1. Linear Regression

Linear Regression is a basic statistical and machine learning technique used to model the relationship between a dependent variable (target) and one or more independent variables (features). The goal is to find the line (in the case of one feature) or the hyperplane (in the case of multiple features) that best fits the data.

- **Equation of a line**: In simple linear regression (one feature), the model can be represented by the equation:

y=mx+cy = mx + cy=mx+c

Where:

- y is the predicted value (dependent variable).
- x is the input value (independent variable).
- m is the slope of the line (how steep the line is).
- c is the intercept (the point where the line crosses the y-axis).

How does it work?: The model tries to find the values of mmm and ccc such that the predicted values y are as close as possible to the actual values in the dataset. This is done by minimizing the error (difference between predicted and actual values) using a method like **Least Squares**.

Example:

Suppose we want to predict a person's weight based on their height.

- Height (x): [150, 160, 170, 180, 190]
- Weight (y): [50, 60, 65, 75, 80]

The goal of linear regression is to find the line that best fits these data points.

2. Train-Test Split

In machine learning, the **train-test split** is a method used to evaluate the performance of a model. It involves splitting the dataset into two parts:

1. **Training set**: This portion is used to train the model (e.g., 80% of the data).
2. **Test set**: This portion is used to test the model after training (e.g., 20% of the data).

The reason for splitting is to ensure that the model doesn't "see" the test data during training, which would lead to overfitting (where the model performs well on the training data but poorly on new, unseen data).

How to split:

A typical train-test split is 70%-30%, 80%-20%, or 90%-10%, depending on the size of the dataset.

In Python's scikit-learn, this can be done using the train_test_split function.

```
//

from sklearn.model_selection import train_test_split

# Sample dataset

X = [[150], [160], [170], [180], [190]]  # feature (height)

y = [50, 60, 65, 75, 80]  # target (weight)

# Split data into 80% training and 20% testing

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# X_train, X_test, y_train, y_test now hold the split data

//
```

3. Root Mean Square Error (RMSE)

Root Mean Square Error (RMSE) is a commonly used metric to measure the accuracy of a model’s predictions. It calculates the square root of the average of the squared differences between the predicted and actual values.

The formula for RMSE is:

RMSE=1n∑i=1n(ypredictedi−yactuali)2RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (y_{\text{predicted}}^i - y_{\text{actual}}^i)^2}RMSE=n1i=1∑n(ypredictedi−yactuali)2

Where:

- ypredictediy_{\text{predicted}}^iypredictedi is the predicted value for the ithi^{th}ith data point.
- yactualiy_{\text{actual}}^iyactuali is the actual value for the ithi^{th}ith data point.
- nnn is the number of data points.

How does it work?

- RMSE gives us an idea of how much error the model has, with larger values indicating worse performance.
- A smaller RMSE value means that the model's predictions are closer to the actual values.

Example:

Let’s say you have the following predicted and actual values for weight:

- **Predicted values**: [51, 59, 67, 74, 79]
- **Actual values**: [50, 60, 65, 75, 80]

To calculate RMSE:

1. Find the squared differences:
    - (51-50)^2 = 1
    - (59-60)^2 = 1
    - (67-65)^2 = 4
    - (74-75)^2 = 1
    - (79-80)^2 = 1
2. Calculate the average of these squared differences:
    - (1 + 1 + 4 + 1 + 1) / 5 = 1.6
3. Take the square root of the average:
    - 1.6=1.26\sqrt{1.6} = 1.261.6=1.26

Thus, the **RMSE** is **1.26**.

How to calculate RMSE in Python:

python

Copy code

```
from sklearn.metrics import mean_squared_error

import numpy as np



# Actual values

y_actual = [50, 60, 65, 75, 80]

# Predicted values

y_pred = [51, 59, 67, 74, 79]
```

```
# Calculate RMSE

rmse = np.sqrt(mean_squared_error(y_actual, y_pred))

print("RMSE:", rmse)
```

Conclusion:

- **Linear Regression** is used to predict continuous values based on one or more independent variables.
- **Train-Test Split** helps ensure that the model is evaluated on data it hasn’t seen before, preventing overfitting.
- **RMSE** is a useful metric to measure the accuracy of the model’s predictions, where a lower value means better performance.