CHAPTER 4 TRAINING MODELS

LINEAR REGRESSION:

Linear Regression is a **simple machine learning algorithm** used to model the relationship between **input features (X)** and a **target variable (Y)**. It tries to find the best-fitting straight line (or plane, in higher dimensions) that predicts YYY based on XXX.

**How Does it Work?**

The goal is to find a line that minimizes the difference between the actual values and the predicted values. The line is defined by the equation:

y=θ0+θ1x

Where:

* y: Predicted value
* x: Input feature
* ​θ0: Intercept (where the line crosses the y-axis)
* ​θ1: Slope of the line (how steep it is)

In higher dimensions (with multiple features), the equation becomes:

y=θ0+θ1x1+θ2x2+⋯+θnxn

Here, θ values are the **parameters** we want to find.

**How Do We Find the Best Line?**

We need to measure how well a line fits the data. This is done using a **cost function**, which calculates the error between predicted values (y^​i) and actual values (y).

**Cost Function:**

The most common cost function is the **Mean Squared Error (MSE)**:

m

MSE=1/m ∑ (y^i−yi)square

i=1

Where:

* m: Number of data points
* y^i: Predicted value for the ith data point
* yi​: Actual value for the ith data point

The goal is to minimize this cost function.

THE NORMAL EQUATION:

The **Normal Equation** is a mathematical formula that directly calculates the best parameters (θ\thetaθ) for Linear Regression, minimizing the cost function without requiring iterations.

**Formula:**

T -1 T

θ=( X X) X y

Where:

* X: Input feature matrix.
* y: Target variable (output) vector.
* X^T: Transpose of X.
* (X^T X)^-1: Inverse of X^T X.

GRADIENT DESCENT:

Gradient Descent is a way to find the **best-fit line** (or model) by minimizing the "error" (how far off our predictions are).

**How It Works:**

1. **Start Somewhere**:
   * Pick random starting values for the parameters (θ) of your model.
2. **Measure Error**:
   * Use a formula (called the cost function) to calculate how "bad" your model is. For Linear Regression, this is typically the Mean Squared Error.
3. **Find the Slope**:
   * Figure out which direction to move to reduce the error. This slope is called the **gradient**.
4. **Take a Step**:
   * Adjust your parameters a little bit in the direction that reduces the error.
   * The step size is controlled by the **learning rate** (α\alphaα).
5. **Repeat**:
   * Keep repeating steps 2–4 until the error is as small as possible (this is called **convergence**).

BATCH GRADIENT DESCENT:

Batch Gradient Descent is a version of Gradient Descent where we use **all the data** at once to calculate the direction (slope) and size of the step to adjust our model's parameters.

**How It Works (Step by Step):**

1. **Start with Initial Parameters**:
   * Randomly pick values for the model parameters (θ).
2. **Calculate Predictions**:
   * Use your current parameters to predict the training data outputs.
3. **Measure the Error**:
   * Compare the predictions to the actual values and calculate the overall error (using a cost function like Mean Squared Error).
4. **Find the Gradient**:
   * Use all the data to compute the slope (gradient) of the cost function with respect to each parameter.
5. **Update Parameters**:
   * Adjust the parameters in the direction that reduces the error:

θ= θ−α⋅Gradient

* + - α is the learning rate, which controls how big a step you take.

1. **Repeat Until Convergence**:
   * Keep repeating steps 2–5 until the error stops getting smaller.

**Key Characteristics of Batch Gradient Descent:**

* **Uses all the training data** in each step to compute the gradient.
* **Accurate updates** because it considers the whole dataset.
* Slower for large datasets because it processes all the data at every step.

STOCHASTIC GRADIENT DESCENT:

Stochastic Gradient Descent is a version of Gradient Descent where instead of using **all the data** at once (like Batch Gradient Descent), it updates the model **one data point at a time**.

**How It Works (Step by Step):**

1. **Start Somewhere**:
   * Randomly pick initial values for the model parameters (θ\thetaθ).
2. **Pick One Data Point**:
   * Choose one training example from the dataset.
3. **Calculate Error for This Point**:
   * Compare the model’s prediction for that point to its actual value and compute the error.
4. **Update Parameters**:
   * Adjust the model parameters slightly based on the error from that one data point:

θ=θ−α⋅Gradient

1. **Repeat**:
   * Move to the next data point, compute the error, and update the parameters.
   * Keep doing this for all the data points, again and again, until the model improves.

**Key Characteristics of SGD:**

* **Updates quickly** after processing just one point.
* **Noisy** updates, which can cause the path to bounce around (instead of moving smoothly downhill).
* Often faster for large datasets because it doesn’t wait to process the entire dataset.

MINI BATCH GRADIENT DESCENT:

Mini-Batch Gradient Descent is a middle-ground approach between Batch Gradient Descent (which uses all data) and Stochastic Gradient Descent (which uses one data point at a time). In Mini-Batch Gradient Descent, you divide the data into **small batches** and use one batch at a time to update the model.

**How It Works (Step by Step):**

1. **Divide Data Into Batches**:
   * Split the dataset into smaller groups (batches). Each batch contains a fixed number of data points (e.g., 32 or 64).
2. **Start with Initial Parameters**:
   * Randomly set initial values for the model parameters (θ\thetaθ).
3. **Pick a Batch**:
   * Select one batch of data.
4. **Calculate the Gradient for the Batch**:
   * Use only the selected batch to compute the error and gradient.
5. **Update Parameters**:
   * Adjust the parameters slightly based on the gradient of the batch: θ=θ−α⋅Gradient
6. **Repeat for All Batches**:
   * Go through all batches in the dataset (one pass is called an **epoch**).
   * Keep repeating for multiple epochs until the model improves.

**Key Characteristics of Mini-Batch Gradient Descent:**

* **Balances Speed and Stability**:
  + Faster than Batch Gradient Descent because it processes small groups of data instead of all at once.
  + More stable than Stochastic Gradient Descent because it averages over a batch of data.
* **Batch Size**:
  + Common sizes are 32, 64, or 128. Smaller sizes give faster updates; larger sizes are more stable.

POLYNOMIAL REGRESSION:

Polynomial Regression turns a linear model into a **curved model** by adding new features (powers of existing features) and then fitting a straight line to this extended set of features.

1. **Start with Your Data**:
   * Let’s say your data has one feature (like house size) and a target (like price). If the relationship between them is not a straight line, a simple linear model won’t work well.
2. **Create New Features**:
   * To capture the curve, you can add the **square** of the feature (and higher powers if needed). For example, if your feature is X, you create a new feature X^2.
3. **Use Linear Regression**:
   * Once you have the new features (like X and X^2X2), you can apply **Linear Regression** to this expanded data. Even though the data is nonlinear, you're still using a linear model, but with more complex features.
4. **Get the Model**:
   * Now the model will learn the best coefficients for both X and X^2, allowing it to fit a curve to the data.

LEARNING CURVE:

A **Learning Curve** is a graph that shows how a machine learning model's performance improves as it is trained on more data or as it trains for more time (in terms of epochs). It helps you understand how well the model is learning over time and whether it's improving or facing issues like overfitting or underfitting.

REGULARIZATION OF LINEAR MODEL:

**Regularization** is a technique used to prevent a model from **overfitting** the training data. Overfitting happens when the model becomes too complex, learning noise and details from the training data that do not generalize well to new, unseen data. Regularization helps by adding a penalty to the model's complexity, which discourages it from fitting the data too closely.

RIDGE REGRESSION:

Ridge Regression is a type of **regularized Linear Regression** that applies **L2 regularization** to the cost function. The goal is to improve the generalization of the model and prevent overfitting by penalizing large coefficients.

The **L2 penalty** is the sum of the squares of the model parameters θ\thetaθ (excluding the intercept θ0\theta\_0θ0​), multiplied by a regularization parameter λ.

n 2

L2 penalty=λ ∑ θ

j=1 j

Where:

* θj​ are the coefficients of the model (parameters),
* λ is the **regularization parameter** that controls the strength of the penalty,
* n is the number of features (or parameters) in the model.

LASSO REGRESSION:

Lasso Regression is a type of **regularized linear regression** that uses **L1 regularization** to improve the model by preventing overfitting and simplifying the model. Lasso stands for **Least Absolute Shrinkage and Selection Operator**.

ELASTIC NET:

Elastic Net is a regularization technique that combines the strengths of both **Lasso Regression** (L1 regularization) and **Ridge Regression** (L2 regularization). It is particularly useful when there are many correlated features in the data. Elastic Net allows for both **feature selection** (like Lasso) and **shrinkage of coefficients** (like Ridge), making it a versatile tool for linear models.

EARLY STOPPING:

Early stopping is a technique used to prevent **overfitting** during the training of machine learning models, particularly for models that involve iterative learning processes, such as **Gradient Descent**. It involves stopping the training process before the model has had a chance to fully converge to the optimal solution.

The idea is to monitor the model's performance on a **validation set** and stop training when the performance stops improving or starts getting worse, rather than continuing to train until a predefined number of epochs or iterations.

LOGISTIC REGRESSION:

Logistic Regression is a type of **classification** algorithm used to predict the probability of a binary outcome. Unlike **Linear Regression**, which predicts continuous values, **Logistic Regression** predicts a value between 0 and 1, representing the probability of one of two possible outcomes.

It's called "Logistic" because it uses the **logistic function**, also known as the **sigmoid function**, to map the output of a linear equation to a probability.