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.

$$\nabla J(W,b) = \begin{bmatrix} \frac{\partial J}{\partial W} \\ \frac{\partial J}{\partial b} \end{bmatrix} = \begin{bmatrix} \frac{1}{m}\sum_{i=1}^{m} -2X^{(i)}(y_i - y_i') \\ \frac{1}{m}\sum_{i=1}^{m} -2(y_i - y_i') \end{bmatrix} \tag{1.8}$$

Now we update W and b accordingly.

$$W_{new} = W - \alpha\frac{\partial J}{\partial W} \tag{1.9}$$

$$b_{new} = b - \alpha\frac{\partial J}{\partial b} \tag{1.10}$$

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.

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

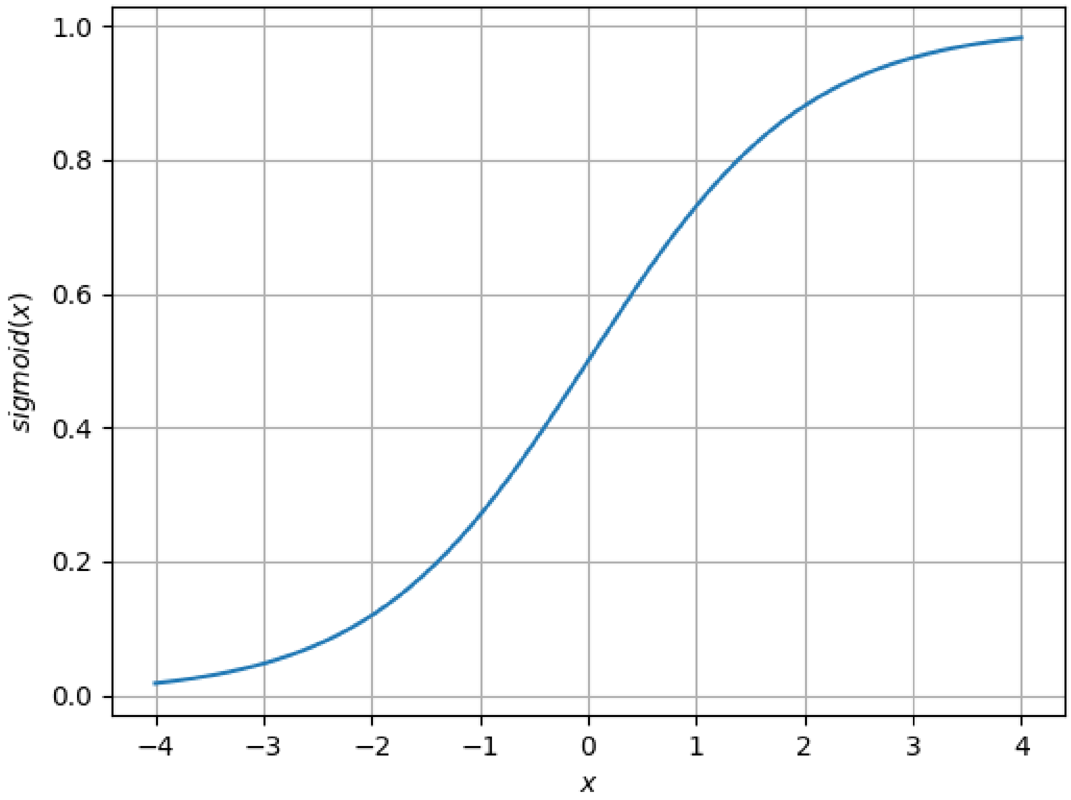

Figure 2.1: Sigmoid function graph

The graph of Sigmoid function is shown in figure 2.1

2.2.2 Mathematical Understanding

As seen in chapter 1 and discussed in section 2.2,

$$y'_i = \frac{1}{1 + e^{-\vec{W} \cdot \vec{X}_{(i)}}} \tag{2.2}$$

2.2.3 Finding W and b

We use a little more math here to make the computation easier. We define cost function in a slightly different way

$$h_i = (y'_i)^{y_i}(1 - y'_i)^{1-y_i} \tag{2.3}$$

If you observe keenly and try to understand, the h value gives the absolute difference between the calculated outcome and original outcome. The original outcome is either 0 or 1. We try to spread y' in such a way that the outcome of a particular instance will be close to 0 or 1 depending one the original outcome.

As we are going to use gradient descent, we will be calculating gradient in every round. Calculating gradient for h itself is a little heavy task for computation. We will take $\log(h)$ to make it easier.

So we will be defining the cost function as follows

$$J(W, b) = \frac{1}{m}\sum_{i=1}^{m} \log h_i \tag{2.4}$$

The gradient for cost function J

$$\nabla J(W, b) = \begin{bmatrix} \frac{\partial J}{\partial W} \\ \frac{\partial J}{\partial b} \end{bmatrix} = \begin{bmatrix} \frac{1}{m}\sum_{i=1}^{m} X^{(i)}(y_i - y'_i) \\ \frac{1}{m}\sum_{i=1}^{m}(y_i - y'_i) \end{bmatrix} \tag{2.5}$$

Now we update W and b accordingly

$$W_{new} = W - \alpha\frac{\partial J}{\partial W} \tag{2.6}$$

$$b_{new} = b - \alpha\frac{\partial J}{\partial b} \tag{2.7}$$

We use the obtained W and b for predicting outcome. Notice here that y' contains real number between 0 and 1. We will replace all the values in y' greater than 0.5 with 1 and others with 0.

2.3 Questions for Curiosity

3

Decision Trees

3.1 Introduction

Decision Trees are simple but effective models which can do both classification and regression. Unlike logistic regression, it can classify into more than two classes. They are easy to understand visually and can be used explicitly to represent decisions.

3.2 Model

We basically construct a binary tree, with every node containing a statement about a feature. The node gives branches based on the truth value of the statement. The child might be a node containing another statement or a leaf with a class label (in case of classification) or a leaf with a value (or range) in case of regression. Decision Tree is a very wide concept and contains many algorithms and types of implementation.

3.2.1 Mathematical Understanding

As discussed, the decision tree outputs discrete values. For regression problems, we use those discrete values as ranges and convert them to continuos values. So the key idea is to split the n-dimensional space into regions, which we can further decide upon.

In technical terms we follow a recursive greedy, top-down, partioning approach. We have a parent partion R_p which we decide to split further. Let $S_p(j,t)$ be a split in R_p such that

$$S_p(j,t) = (\{X|x_j < t, X \in R_p\}, \{X|x_j \geq t, X \in R_p\}) \tag{3.1}$$

Our aim here is to make sure that we achieve good classification with each split. For that we will do some math.

Define $L(R)$ as the loss on R. Here loss means the presence of instances belonging to other classes compared to the major class. There are many ways of defining this loss. Our aim is to loss is minimum at every split.We can maximize the error before and after split to achieve that. For that $L(R_p) - (L(R_1) + L(R_2))$ should be maximum.

3.3 Questions For Curiosity

4

Naive Bayes

4.1 Introduction

Naive Bayes, as the name suggests, is an application of Bayes' theorem with strong assumptions. We assume a strong independence between the features. We use a very different approach in Naive Bayes for building the model.

4.2 Model

Consider there are K classes the model has to classify the input into. We need $\Pr(C_k|X)$ for every class C_k.

$$\Pr(C_k|X) = \frac{\Pr(C_k)\Pr(X|C_k)}{\Pr(X)} \tag{4.1}$$

The numerator represents joint probability i.e. $\Pr(C_k, x_1, x_2, \cdots, x_n)$.

$$\Pr(x_1, x_2, \cdots, x_n, C_k) = \Pr(x_1|x_2, \cdots, x_n, C_k)\Pr(x_2|x_3, \cdots, x_n, C_k)\cdots\Pr(x_n|C_k)\Pr(C_k) \tag{4.2}$$

We assume strong independence between features. So

$$\Pr(x_1, x_2, \cdots, x_n, C_k) = \Pr(x_1|C_k)\Pr(x_2|C_k)\cdots\Pr(x_n|C_k)\Pr(C_k) \tag{4.3}$$

$$\Pr(C_k|X) \propto \prod_{i=1}^{n} \Pr(x_i|C_k) \tag{4.4}$$

Now from this

$$y' = \operatorname*{argmax}_{k\in\{1,2,\cdots,n\}} \Pr(C_k)\prod_{i=1}^{n} \Pr(x_i|C_k) \tag{4.5}$$

5

k Nearest Neighbours

5.1 Introduction

The k Nearest Neighbours (k-nn) evaluates the outcome by considering k (user given) neighbours. It can be used for classification and regression. Fot classification, it takes the votes of neighbours and for regression, it outputs the average of the neighbours. As we take distance as a measure for considering neighbours, normalising the features will improve accuracy.

5.2 Model

The training just involves storing training dataset. Whenever a test dataset is given, the necessary calculations are made and outcome is computed.

5.2.1 Mathematics

The distance is calculated using

$$d_l = \sqrt{\sum_{i=1}^{n} (x_i - x_{i(l)})^2} \tag{5.1}$$

where x_i are the scaled feature values. The distances are calculated and arranged in ascending order. The most repeated class in the first k distance is given as the outcome.

For regression, the average of first k neighbours will be given as outcome.

5.3 Questions For Curiosity

6

k Means Clustering

6.1 Introduction

k means clustering is used to partion m observations into k clusters. It is an unsupervised learning algorithm.

6.2 Model

The model works on te aim to divide the instances (instance is called observation further, as it makes more sense) into k classes and minimise the within-cluster variance.

6.2.1 Mathematical Understanding

We have

$$X = \begin{bmatrix} X^{(1)} & X^{(2)} & \cdots & X^{(m)} \end{bmatrix}$$

We generate k means arbitrarily.

$$\mu_1^{(1)} \ \mu_2^{(1)} \ \mu_3^{(1)} \ \cdots \ \mu_k^{(1)}$$

μ is a vetor of n dimensity. We assign each observation a cluster based on the nearest mean. After the assignment, we go through a series of iterations as following.

Assignment step:

$$S_i^{(t)} = \left\{X^{(p)} : \left\|X^{(p)} - \mu_i^{(t)}\right\|^2 \leq \left\|x_p - \mu_j^{(t)}\right\|^2 \forall j, 1 \leq j \leq k\right\} \tag{6.1}$$

We recalculate the mean of the clustered observations.

Update step:

$$m_i^{(t+1)} = \frac{1}{|S_i^{(t)}|} \sum_{X^{(j)} \in S_i^{(t)}} X^{(j)} \tag{6.2}$$

Since this is a converging algorithm, it will halt when the means no longer change. We will use various other versions of it to deal with this problem.

6.3 Questions For Curiosity

Acknowledgments