# PAL Machine Learning Workshop Week 3: Linear Regression

05/12/2022 18:00 Future Technologies Lab

# Contents

# Supervised Learning Problem

- Training data comes in pairs of inputs and targets.

$$D_{train} = \{(x^{(i)}, y^{(i)}) \,|\, i = 1,2,3,\ldots,n\}, \text{ where } x^{(i)} \in X, y^{(i)} \in Y$$

- The predictive model tries to model the relationship between inputs and targets.

$$f : X \to Y$$

- The goal is to accurately predict the label of new, unseen data based on the patterns learned from the labeled training data.

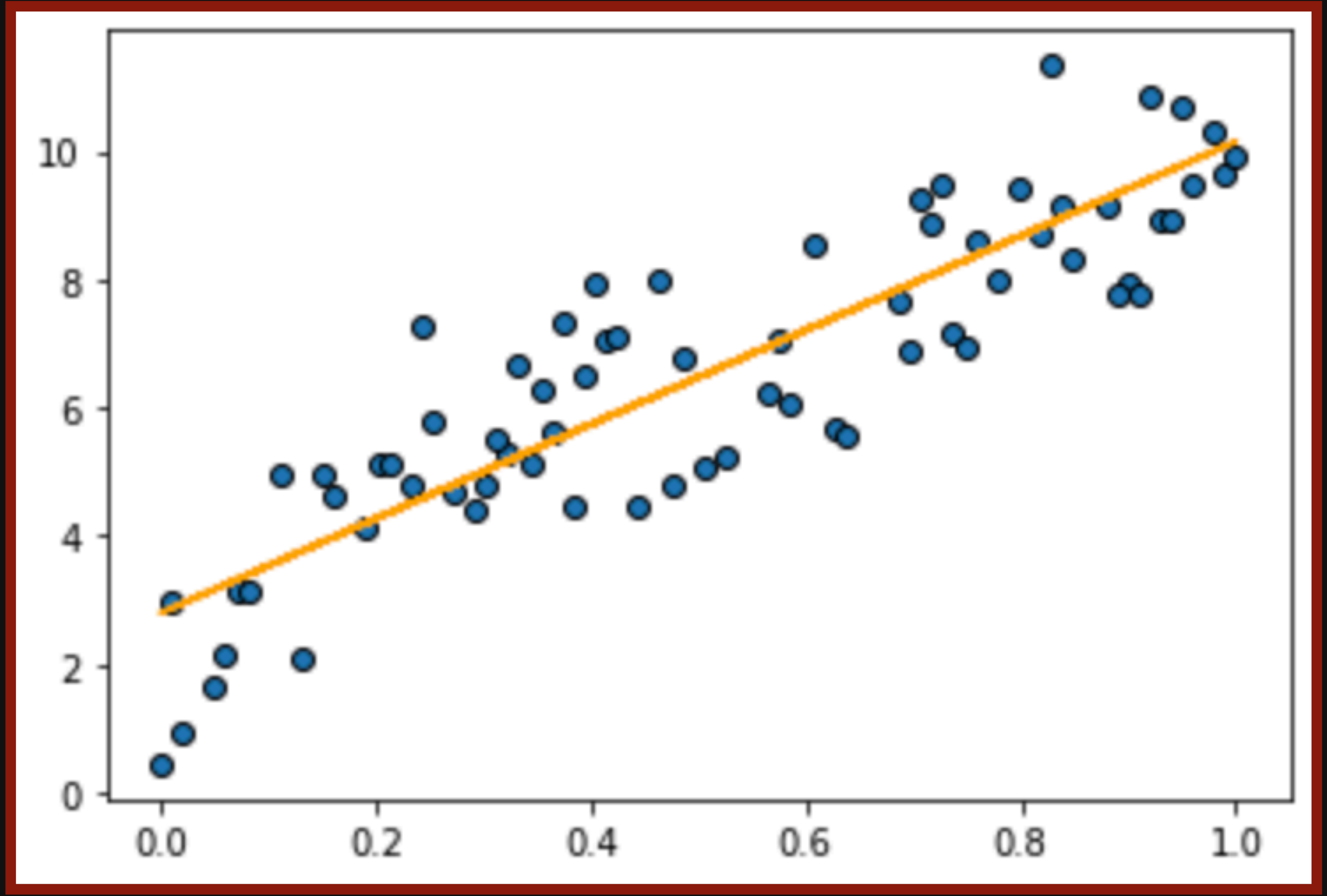
- Supervised learning is used in many real-world applications, such as image classification, natural language processing, and speech recognition.

# Linear Regression

- We could assume that $y$ is some linear function of $x$. In other words, for some unknown $\theta_1, \theta_2 \in \mathbb{R}$, we have:

$$f(x) = \theta_0 + \theta_1 x$$

- We will refer to $\theta_1, \theta_2$ as the *parameters,* where $x$ is the *independent* variable and $y$ is the *dependent* variable.

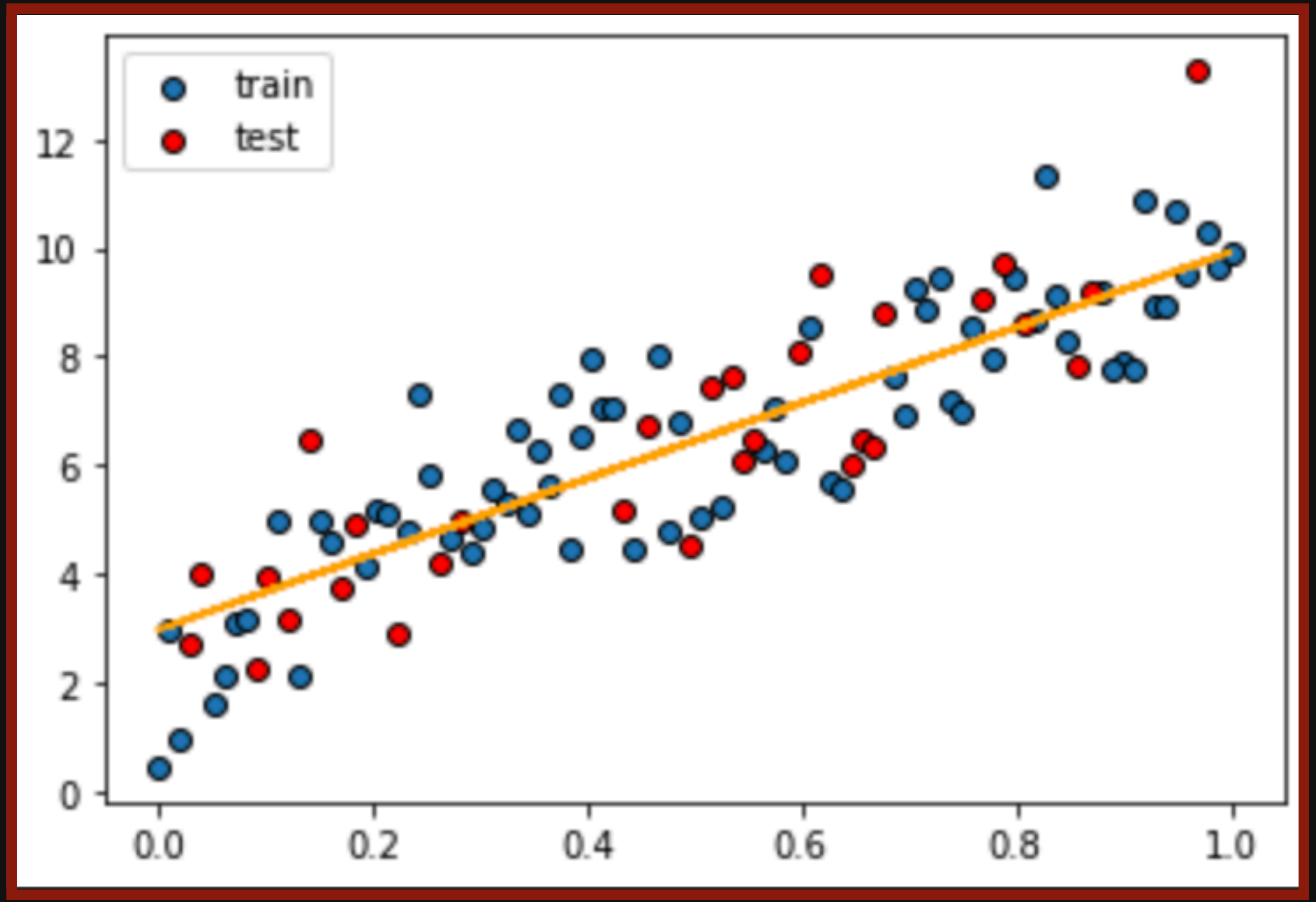


# Linear Regression

- Our aim is to find the best set of parameters $\theta^* = \{\theta_1^*, \theta_2^*\}$, i.e. the one that gives us the best result on our training data.

- Once we've learned the predictive model $f(x)$, we want to use it to make predictions $y'$ for the new data $x'$ that we haven't seen before:

$$y' = f(x') = x'\theta^*$$



# Multiple Linear Regression

- We can use more than two independent variables.

$$f(x) = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_d x_d$$
$$= \sum_{i=0}^{d} \theta_d x_d$$

- The term $x_0$ is always equal to $1$. This is a convention used to represent the intercept or the constant term in the model equation.

# Linear Regression

## Matrix Notation

- We can represent our data, parameters and target values using matrix notation:

$$X = \begin{array}{cc} \begin{bmatrix} x_{10} & \cdots & x_{1d} \\ \vdots & \ddots & \\ x_{n0} & & x_{nd} \end{bmatrix} & n \\ d+1 & \end{array}$$

- $x$ is a *design matrix*, where $d$ is the number of features and $n$ is the number of examples
- We add a column of ones to capture an intercept

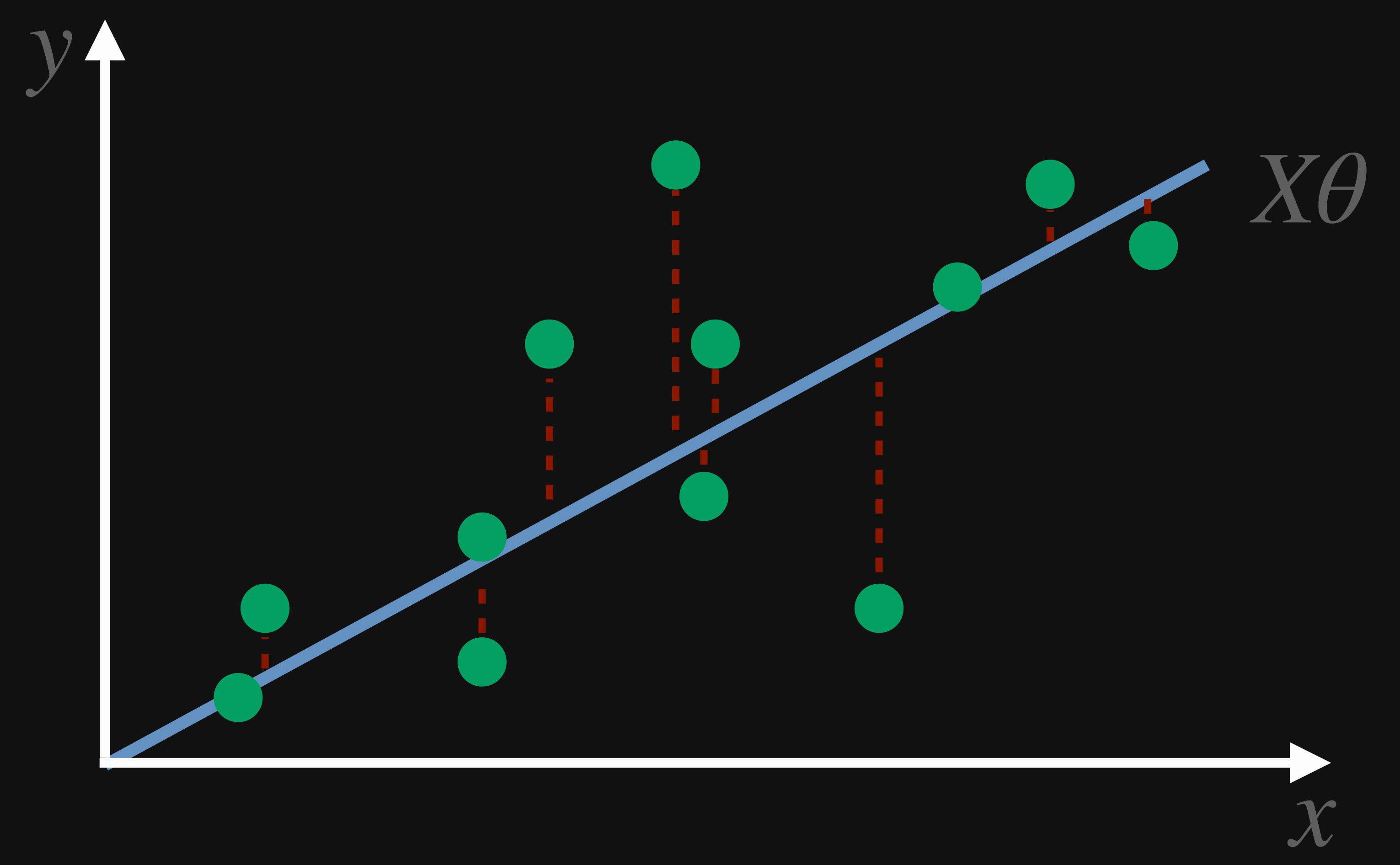
$$= \begin{array}{cc} \begin{bmatrix} 1 & \cdots & x_{1d} \\ \vdots & \ddots & \\ 1 & & x_{nd} \end{bmatrix} & n \\ d+1 & \end{array} \qquad \theta = \begin{bmatrix} \theta_0 \\ \vdots \\ \theta_d \end{bmatrix} d+1 \qquad \hat{y} = \begin{bmatrix} \hat{y}_0 \\ \vdots \\ \hat{y}_n \end{bmatrix} n \qquad \hat{y} = X\theta$$

# Loss Functions

- Loss function measures deviation of the model's prediction from the ground truth.
- Allows to evaluate the fit of a machine learning model.
- MSE is defined as the average sum of the squared differences between the prediction and the ground truth.

$$RSS = \frac{1}{2}\sum_{i=1}^{n}(y - X\theta)^2$$

$$MSE = \frac{1}{2n}\sum_{i=1}^{n}(y - X\theta)^2$$



# Optimization using Normal Equations

- The gradient of RSS can be defined as follows:

$$\begin{aligned}
\nabla_\theta J(\theta) &= \frac{1}{2}\nabla_\theta (y - X\theta)^T(y - X\theta) \\
&= \frac{1}{2}\nabla_\theta \left((X\theta)^T(X\theta) - y(X\theta)^T - (X\theta)^T y + y^T y\right) \\
&= \frac{1}{2}\nabla_\theta \left(\theta^T X^T X\theta - 2(X\theta)^T y + y^T y\right) \qquad a^T b = b^T a \\
&= \frac{1}{2}\left(2(X^T X)\theta - 2X^T y\right) \qquad \nabla_x x^T A x = 2Ax \text{ for a symmetric matrix } A, \quad \nabla_x b^T x = b \\
&= (X^T X)\theta - X^T y
\end{aligned}$$

# Optimization using Normal Equations

- The derivative becomes $0$ at the minimum of a function.
- Since $J(\theta)$ is a quadratic function it will only have one minimum.
- If we solve the above expression for $\theta$ we will get an expression for the minimum of our MSE loss function:

$$(X^T X)\theta - X^T y = 0$$

$$(X^T X)\theta = X^T y$$

- Hence the value $\theta*$ that minimises the objective is given by:

$$\theta* = (X^T X)^{-1} X^T y = X^\dagger y$$

# Implementing Linear Regression

- Now let's try to write a linear regression by ourselves in Python!
- You can download today's notebook from Canvas or open it using this link.

# Extension: Gradient Descent Optimization

- Gradient Descent is an optimisation technique that finds a minimum of a function by changing its parameters in proportion to the negative of the gradient of the function at the current value.

- Don't worry too much about how it works. Just try to get an idea of what the optimization process actually looks like

- If you want more detail, we start by randomly initializing the weights, and then update them by taking the gradient of the loss function w.r.t each weight, multiplying it by a small number called the learning rate, and subtracting that value from our current weight value. This leads to a slow movement to the minimum of the function.