_n x_n^{(i)} + b \end{aligned} \tag{1.6}$$

Here we can understand w_i as weight of i^{th} feature. The value of each feature is multiplied by it's weight and added to the total. The weight balances the influence of a particular feature on the outcome.

Some features might be generally in higher magnitude, but have a lesser influence on the outcome. Obviously their weight will be small and compensate the high magnitude. Similarly some features might have small magnitudes but higher influence on the outcome. As expected, their weights will be large.

We add b as a constant which will adjust the magnitude of product of weights and features to that of outcome.

1.2.3 Finding W and b

Initially we know nothing about W and b. So, we will assume all the weights to be equal to 0. We will compute the y' and compare it y, the original outcome. We define a cost function which represents the error in the calculated outcome.

$$J(W, b) = \frac{1}{m} \sum_{i=1}^{m} (y_i - y'_i)^2 \tag{1.7}$$

This is called as mean square error. We consider square here to get the absolute value here. We can use other functions like absolue value or 4th power, but square makes the further mathematics simple. Our aim is, as discussed above,

to find the values of W and b where the cost function J is minimum. So, we will update the W and b to achieve minimum using gradient descent. Gradient descent is explained briefly in the next section.

We will find the gradient equation for cost function J.

$$J'(W,b) = \begin{bmatrix} \frac{df}{dW} \\ \frac{df}{db} \end{bmatrix} = \begin{bmatrix} \frac{1}{m}\sum_{i=1}^{m} -2X^{(i)}(y_i - y'_i) \\ \frac{1}{m}\sum_{i=1}^{m} -2(y_i - y'_i) \end{bmatrix} \tag{1.8}$$

Now we update W and b accordingly.

$$W_{new} = W - \alpha\frac{df}{dW} \tag{1.9}$$

$$b_{new} = b - \alpha\frac{df}{db} \tag{1.10}$$

We use the obtained W and b for predicting outcome.

1.2.4 Gradient Descent

Gradient descent is an iterative algorithm for finding minimum value of a function. The algorithm roughly works in the following steps:

1. Choose a starting point.
2. Calculate the gradient at this point.
3. Make a scaled step against the gradient.
4. Repeat the steps 2 and 3 until minimum is reached, i.e. the gradient reaches almost 0.

Gradient

Gradient is slope of the function at a given point in a given direction. Gradient of a n-dimensional function $f(p)$ is given

$$\nabla f(p) = \begin{bmatrix} \frac{\partial f}{\partial x_1}(p) \\ \vdots \\ \frac{\partial f}{\partial x_n}(p) \end{bmatrix} \tag{1.11}$$

Algorithm

In the ith iteration,

$$p_{i+1} = p_i - \alpha \nabla f(p_i) \tag{1.12}$$

Where α is the learning rate which controls the step size. The learning rate should be chosen carefully according to number of iterations as

- Small learning rate might make the algorithm stop, i.e reach the maximum iteration, before reaching the minimum.
- Large learning rate might make the algorithm jump too much and thus might skip the minimum.

1.3 Questions for curiosity

2 Logistic Regression

2.1 Introduction

As discussed in the previous chapter, we will build a model which will predict the outcome based on the model's learning from training dataset. Logistic Regression is used for predicting discrete values (only 0 an 1 here) unlike continuos values in Linear Regression. It simply calculates the probabilty of the value, calculated from Linear Regression, lying on the extremes and output it as 0 and 1.

2.2 Model

The model is simply a little additional calculation on Linear Regression. We use sigmoid function to restrict the outcome between 0 and 1. If the outcome is close to 1 then we treat it as 1 and 0 otherwise.

2.2.1 Sigmoid Function

For restricting the real number value from the outcome of Linear Regression between 0 and 1, we use Sigmoid function. It takes real numbers as input and outputs continuos values between 0 and 1.

$$Sigmoid(x) = \frac{1}{1 + e^{-x}} \tag{2.1}$$

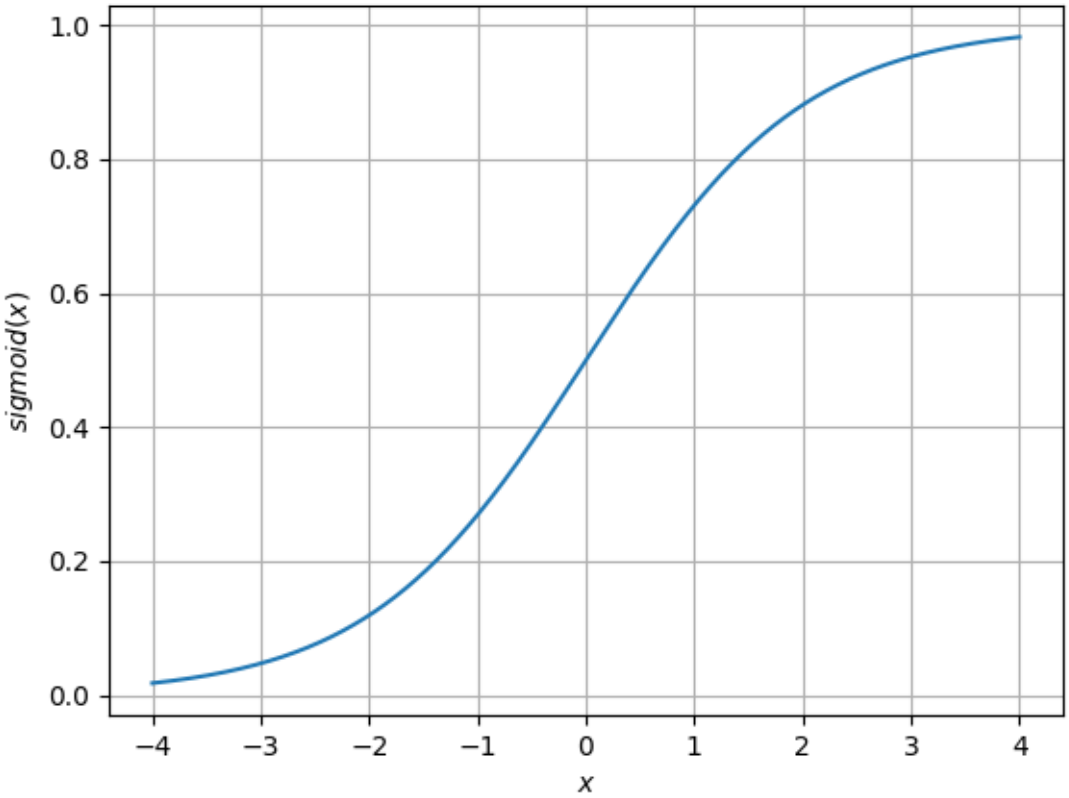

Figure 2.1: Sigmoid function graph

The graph of Sigmoid function is shown in figure 2.1

2.2.2 Mathematical Understanding

3 Introduction

Random citation [1].
Random footnote.[1]

3.1 A section

Example of list

- Item 1
- Item 2

3.1.1 A subsection

Example of acronym

Comma Separated Values (CSV)

Example of enumeration

1. Item 1
2. Item 2

[1] https://lucamartinelli.eu.org

Example of quote

> *Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.*

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

4

Background

Algorithm 1 An algorithm with caption

Require: $n \geq 0$
Ensure: $y = x^n$
  $y \leftarrow 1$
  $X \leftarrow x$
  $N \leftarrow n$
  while $N \neq 0$ **do**
    if N is even **then**
      $X \leftarrow X \times X$
      $N \leftarrow \frac{N}{2}$ {This is a comment}
    else if N is odd **then**
      $y \leftarrow y \times X$
      $N \leftarrow N - 1$
    end if
  end while

$$e^{j\pi} + 1 = 0 \tag{4.1}$$

5 Analysis

5.1 A section

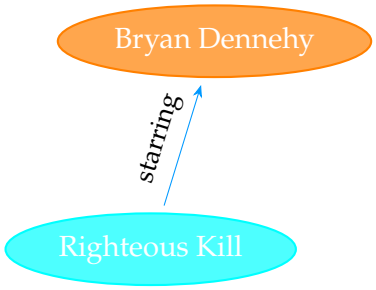

Figure 5.1: Image created with TikZ

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

```
import numpy as np

def incmatrix(genl1,genl2):
    m = len(genl1)
    n = len(genl2)
    M = None #to become the incidence matrix
```

```
    VT = np.zeros((n*m,1), int)  #dummy variable

    test = "String"

    #compute the bitwise xor matrix
    M1 = bitxormatrix(genl1)
    M2 = np.triu(bitxormatrix(genl2),1)

    for i in range(m-1):
        for j in range(i+1, m):
            [r,c] = np.where(M2 == M1[i,j])
            for k in range(len(r)):
                VT[(i)*n + r[k]] = 1;
                VT[(i)*n + c[k]] = 1;
                VT[(j)*n + r[k]] = 1;
                VT[(j)*n + c[k]] = 1;

                if M is None:
                    M = np.copy(VT)
                else:
                    M = np.concatenate((M, VT), 1)

                VT = np.zeros((n*m,1), int)

    return M
```

Code 5.1: Code snippet example

6

Conclusions and Future Works

A	**B**
C	D
E	F
G	H

Table 6.1: Table example

References

[1] Marco Alecci et al. “Development of an IR System for Argument Search.” In: *CLEF (Working Notes)*. 2021, pp. 2302–2318.

Acknowledgments