Linear Regression

Madiba Hudson-Quansah

Contents

Chapter 1

Linear Regression

1.1 Introduction

Definition 1.1.1: Linear Regression

A model that assumes a linear relationship between the input variables (**X**) and the single output variable (**Y**). This model makes predictions by computing a weighted sum of the input variables, plus a constant bias / intercept term (**b**). Represented as:

$$\mathbf{y} = b + w_1 x_1 + w_2 x_2 + \ldots + w_n x_n$$

Or

$$\mathbf{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_n x_n$$

Where:

- $\mathbf{y}$ is the predicted value.
- n is the number of features.
- x_i is the i-th feature value.
- θ_j is the j-th model parameter including the bias term θ_0 and feature weights $\theta_1, \theta_2, \ldots, \theta_n$

In vectorized form, the linear regression model can be represented as:

$$\mathbf{y} = h_\theta(\mathbf{x}) = \boldsymbol{\theta} \cdot \mathbf{x}$$

Where:

- $h_{\boldsymbol{\theta}}(\mathbf{x})$ i the hypothesis function for some parameters $\boldsymbol{\theta}$.
- $\boldsymbol{\theta}$ is the model's parameter vector, containing the bias term θ_0 and the feature weights $\theta_1, \theta_2, \ldots, \theta_n$.
- $\mathbf{x}$ is the input feature vector, containing x_0 to x_n, with x_0 always equal to 1.
- $\boldsymbol{\theta} \cdot \mathbf{x}$ is the dot product of the vectors $\boldsymbol{\theta}$ and $\mathbf{x}$, which is of course equal to $\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_n x_n$.

Note:-

In machine learning vectors are often represented as column vectors, which is why $\mathbf{x}$ is a column vector, and the dot product $\boldsymbol{\theta} \cdot \mathbf{x}$, simplifies to $\boldsymbol{\theta}^T \mathbf{x}$, where $\boldsymbol{\theta}^T$ is the transpose of $\boldsymbol{\theta}$.

1.2 Loss / Cost Function

In training a linear regression model, we need to find the value of $\boldsymbol{\theta}$ such that the model makes the best predictions on the training data. To do this, we need a way to measure how well (or poorly) the model is performing on the training data. We can do this by defining a **loss function** that measures the difference between the predicted value and the actual value. The goal is to minimize this difference.

For linear regression, the most common loss function is the **Mean Squared Error (MSE)**:

$$\text{MSE}(\mathbf{X}, h_{\boldsymbol{\theta}}) = \frac{1}{m} \sum_{i=1}^{m} \left(\boldsymbol{\theta}^T \mathbf{x}^i - \mathbf{y}^i \right)^2$$

Where m is the number of instances in the dataset.

1.2.1 The Normal Equation

One way to find the value of $\boldsymbol{\theta}$ that minimizes the cost function is to use the **Normal Equation**. This is a closed-form solution that gives the result directly as is derived from the derivation of the cost function. Given by:

$$\hat{\boldsymbol{\theta}} = \left(\mathbf{X}^T \mathbf{X} \right)^{-1} \mathbf{X}^T \mathbf{y}$$

Where:

- $\hat{\boldsymbol{\theta}}$ is the value of $\boldsymbol{\theta}$ that minimizes the cost function.
- $\mathbf{y}$ is the vector of target values containing $\mathbf{y}^{(1)}$ to $\mathbf{y}^{(m)}$.

This method is computationally expensive when the number of features is large, as the complexity of inverting the matrix is $O(n^3)$. However, it is linear with regard to the number of instances in the training set, so it handles large training sets efficiently. Another less computationally expensive method is the **Gradient Descent** algorithm.

1.2.2 Derivation of the Normal Equation

The loss function:

$$\text{MSE}(X, h_{\boldsymbol{\theta}}) = \frac{1}{m} \sum_{i=1}^{m} \left(\mathbf{x}^i \boldsymbol{\theta} - y^i \right)^2$$

Where

- $\mathbf{x}^i$ is a vector of the i-th instance's feature values from the input matrix X, where each instance is represented as row of the matrix. I.e. $\mathbf{x}^i$ is of the form:

  $$\mathbf{x}^i = \begin{bmatrix} 1 \\ x_1^i \\ x_2^i \\ \vdots \\ x_n^i \end{bmatrix}$$

  Where n is the number of features.

- $\boldsymbol{\theta}$ is a vector of the model's parameters / weights for each input feature including the bias / intercept θ_0. I.e. $\boldsymbol{\theta}$ is of the form:

  $$\boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}$$

  Where n is the number of features. This results in the multiplication $\mathbf{x}^i \theta$ being equivalent to the dot product $\mathbf{x}^i \cdot \theta$ and therefore a scalar value.

- y^i is the actual target value of the i-th instance and is a scalar value.
- m is the number of instances in the dataset.

Can be expressed in matrix-vector form as:

$$\text{MSE}(X, h_{\boldsymbol{\theta}}) = \frac{1}{m} \left(X\boldsymbol{\theta} - \mathbf{y} \right)^T \left(X\boldsymbol{\theta} - \mathbf{y} \right)$$

Where:

- X is a input matrix containing all the input features of all instances in the dataset. X is of the form:

$$X = \begin{bmatrix} \left(\mathbf{x}^1\right)^T \\ \left(\mathbf{x}^2\right)^T \\ \vdots \\ \left(\mathbf{x}^m\right)^T \end{bmatrix}$$

  With each vector $\mathbf{x}^i$ being an instance's input features previously defined as column vectors, hence the transposition.

- $\mathbf{y}$ is a vector of the target values of all instances in the dataset. $\mathbf{y}$ is of the form:

$$\mathbf{y} = \begin{bmatrix} y^1 \\ y^2 \\ \vdots \\ y^m \end{bmatrix}$$

- $\boldsymbol{\theta}$ is still the model's parameter vector as previously defined.
- m is still the number of instances in the dataset.

This leads us to finding the partial derivative of this expression with respect to $\boldsymbol{\theta}$ and minimizing it, equating it to zero, given below where $L(\boldsymbol{\theta})$ is the loss function:

$$\begin{aligned} L(\boldsymbol{\theta}) &= \frac{1}{m}\left(X\boldsymbol{\theta} - \mathbf{y}\right)^T\left(X\boldsymbol{\theta} - \mathbf{y}\right) \\ L(\boldsymbol{\theta}) &= \frac{1}{m}\left(\left(X\boldsymbol{\theta}\right)^T - \mathbf{y}^T\right)\left(X\boldsymbol{\theta} - \mathbf{y}\right) \\ &= \frac{1}{m}\left(\boldsymbol{\theta}^T X^T - \mathbf{y}^T\right)\left(X\boldsymbol{\theta} - \mathbf{y}\right) \\ &= \frac{1}{m}\left(\boldsymbol{\theta}^T X^T X\boldsymbol{\theta} - \boldsymbol{\theta}^T X^T\mathbf{y} - \mathbf{y}^T X\boldsymbol{\theta} + \mathbf{y}^T\mathbf{y}\right) \\ &\text{Let } \mathbf{a} = X^T\mathbf{y} \text{ and } S = X^T X \text{ then} \\ &= \frac{1}{m}\left(\boldsymbol{\theta}^T S\boldsymbol{\theta} - \boldsymbol{\theta}^T\mathbf{a} - \mathbf{a}\boldsymbol{\theta} + \mathbf{y}^T\mathbf{y}\right) \end{aligned}$$

We then take the partial derivative of the resulting expression with respect to $\boldsymbol{\theta}$:

$$\begin{aligned} \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}) &= \nabla_{\boldsymbol{\theta}}\left(\frac{1}{m}\boldsymbol{\theta}^T S\boldsymbol{\theta} - \boldsymbol{\theta}^T\mathbf{a} - \mathbf{a}\boldsymbol{\theta} + \mathbf{y}^T\mathbf{y}\right) \\ &= \frac{1}{m}\left(2S\boldsymbol{\theta} - \mathbf{a} - \mathbf{a} + 0\right) \\ &= \frac{1}{m}\left(2S\boldsymbol{\theta} - 2\mathbf{a}\right) \end{aligned}$$

Then equate the partial derivative to zero:

$$\begin{aligned} \nabla_{\theta} L(\boldsymbol{\theta}) &= \frac{1}{m}\left(2S\boldsymbol{\theta} - 2\mathbf{a}\right) \\ 0 &= 2S\boldsymbol{\theta} - 2\mathbf{a} \\ 0 &= S\boldsymbol{\theta} - \mathbf{a} \\ S\boldsymbol{\theta} &= \mathbf{a} \\ \hat{\boldsymbol{\theta}} &= S^{-1}\mathbf{a} \\ \hat{\boldsymbol{\theta}} &= \left(X^T X\right)^{-1} X^T\mathbf{y} \end{aligned}$$

1.3 Gradient Descent

Definition 1.3.1: Gradient Descent

A generic optimization algorithm capable of finding optimal solutions to a wide range of problems. The general idea of Gradient Descent is to tweak parameters iteratively in order to minimize a cost function.

1.3.1 Batch Gradient Descent

To implement Gradient Descent, we need to compute the gradient of the cost function with regard to each model parameter θ_j or parameter vector $\boldsymbol{\theta}$. In other words, we need to calculate how much the cost function will change if we change θ_j just a little bit. This is called a **partial derivative**. This is given by:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta})$$

Where:

- α is the learning rate.
- $\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta})$ is the gradient of the cost function with respect to the current $\boldsymbol{\theta}$ and is equivalent to the partial derivative of the cost function with respect to θ_j.

The partial derivative of the cost function with respect to θ_j is found the same way as the normal equation but we do not equate it to zero. Given by:

$$\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}) = \frac{2}{m} \left(X^T X \boldsymbol{\theta} - X^T \mathbf{y} \right)$$

1.3.2 Stochastic Gradient Descent

Definition 1.3.2: Stochastic

Randomly determined; having a random probability distribution or pattern that may be analyzed statistically but may not be predicted precisely.

Batch gradient descent requires the whole training set to compute the gradients at each step, which makes it slow when the dataset is large. The Stochastic Gradient Descent algorithm is a variation of the gradient descent algorithm that updates the weights based on a randomly chosen training instance. So instead of the equation above, we have:

$$\boldsymbol{\theta} \leftarrow \theta_i - \alpha \nabla_{\theta_i} L(\theta_i)$$

This makes the algorithm much faster because it has little data to manipulate at each step. However, due to its stochastic nature, it is less stable than batch gradient descent often converging to a value close to the minimum but not the minimum itself.

Chapter 2

Locally Weighted Regression (LOWESS)

2.1 Introduction

Definition 2.1.1: Locally Weighted Regression

A non-parametric regression algorithm that makes predictions by fitting several local linear models to a dataset. It generally follows the same representation as linear regression, but the weight of each instance is determined by a kernel / weight function that gives more weight to instances closer to the instance being predicted, usually using the Gaussian kernel, given by:

$$w = \exp\left(-\frac{\left(x^{(i)} - x\right)^2}{2\tau^2}\right)$$

Instead of fitting a single model to the entire dataset, LOWESS fits a model to a subset of the data, where the size of the subset is determined by the bandwidth parameter τ. The weight function w gives more weight to instances closer to a specific point of interest usually the point being predicted, with the weight decreasing as the distance from the point of interest increases.

The bandwidth parameters τ, determines how many points are considered in the local model, with smaller values making the model more sensitive to local fluctuations in the data, while larger values make the model produce a smoother fit.

2.2 Kernel Functions

Definition 2.2.1: Kernel Function

Determines how weights are assigned to the neighbouring points.

2.2.1 Gaussian Kernel

Provides weights based on the normal distribution, giving more weight to points closer to the point of interest. Given by:

$$w = \exp\left(-\frac{\left(x^{(i)} - x\right)^2}{2\tau^2}\right)$$

Where:

- w is the weight assigned to the i-th instance.
- x^i is the i-th instance's feature value.
- x is the feature value of the point of interest.
- τ is the bandwidth parameter which determines the size of the subset of the data that the model fits to.

2.2.2 Tri-cube Kernel

Provides weights based on the tri-cube function, giving more weight to points closer to the point of interest. Given by:

$$w = \left(1 - \left(\frac{\left|x^{(i)} - x\right|}{\tau}\right)^3\right)^3$$

Where:

- w is the weight assigned to the i-th instance.
- x^i is the i-th instance's feature value.
- x is the feature value of the point of interest.
- τ is the bandwidth parameter which determines the size of the subset of the data that the model fits to.

2.3 Parametric vs Non-parametric Models

Definition 2.3.1: Parametric Model

This model assumes a specific form for the underlying data distribution and have a fixed number of parameters, with these parameters being learned from the training data.

Definition 2.3.2: Non-parametric Model

This model does not assume a specific form for the underlying data distribution and can adapt their shape based on the data, with the number of parameters increasing with the size of the training data.

Chapter 3

Logistic Regression

3.1 Introduction

Definition 3.1.1: Logistic Regression

A classification algorithm based on regression that estimates the probability that an instance belongs to a particular class. Given by:

$$\hat{p} = h_{\boldsymbol{\theta}}(\boldsymbol{\theta} \cdot \mathbf{x}) = \sigma\left(\boldsymbol{\theta}^T \times \mathbf{x}\right)$$

Where σ is the **sigmoid function** defined by:

$$\sigma(z) = \frac{1}{1 + \exp^{-z}}$$

This function outputs a value between 0 and 1, which can be interpreted as a probability.

Once a Logistic regression model has estimated the probability $\hat{p}$ that an instance belongs to the positive class, it can make its prediction $\hat{y}$, based on a specified threshold value, usually 0.5. Given by:

$$\hat{y} = \begin{cases} 0 & \text{if } \hat{p} < 0.5 \\ 1 & \text{if } \hat{p} \geqslant 0.5 \end{cases}$$

3.2 Training and Loss Function

In training a Logistic regression model, we need to find a parameter vector $\boldsymbol{\theta}$ such that the model estimates high probabilities for positive instances $y = 1$ and low probabilities for negative instances $y = 0$. The loss function for a single instance can then be given by:

$$L(\boldsymbol{\theta}) = \begin{cases} -\log\left(\hat{p}\right) & \text{if } y = 1 \\ -\log\left(1 - \hat{p}\right) & \text{if } y = 0 \end{cases}$$

This loss function works as the value of $-\log t$ grows large when t approaches 0, i.e. the model will be penalized if it estimates a probability close to 0 for a positive instance, and similarly for a probability close to 1 for a negative instance. On the other hand $-\log t$ is close to 0 when t is close to 1, so the loss will be small if the model estimates a probability close to 0 for a negative instance or close to 1 for a positive instance.

Applying this loss function over the entire training set, we want the average loss over all the training instances so we must find the sum of all the losses and divide by the number of instances. Given by:

$$L(\boldsymbol{\theta}) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^i \log\left(\hat{p}^i\right) + \left(1 - y^i\right)\log\left(1 - \hat{p}^i\right)\right]$$

This in turn was derived from the log likelihood function, which is the product of the probabilities of the instances in the training set being classified correctly, the use of product due to the AND operation of the probabilities. Given by:

$$\log L(\boldsymbol{\theta}) = \log \prod_{n=1}^{m} \left(\frac{1}{\sqrt{2\pi\boldsymbol{\theta}}} \right) \exp\left(-\frac{\left(y^i - x^i\boldsymbol{\theta}\right)^2}{2\theta^2} \right)$$

This was simplified to a summation due to $\log(a \times b) = \log a + \log b$ giving us:

$$\log L(\boldsymbol{\theta}) = \sum_{i=1}^{m} \left(\log \frac{1}{\sqrt{2\pi\boldsymbol{\theta}}} + \log \exp\left(\frac{\left(y^i + x^i\boldsymbol{\theta}\right)}{2\boldsymbol{\theta}^2} \right) \right)$$

This was further simplified using the properties of the logarithm function to give us:

$$\log L(\boldsymbol{\theta}) = \sum_{i=1}^{m} \left(-\frac{1}{2} \log 2\pi\boldsymbol{\theta} - \frac{\left(y^i + x^i\boldsymbol{\theta}\right)^2}{2\boldsymbol{\theta}^2} \right)$$

Using Bernoulli's distribution and applying the sigmoid function to the linear regression model, we can posit that the hypothesis function's output is the probability that the instance belongs to the positive class and now exists in the range $[0, 1]$. Given this we can simplify the likelihood of a positive instance and the inverse probability of a negative instance using the output of the hypothesis function, giving us:

$$\log L(\boldsymbol{\theta}) = \sum_{i=1}^{m} \left(y^i \log\left(\hat{p}^i\right) + \left(1 - y^i\right) \log\left(1 - \hat{p}^i\right) \right)$$

By using the log loss we are implicitly making the assumption that the instances follow a Gaussian distribution around the mean of their respective classes. This is a common assumption in logistic regression and is the reason why the log loss is used as the loss function.

There exists no closed-form equation to compute the value of $\boldsymbol{\theta}$ that minimizes the cost function, but due to the convex nature of the cost function, we are guaranteed to find the global minimum using the Gradient Descent or any other optimization algorithm. The partial derivative of the cost function with respect to θ_j is given by:

$$\nabla_{\theta_j} L(\boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^{m} \left(\sigma\left(\boldsymbol{\theta}^T \mathbf{x}^{(i)}\right) - y^{(i)} \right) x_j^{(i)}$$