Machine Learning

|Chapter 04 – Training Models

Prof. Dr. Mostafa Elhosseini

https://youtube.com/@ProfElhosseiniTechVerse

Agenda

- Linear regression
- Def.
- Loss Function & Performance Measure
- Closed-form equation : Normal equation
- Computational complexity

Intro

- you may have been surprised by how much you can get done without knowing anything about what's under the hood
  - you optimized a regression system,
  - you improved a digit image classifier, and
  - you even built a spam classifier from scratch
- However, having a good understanding of how things work can help you quickly
  - Right training algorithm to use
  - good set of hyperparameters for your task
  - Understanding what's under the hood will also help you debug issues and perform error analysis more efficiently

Linear regression model

- if the experiment values obtained in an experiment and thus involve an experimental error, and if the nature of the experiment suggests a linear relation, we better fit a straight line through the points.
- Linear Regression is one of the most common, some 200 years old and most easily understandable in statistics and machine learning
  - it comes under predictive modelling.
  - Predictive modelling is a kind of modelling here the possible output(Y) for the given input(X) is predicted based on the previous data or values.
- A widely used principle for fitting straight lines is the method of least squares by Gauss and Legendre

Linear regression model

- The straight line

$$y = mx + b$$

should be fitted through the given points $(x_i, y_i), \ldots, (x_n, y_n)$ so that the sum of the squares of the distances of those points from the straight line is minimum, where the distance is measured in the vertical direction (the y-direction)

Linear regression model

$$\boldsymbol{life\ satisfaction = \theta_0 + \theta_1 GDP\_per\_capita}$$

- This model is just a linear function of the input feature GDP_per_capita
- $\boldsymbol{\theta_0, \theta_1}$ is a set of parameters for the function $h(x)$ – we try to find the optimal settings of these parameters
- Model function: Our model ("hypothesis" or "estimator" or "predictor") will be a straight line "fit" to the training set".

Linear regression model

- Linear model makes a prediction by simply computing a weighted sum of the input features, plus a constant called the *bias term* (also called the *intercept term*)

$$\widehat{Y} = \boldsymbol{\theta_0} + \boldsymbol{\theta_1 x_1} + \boldsymbol{\theta_2 x_2} + \cdots + \boldsymbol{\theta_n x_n}$$

  - $\widehat{\boldsymbol{Y}}$ is the predicted value
  - $\boldsymbol{n}$ is the number of features
  - $\boldsymbol{x_i}$ is the ith feature value
  - $\boldsymbol{\theta_j}$ is the jth model parameter (including the bias term $\boldsymbol{\theta_0}$ and the feature weights $\boldsymbol{\theta_1}, \boldsymbol{\theta_2}, \cdots, \boldsymbol{\theta_n}$)

Linear regression model

$$\hat{Y} = \boldsymbol{\theta_0} + \boldsymbol{\theta_1 x_1} + \boldsymbol{\theta_2 x_2} + \cdots + \boldsymbol{\theta_n x_n}$$

- In vectored form:

$$\hat{\boldsymbol{Y}} = h_\theta(X) = \theta^T . X$$

$$\hat{\boldsymbol{Y}} = [\boldsymbol{\theta_0} \quad \boldsymbol{\theta_1} \ldots \quad \boldsymbol{\theta_n}] \begin{bmatrix} 1 \\ \boldsymbol{x_1} \\ . \\ . \\ . \\ \boldsymbol{x_n} \end{bmatrix}$$

Linear regression model

- Training a model means setting its parameters so that the model best fits the training set.
  - how do we train it?
  - we first need a measure of how well (or poorly) the model fits the training data
  - Cost Function: Sum of squared errors that we will minimize with respect to the model parameters.
  - The distance between the points and line are taken and each of them is squared to get rid of negative values and then the values are summed which gives the error which needs to be minimized – how to minimize error?
  - Therefore, to train a Linear Regression model, you need to find the value of θ that minimizes the cost function

Loss Function & Performance Measure

- **Loss function** (also known as a **cost function** or **error function**) is a mathematical function that measures the **difference** between the predicted values generated by a model and the actual values from the data
- The goal during training is to **minimize this loss function**, guiding the model to improve its predictions.
- Typically designed to be **smooth** and **differentiable** to facilitate optimization using **gradient-based methods**.
- Different loss functions are used depending on the type of problem (e.g., Mean Squared Error for regression, Cross-Entropy Loss (Log Loss) for classification).

Loss Function & Performance Measure

- A **performance measure** (or **performance metric**) is a criterion used to evaluate how well a machine learning model performs on a given task.
- **Unlike the loss function** used during training, performance measures assess the model's effectiveness on **unseen or test data**, providing insights into its generalization and practical utility
- Performance measures should **reflect the real-world goals** and requirements of the task. They are chosen based on **what is most important for the application**, such as accuracy, precision, or recall.

Loss Function & Performance Measure

Aspect	Loss Function	Performance Measure
Purpose	Guides model training by minimizing errors during learning.	Evaluates the model's effectiveness on unseen data.
Used During	Training phase	Evaluation phase
Mathematical Properties	Typically differentiable and smooth to facilitate optimization.	May not be differentiable; focuses on practical performance.
Examples	Mean Squared Error (MSE), Cross-Entropy Loss (Log Loss)	Accuracy, Precision, Recall, F1-Score, AUC-ROC
Task-Specific Variants	Different for tasks like regression (MSE) vs classification (Log Loss).	Different for tasks like classification (Accuracy) vs regression (MAE).
Influences	Directly impacts the adjustment of model parameters.	Guides model selection and hyperparameter tuning.
Incorporates Regularization	Often includes terms to prevent overfitting (e.g., L1, L2 regularization).	Focuses on end results without regularization terms.
Optimization Goal	Minimize the loss to improve the model's fit to training data.	Maximize or achieve a favorable metric value for best performance.
Ease of Use in Algorithms	Chosen for compatibility with gradient-based optimization techniques.	Selected based on relevance to the business or application objective.

Cost function

- In practice, it is simpler to minimize the Mean Square Error (MSE) than the RMSE, and it leads to the same result
- $MSE(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left(\theta^T . x^i - y^i\right)^2$
  - Because the value that minimizes a function also minimizes its square root

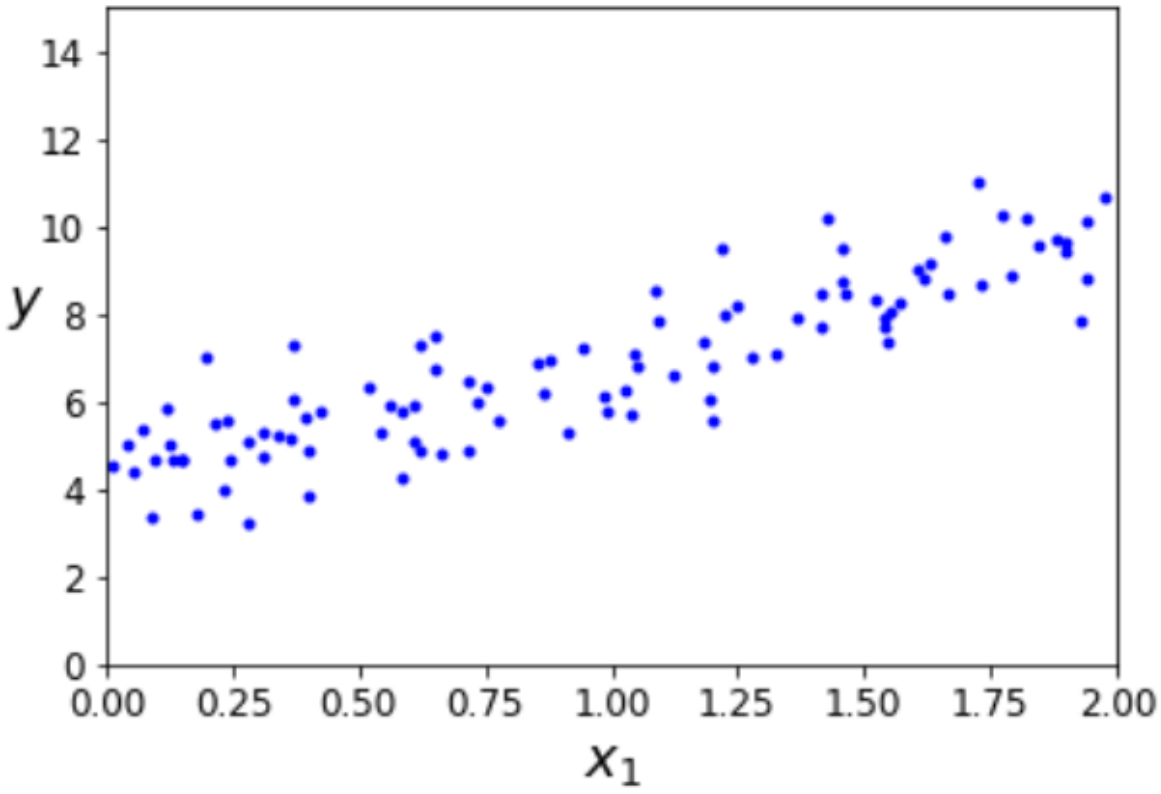
① Closed-form equation: Normal equation

② Iterative Optimization Approach
  - Gradient Descent
  - Batch
  - Stochastic
  - Minibatch

Normal equation

- Closed-Form Solution
- Analytical Solution - Direct Method
- Ordinary Least Squares (OLS)
- To find the value of θ that minimizes the cost function, there is a *closed-form solution* — in other words, a mathematical equation that gives the result directly
  - $\widehat{\boldsymbol{\theta}} = \left(\boldsymbol{X}^{\boldsymbol{T}}.\boldsymbol{X}\right)^{-1}.\boldsymbol{X}^{\boldsymbol{T}}.\boldsymbol{Y}$
  - $\hat{\theta}$ is the value of θ that minimizes the cost function
  - Y is the vector of target values containing $y^{(1)}$ to $y^{(m)}$

Linear-looking data




```
import numpy as np
X = 2 * np.random.rand(100, 1)
y = 4 + 3 * X + np.random.randn(100, 1)
```

Normal equation

```
from sklearn.preprocessing import add_dummy_feature

X_b = add_dummy_feature(X)  # add x0 = 1 to each instance
theta_best = np.linalg.inv(X_b.T @ X_b) @ X_b.T @ y
```

```
array([[4.21509616],
       [9.75532293]])
```

- The @ operator performs matrix multiplication
- Many other libraries, like TensorFlow, PyTorch, and JAX, support the @ operator as well. However, you cannot use @ on pure Python arrays (i.e., lists of lists)

Normal equation

```
from sklearn.preprocessing import add_dummy_feature

X_b = add_dummy_feature(X)  # add x0 = 1 to each instance
theta_best = np.linalg.inv(X_b.T @ X_b) @ X_b.T @ y
```

```
array([[4.21509616],
       [9.75532293]])
```

- $y = 4 + 3x_1 +$ Gaussian noise
- Noise made it impossible to recover the exact parameters of the original function.
- The smaller and noisier the dataset, the harder it gets

Normal equation

```
from sklearn.preprocessing import add_dummy_feature

X_b = add_dummy_feature(X)  # add x0 = 1 to each instance
theta_best = np.linalg.inv(X_b.T @ X_b) @ X_b.T @ y
```

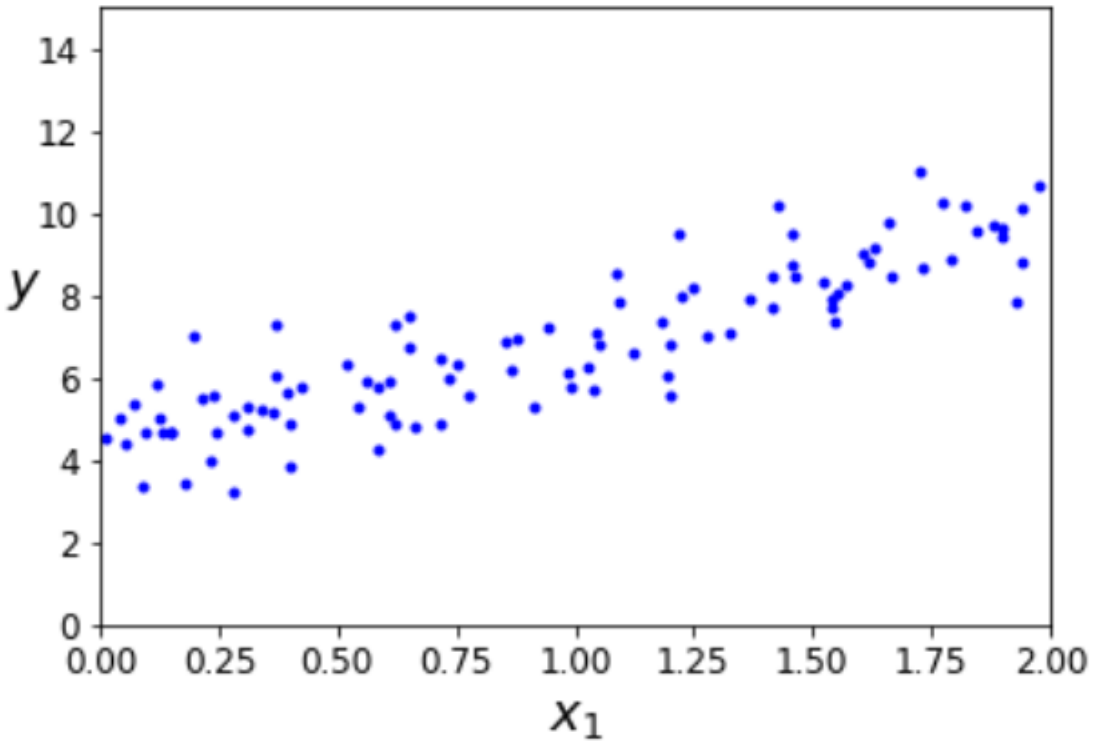
```
array([[4.21509616],
       [9.75532293]])
```

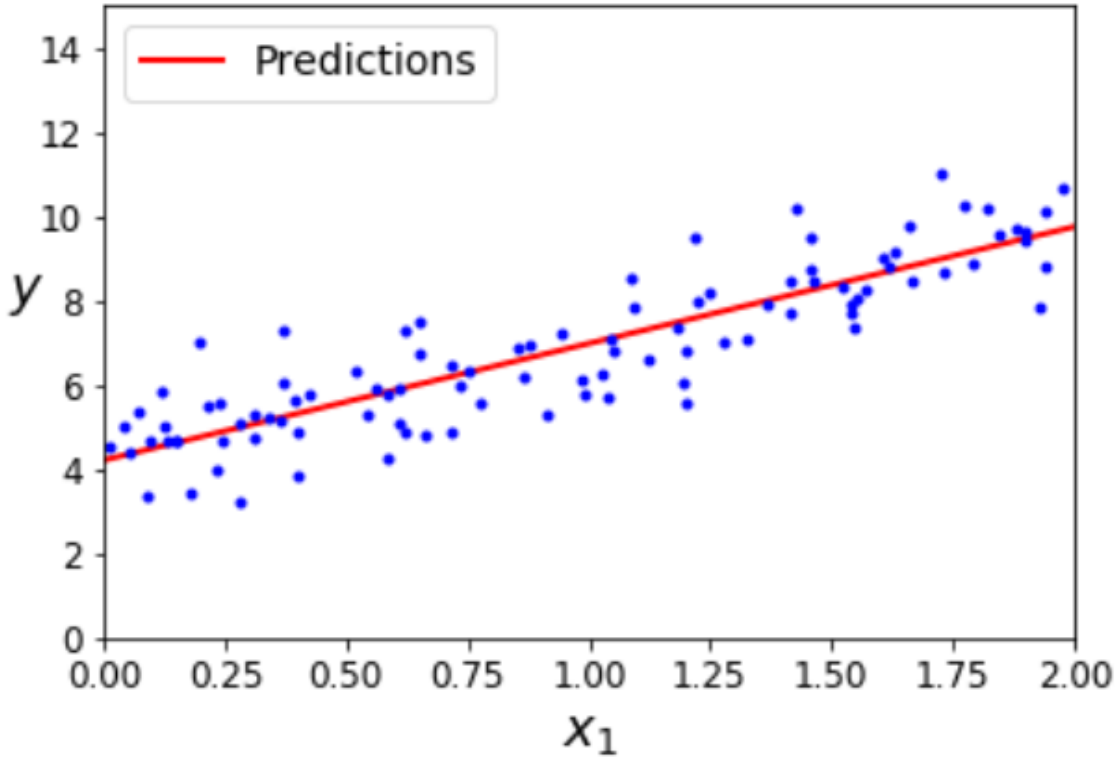
- Predictions

```
X_new = np.array([[0], [2]])
X_new_b = add_dummy_feature(X_new)  # add x0 = 1 to each instance
y_predict = X_new_b @ theta_best
y_predict
```

```
array([[4.21509616],
       [9.75532293]])
```

Normal equation





Equivalent Scikit-Learn

```
from sklearn.linear_model import LinearRegression

lin_reg = LinearRegression()
lin_reg.fit(X, y)
lin_reg.intercept_, lin_reg.coef_
```

```
(array([4.21509616]), array([[2.77011339]]))
```

```
lin_reg.predict(X_new)
```

```
array([[4.21509616],
       [9.75532293]])
```

- The LinearRegression class is based on the scipy.linalg.lstsq() function
  - The name stands for "least squares"

Computational complexity

- The Normal Equation computes the inverse of $\boldsymbol{X}^{\boldsymbol{T}}.\boldsymbol{X}$, which is an $n \times n$ matrix (where n is the number of features).
- The *computational complexity* of inverting such a matrix is typically about $O(n^{2.4})$ to $O(n^3)$ (depending on the implementation).
  - In other words, if you double the number of features, you multiply the computation time by roughly $2^{2.4} = 5.3$ to $2^3 = 8$
- The Normal Equation gets very slow when the number of features grows large (e.g., 100,000).
- On the positive side, this equation is linear with regards to the number of instances in the training set (it is $O(m)$), so it handles large training sets efficiently, provided they can fit in memory
- once trained (using the Normal Equation or any other algorithm),
  - predictions are very fast: the computational complexity is linear with regards to both the number of instances and the number of features

Computational complexity

- we will look at very different ways to train a Linear Regression model, better suited for cases where there are a large number of features, or too many training instances to fit in memory

Gradient Descent

- *Gradient Descent (النزول التدريجي)* is 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
- Next Lecture