Question Bank for Machine Learning

Stratified Sampling

Consider a factory producing three different types of products: A, B, and C. The production rates are 40%, 30%, and 30%, respectively. If you want to select a sample of 50 products for quality control, use stratified sampling.

Solution:

Determine the number of samples to be selected from each stratum:

For product A: 50 * 0.40 = 20 samples

For product B: 50 * 0.30 = 15 samples

For product C: 50 * 0.30 = 15 samples

Randomly select the specified number of samples from each stratum.

Systematic Sampling

Suppose you have a list of 200 employees, and you want to select a sample of 20 employees using systematic sampling. Choose a random starting point between 1 and 10, and then select every 10th employee from the list.

Solution:

Choose a random starting point, let's say 5.

Select every 10th employee from the list:

Employee 5, 15, 25, ..., 195

Cluster Sampling

Consider a university with 10 departments. If you want to survey students and decide to use cluster sampling by selecting three departments randomly, and surveying all students in those departments.

Solution:

Randomly select three departments from the ten.

Survey all students within the chosen three departments.

Q. Suppose you have a dataset containing the following numerical feature values:

{3.2,4.5,2.1,5.8,3.9,6.7,2.8,4.1,5.3,3.0}

Perform equal-width discretization into three bins.

1. Calculate the range of the feature:

   $$\text{Range} = \text{Max(Data)} - \text{Min(Data)}$$
   $$\text{Range} = 6.7 - 2.1 = 4.6$$

2. Calculate the width of each bin:

   $$\text{Width} = \frac{\text{Range}}{\text{Number of Bins}}$$
   $$\text{Width} = \frac{4.6}{3} \approx 1.53$$

3. Define the bins:
   - Bin 1: $[2.1, 3.63)$
   - Bin 2: $[3.63, 5.16)$
   - Bin 3: $[5.16, 6.7]$

4. Discretize the data:

   $$\{3.2, 4.5, 2.1, 5.8, 3.9, 6.7, 2.8, 4.1, 5.3, 3.0\}$$

   Discretized data:

   $$\{2, 3, 1, 3, 2, 3, 2, 3, 3, 2\}$$

Binarization

Consider a dataset with the following continuous feature:

{1.2,2.5,0.8,3.7,2.0,4.1,1.6,3.0,2.3,1.9}

Perform binarization using a threshold of 2.0.

Define the threshold:

Threshold=2.0Threshold=2.0

Binarize the data:

{1.2,2.5,0.8,3.7,2.0,4.1,1.6,3.0,2.3,1.9}{1.2,2.5,0.8,3.7,2.0,4.1,1.6,3.0,2.3,1.9}

Binarized data:

{0,1,0,1,0,1,0,1,0,0}{0,1,0,1,0,1,0,1,0,0}

Q. How Outliers can be handled while developing Machine Learning models. explain with examples

Sol.

Handling outliers in machine learning models is crucial for improving model performance and robustness. Outliers can significantly impact the training process and may lead to models that don't generalize well. Here are several common approaches to handle outliers

Removing Outliers:

Example: Suppose you have a dataset of house prices, and some houses have unrealistically high prices. You can remove outliers based on a threshold.

import pandas as pd

```
# Sample dataset with outliers

data = {'House_Price': [300000, 400000, 500000, 7000000, 450000, 550000, 600000]}

df = pd.DataFrame(data)

# Remove outliers (e.g., anything above 1 million)

threshold = 1000000

df_cleaned = df[df['House_Price'] < threshold]
```

Transforming Data:

Example: Apply a transformation to make the data less sensitive to outliers. For instance, use a logarithmic transformation.

```
import numpy as np

# Sample dataset with skewed values

data = {'Feature': [100, 200, 500, 50, 1200, 1500, 80, 90]}

df = pd.DataFrame(data)

# Apply log transformation

df['Log_Feature'] = np.log p(df['Feature'])
```

Binning or Discretization:

Example: Convert continuous data into discrete bins, reducing the impact of extreme values.

```
# Using pandas cut for binning

df['Binned_Feature'] = pd.cut(df['Feature'], bins=3, labels=False)
```

Winsorizing:

Example: S from scipy.stats.mstats import winsorize

```
# Sample dataset with outliers

data = {'Income': [50000, 60000, 80000, 120000, 200000, 300000, 1000000]}

df = pd.DataFrame(data)


# Winsorize the 'Income' column

df['Winsorized_Income'] = winsorize(df['Income'], limits=[0, 0.05]) et extreme values to a specified percentile.
```

Winsorizing is a statistical technique used to handle outliers in a dataset by replacing extreme values with less extreme values, specifically those at a certain percentile. Rather than removing outliers entirely, Winsorizing trims or "winsorizes" the dataset by capping extreme values at a predetermined threshold.

Here's a step-by-step explanation of Winsorizing:

Set the Limits:

- You define the limits or percentiles beyond which values will be considered outliers and replaced. For example, if you set the lower limit at the 10th percentile and the upper limit at the 90th percentile, values below the 10th percentile and above the 90th percentile will be replaced.

Apply Winsorizing:

- The Winsorizing process involves replacing values outside the specified limits with the values at those limits. For instance, if a data point is below the lower limit, it will be replaced by the value at the lower limit; similarly, if a data point is above the upper limit, it will be replaced by the value at the upper limit.

Preserve Information:

- Winsorizing preserves information from extreme values while reducing their impact on statistical analyses or machine learning models. This is particularly useful when extreme values are influential but are considered to be potential outliers.

Control the Degree of Winsorizing:

- You have control over the degree of Winsorizing by adjusting the specified limits. A higher limit would be more conservative, preserving more extreme values, while a lower limit would be more aggressive, replacing more extreme values.

Example:

Suppose you have a dataset of monthly incomes:

{5000,5500,6000,6200,7000,8000,9000,9500,10000,12000,30000}

We want to Winsorize this dataset by setting the lower limit at the 10th percentile and the upper limit at the 90th percentile.

Step 1: Set the Limits

Lower Limit (10th Percentile): Lower Limit=percentile(10)

Upper Limit (90th Percentile): Upper Limit=percentile(90)

Step 2: Apply Winsorizing

Replace values below the lower limit with the lower limit.

Replace values above the upper limit with the upper limit.

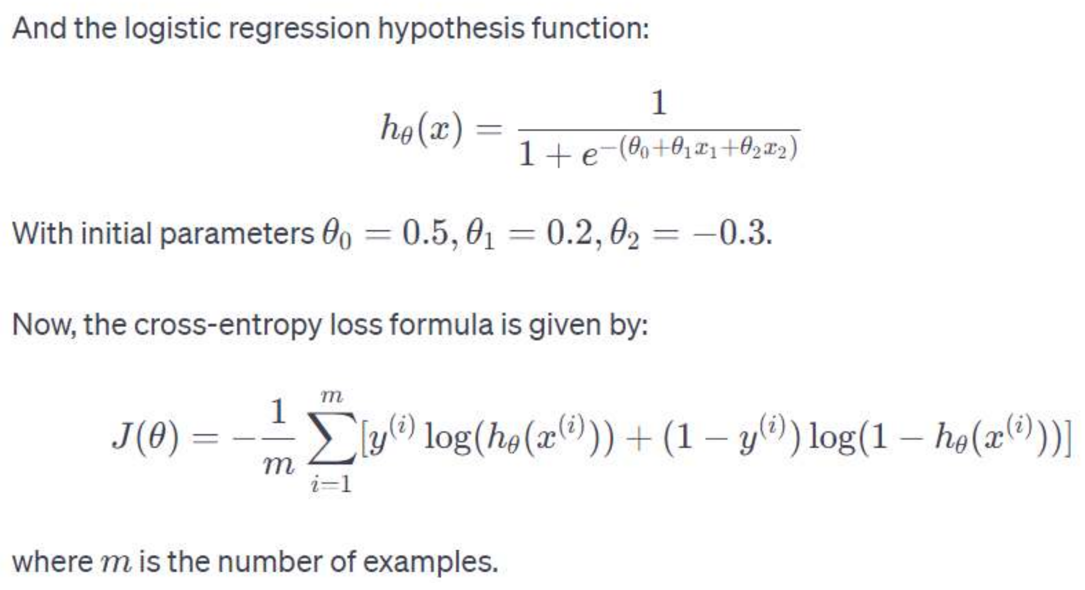
Consider a dataset of student exam scores and whether they passed (1) or failed (0). We want to perform logistic regression to predict the probability of passing based on the exam scores.

Here's the updated dataset:

Exam 1 Score:[45,60,75,30,85]

Exam 2 Score:[50,65,80,40,90]

Pass (1) or Fail (0):[0,1,1,0,1]



And the logistic regression hypothesis function:

$$h_\theta(x) = \frac{1}{1 + e^{-(\theta_0 + \theta_1 x_1 + \theta_2 x_2)}}$$

With initial parameters $\theta_0 = 0.5, \theta_1 = 0.2, \theta_2 = -0.3.$

Now, the cross-entropy loss formula is given by:

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} [y^{(i)} \log(h_\theta(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}))]$$

where m is the number of examples.

Numerical :Logistic Regression

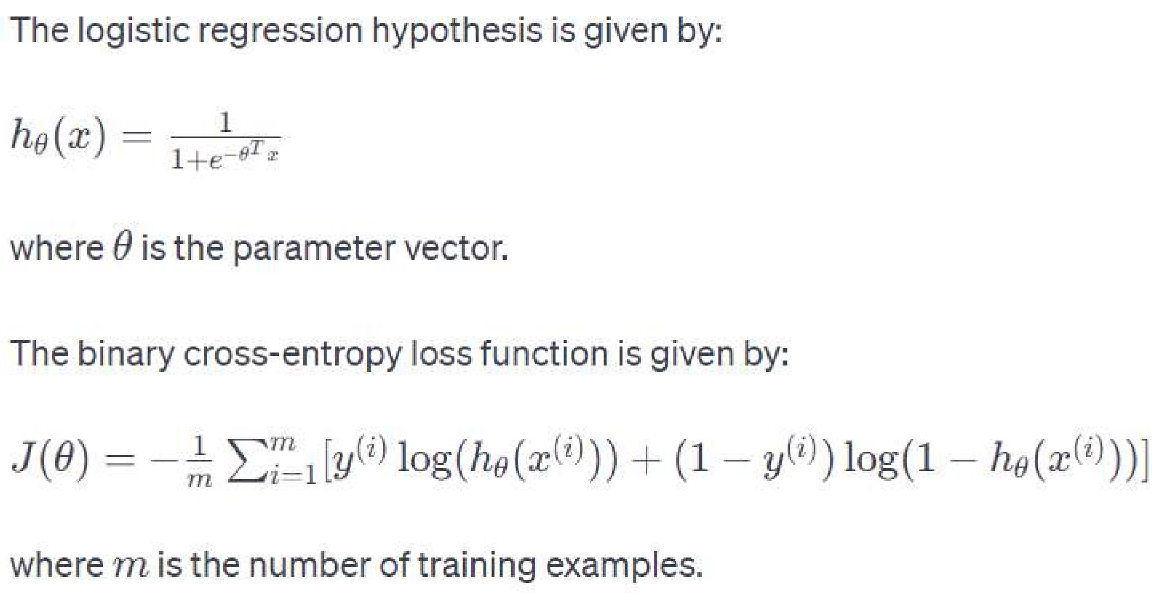
Use logistic regression as the algorithm, and the binary cross-entropy loss function.

Hours Studied (x)	Exam Result (y)
2	0
3	0
4	0
5	1
6	1
7	1
8	1
9	1

Hours Studied (x)	Exam Result (y)
10	1

Here, x is the number of hours studied, and y is the binary result of passing (1) or failing (0).

The logistic regression hypothesis is given by:



The logistic regression hypothesis is given by:

$$h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$$

where θ is the parameter vector.

The binary cross-entropy loss function is given by:

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}\log(h_\theta(x^{(i)})) + (1-y^{(i)})\log(1-h_\theta(x^{(i)}))]$$

where m is the number of training examples.

1. **Initialize Parameters:**

   Let's assume initial values for the parameters θ_0 and θ_1. For example, $\theta_0 = 0.5$ and $\theta_1 = 0.5$.
2. **Hypothesis:**

   Calculate the hypothesis $h_\theta(x)$ for each example using the given formula.
   $h_\theta(x) = \frac{1}{1+e^{-(\theta_0+\theta_1 x)}}$
3. **Compute Loss:**

   Substitute the hypothesis into the binary cross-entropy loss function and calculate the loss.
   $J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}\log(h_\theta(x^{(i)})) + (1-y^{(i)})\log(1-h_\theta(x^{(i)}))]$
4. **Gradient Descent:**

   Update the parameters θ_0 and θ_1 using gradient descent.
   $\theta_j = \theta_j - \alpha\frac{\partial J(\theta)}{\partial \theta_j}$
   where α is the learning rate.
5. **Repeat:**

   Repeat steps 2-4 until convergence.

1. **Initialize Parameters:**

   Let's assume $\theta_0 = 0.5$ and $\theta_1 = 0.5$.

2. **Hypothesis:**

   Calculate the hypothesis $h_\theta(x)$ for each example using the given formula:

   $h_\theta(x) = \frac{1}{1+e^{-(\theta_0+\theta_1 x)}}$

   For the first example ($x = 2$):

   $h_\theta(2) = \frac{1}{1+e^{-(0.5+0.5\times 2)}}$

   $h_\theta(2) = \frac{1}{1+e^{-2}}$

   $h_\theta(2) = \frac{1}{1+0.135}$

   $h_\theta(2) \approx 0.88$

   Repeat this process for all examples.

3. **Compute Loss:**

   Substitute the hypothesis into the binary cross-entropy loss function and calculate the loss:

   $J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}\log(h_\theta(x^{(i)})) + (1 - y^{(i)})\log(1 - h_\theta(x^{(i)}))]$

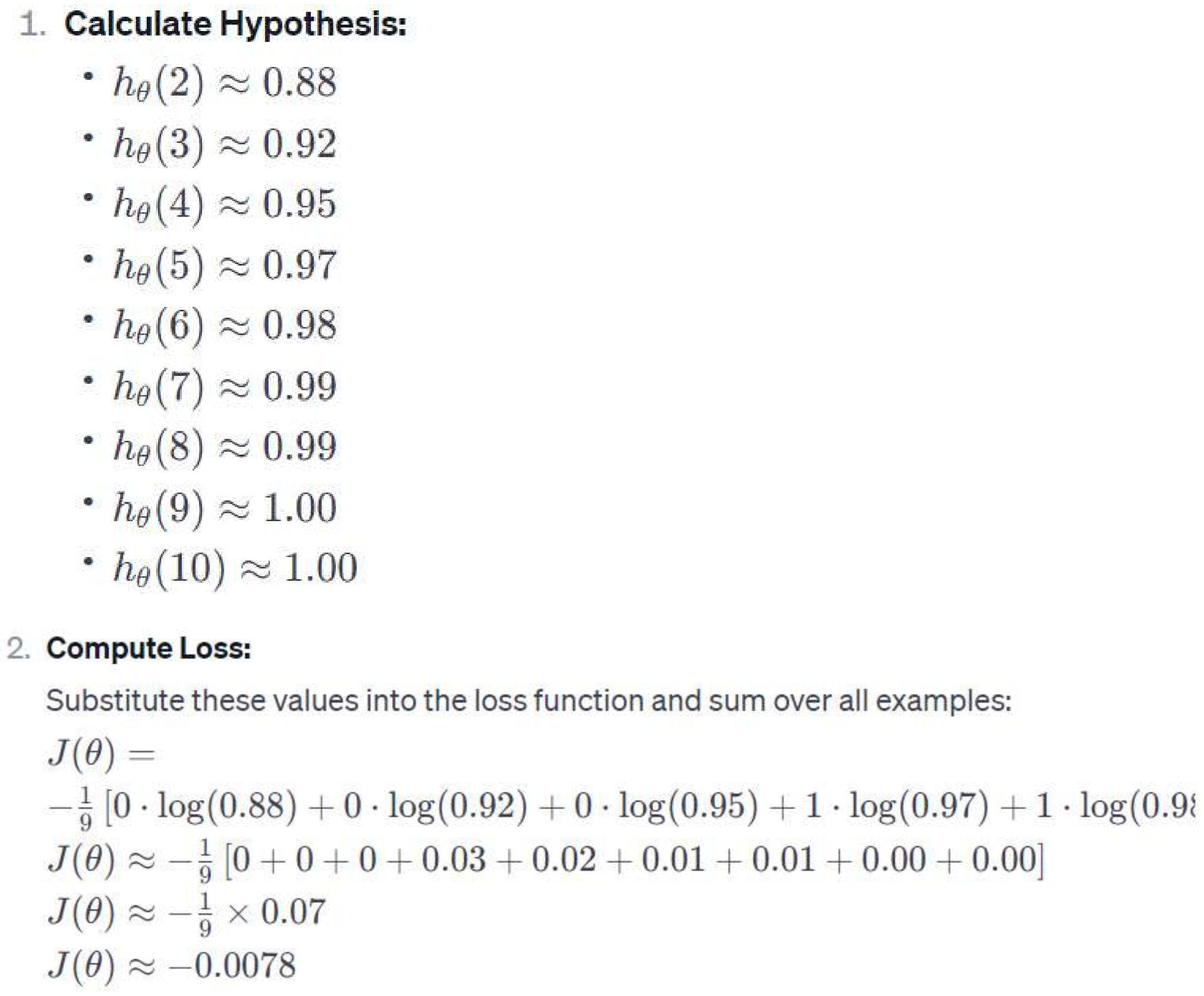
   For the given dataset, compute the loss by summing over all examples.

The binary cross-entropy loss function is given by:

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}\log(h_\theta(x^{(i)})) + (1 - y^{(i)})\log(1 - h_\theta(x^{(i)}))]$$

Using the hypothesis $h_\theta(x) = \frac{1}{1+e^{-(\theta_0+\theta_1 x)}}$, and assuming the initial parameters $\theta_0 = 0.5$ and $\theta_1 = 0.5$, let's calculate the loss for each example and then compute the average.

For simplicity, let's assume a learning rate (α) of 0.01.



1. **Calculate Hypothesis:**
   - $h_\theta(2) \approx 0.88$
   - $h_\theta(3) \approx 0.92$
   - $h_\theta(4) \approx 0.95$
   - $h_\theta(5) \approx 0.97$
   - $h_\theta(6) \approx 0.98$
   - $h_\theta(7) \approx 0.99$
   - $h_\theta(8) \approx 0.99$
   - $h_\theta(9) \approx 1.00$
   - $h_\theta(10) \approx 1.00$

2. **Compute Loss:**

   Substitute these values into the loss function and sum over all examples:

   $J(\theta) =$
   $-\frac{1}{9}\left[0 \cdot \log(0.88) + 0 \cdot \log(0.92) + 0 \cdot \log(0.95) + 1 \cdot \log(0.97) + 1 \cdot \log(0.9\right.$
   $J(\theta) \approx -\frac{1}{9}\left[0 + 0 + 0 + 0.03 + 0.02 + 0.01 + 0.01 + 0.00 + 0.00\right]$
   $J(\theta) \approx -\frac{1}{9} \times 0.07$
   $J(\theta) \approx -0.0078$

the value of the binary cross-entropy loss for these initial parameters and this dataset is approximately -0.0078. Note that the negative sign is often dropped in practice, and the loss is considered as a positive value.

Curse of Dimensionality

Suppose you have a dataset with 100 samples, each represented by a single feature. You decide to add more features to improve the model's performance, but you notice that as you increase the number of features, the model's performance starts to degrade. You are using a nearest-neighbor algorithm for classification.

a) If each feature has a range of [0, 1], how many samples would you need to maintain a certain level of data density in a one-dimensional space?

b) Now, consider that you decide to use two features. What is the equivalent number of samples needed to maintain the same level of data density in a two-dimensional space?

c) Discuss the implications of the curse of dimensionality based on your findings in parts (a) and (b).

Solution:

a) In a one-dimensional space, maintaining a certain level of data density requires more samples as the range of each feature increases. If each feature has a range of [0, 1], and you want to maintain a similar level of data density, the number of samples needed can be calculated using the formula:

$$\text{Number of samples} = \left(\frac{\text{Number of samples in 1D}}{\text{Range in 1D}}\right)^{\text{Dimensionality}}$$

Given that you have 100 samples in 1D and a range of 1, the number of samples needed for a one-dimensional space is:

$$\text{Number of samples in 1D} = 100$$
$$\text{Range in 1D} = 1$$

$$\text{Number of samples needed in 1D} = \left(\tfrac{100}{1}\right)^{1} = 100$$

b) Now, with two features, the equivalent number of samples needed in a two-dimensional space is given by:

$$\text{Number of samples needed in 2D} = \left(\tfrac{100}{1}\right)^{2} = 10000$$

c) The implications of the curse of dimensionality are evident in the drastic increase in the number of samples needed as the dimensionality of the space increases. In this case, moving from 1D to 2D resulted in a 100-fold increase in the number of samples needed to maintain the same data density. This makes it challenging to collect and store sufficient data in high-dimensional spaces, and it can lead to issues such as overfitting and increased computational complexity in machine learning models.

PCA (Principal Component Analysis)

Consider a dataset with the following two-dimensional data points:

$$X = \begin{bmatrix} 2 & 3 & 5 & 7 & 11 \\ 5 & 7 & 11 & 13 & 17 \end{bmatrix}$$

perform PCA step by step to find the principal components.

Step 1: Center the Data

Compute the mean of each feature and subtract it from each data point to center the data.

$$\bar{x}_1 = \frac{2+3+5+7+11}{5} = \frac{28}{5} = 5.6$$
$$\bar{x}_2 = \frac{5+7+11+13+17}{5} = \frac{53}{5} = 10.6$$

Centered Data (X_c):

$$X_c = \begin{bmatrix} -3.6 & -2.6 & -0.6 & 1.4 & 5.4 \\ -5.6 & -3.6 & 0.4 & 2.4 & 6.4 \end{bmatrix}$$

Step 2: Compute Covariance Matrix

The covariance matrix C is given by $C = \frac{1}{n} X_c X_c^T$, where n is the number of data points.

$$C = \frac{1}{5} \begin{bmatrix} 44.8 & 54.8 \\ 54.8 & 70.8 \end{bmatrix}$$

Step 3: Compute Eigenvalues and Eigenvectors

Compute the eigenvalues (λ) and eigenvectors (v) of the covariance matrix.

The characteristic equation $|C - \lambda I| = 0$ leads to:

Substitute the values of C and solve for the eigenvalues:

$$\det(C - \lambda I) = \begin{vmatrix} 44.8 - \lambda & 54.8 \\ 54.8 & 70.8 - \lambda \end{vmatrix} = (44.8 - \lambda)(70.8 - \lambda) - (54.8)^2 = 0$$

Expanding and simplifying:

$$(44.8 - \lambda)(70.8 - \lambda) - (54.8)^2 = 0$$

$$(44.8 \times 70.8) - \lambda(70.8) - \lambda(44.8) + \lambda^2 - (54.8)^2 = 0$$

$$3166.24 - 70.8\lambda - 44.8\lambda + \lambda^2 - 2995.84 = 0$$

$$\lambda^2 - 115.6\lambda + 170.4 = 0$$

Now, solve this quadratic equation for λ using the quadratic formula:

$$\lambda = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

where $a = 1, b = -115.6$, and $c = 170.4$.

$$\lambda = \frac{115.6 \pm \sqrt{(-115.6)^2 - 4(170.4)}}{2}$$

$$\lambda = \frac{115.6 \pm \sqrt{13322.56 - 681.6}}{2}$$

$$\lambda = \frac{115.6 \pm \sqrt{12641.96}}{2}$$

$$\lambda_1 \approx \frac{115.6 + 112.56}{2} \approx 114.08$$

$$\lambda_2 \approx \frac{115.6 - 112.56}{2} \approx 1.52$$

Now that we have the eigenvalues, we can substitute them back into the equation $Cv = \lambda v$ to find the corresponding eigenvectors.

For $\lambda_1 \approx 114.08$:

$$(C - \lambda_1 I) \times v_1 = 0$$

$$\begin{bmatrix} -69.28 & 54.8 \\ 54.8 & -43.28 \end{bmatrix} \times \begin{bmatrix} v_{11} \\ v_{21} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

This system of equations can be solved to find v_1.

This leads to the following system of equations:

$$-69.28v_{11} + 54.8v_{21} = 0$$

$$54.8v_{11} - 43.28v_{21} = 0$$

Solving this system of equations, we find that $v_{11} = 0.685$ and $v_{21} = 0.729$.

So, for $\lambda_1 \approx 114.08$, the corresponding eigenvector v_1 is approximately:

$$v_1 \approx \begin{bmatrix} 0.685 \\ 0.729 \end{bmatrix}$$

For $\lambda_2 \approx 1.52$:

$$(C - \lambda_2 I) \times v_2 = 0$$

$$\begin{bmatrix} 43.28 & 54.8 \\ 54.8 & 69.28 \end{bmatrix} \times \begin{bmatrix} v_{12} \\ v_{22} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Solving this system gives v_2.

This leads to the following system of equations:

$$43.28v_{12} + 54.8v_{22} = 0$$

$$54.8v_{12} + 69.28v_{22} = 0$$

Solving this system of equations, we find that $v_{12} = -0.729$ and $v_{22} = 0.685$.

So, for $\lambda_2 \approx 1.52$, the corresponding eigenvector v_2 is approximately:

$$v_2 \approx \begin{bmatrix} -0.729 \\ 0.685 \end{bmatrix}$$

$$\lambda^2 - 115.6\lambda + 170.4 = 0$$

Now, solve this quadratic equation for λ using the quadratic formula:

$$\lambda = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

where $a = 1, b = -115.6$, and $c = 170.4$.

$$\lambda = \frac{115.6 \pm \sqrt{(-115.6)^2 - 4(170.4)}}{2}$$

$$\lambda = \frac{115.6 \pm \sqrt{13322.56 - 681.6}}{2}$$

$$\lambda = \frac{115.6 \pm \sqrt{12641.96}}{2}$$

$$\lambda_1 \approx \frac{115.6 + 112.56}{2} \approx 114.08$$

$$\lambda_2 \approx \frac{115.6 - 112.56}{2} \approx 1.52$$

Step 4: Choose Principal Components

Sort the eigenvalues in descending order. The first principal component is associated with the largest eigenvalue.

So, the first principal component ($PC1$) is v_1, and the second principal component ($PC2$) is v_2.

Now, we'll sort the eigenvalues in descending order:

$$\text{Sort}(\lambda_1, \lambda_2) = (\lambda_1, \lambda_2)$$

So, the sorted eigenvalues are $\lambda_1 \approx 114.08$ and $\lambda_2 \approx 1.52$.

The corresponding sorted eigenvectors are:

$$PC1 = v_1 \approx \begin{bmatrix} 0.685 \\ 0.729 \end{bmatrix}$$
$$PC2 = v_2 \approx \begin{bmatrix} -0.729 \\ 0.685 \end{bmatrix}$$

Step 5: Project Data onto Principal Components

Project the centered data onto the chosen principal components.

$$Y = \begin{bmatrix} PC1 \\ PC2 \end{bmatrix}^T \times X_c$$

$$Y = \begin{bmatrix} 0.685 & -0.729 \\ 0.729 & 0.685 \end{bmatrix} \times \begin{bmatrix} -3.6 & -2.6 & -0.6 & 1.4 & 5.4 \\ -5.6 & -3.6 & 0.4 & 2.4 & 6.4 \end{bmatrix}$$

The resulting matrix Y contains the data points projected onto the principal components.

Q. Explain the importance of PCA in Machine Learning. Give realworld application of it.

Real-world Case Study: Facial Recognition and Dimensionality Reduction

Background:

Facial recognition is widely used in various applications, including security systems, mobile devices, and social media. One of the challenges in facial recognition is dealing with high-dimensional data, as each pixel in an image contributes to the overall feature space. This case study illustrates the importance of Principal Component Analysis (PCA) in reducing dimensionality for facial recognition tasks.

Problem:

Consider a dataset of facial images, where each image is represented by a large number of pixels. The raw image data results in a high-dimensional feature space, making it computationally expensive and challenging to build accurate facial recognition models. Additionally, high dimensionality can lead to overfitting and increased computational requirements.

Importance of PCA:

Dimensionality Reduction: PCA is employed to reduce the dimensionality of the facial image dataset. It transforms the data into a new set of uncorrelated variables (principal components) that capture the most significant variations in the data.

By selecting a subset of principal components that explain the majority of the variance, PCA reduces the dimensionality while retaining essential information.

Computational Efficiency: The reduced dimensionality allows for more efficient computation of machine learning models. Training and testing facial recognition algorithms on lower-dimensional data significantly speeds up the process, making real-time applications feasible.

Noise Reduction: High-dimensional data often contains noise or irrelevant information. PCA helps filter out noise by emphasizing the principal components that contribute the most to the dataset's variance. This improves the model's generalization and robustness.

Improved Model Performance: By focusing on the most informative features, PCA enhances the performance of facial recognition models. It mitigates the risk of overfitting and allows the model to generalize better to unseen data.

Q. What are various Feature Engineering methods in Machine Learning. explain with examples.

Feature engineering is a crucial aspect of machine learning that involves transforming raw data into a format that can better represent the underlying patterns of the problem to improve model performance.

Imputation:

Method: Fill missing values in features with appropriate values.

Example: If a dataset has missing age values, impute them with the mean or median age of the entire dataset.

Scaling:

Method: Standardize or normalize numerical features to bring them to a common scale.

Example: Scale features like income or age to have zero mean and unit variance using Z-score normalization.

One-Hot Encoding:

Method: Convert categorical variables into binary vectors (0 or 1) to make them suitable for machine learning algorithms.

Example: Convert "Color" categories (Red, Blue, Green) into separate binary columns.

Label Encoding:

Method: Encode categorical labels with numerical values.

Example: Encode "Small," "Medium," and "Large" as 0, 1, and 2, respectively.

Binning:

Method: Group numerical data into bins or intervals to convert continuous features into categorical ones.

Example: Convert ages into bins like (0-18), (19-35), (36-50), (51+).

Polynomial Features:

Method: Create new features by raising existing features to higher powers, helping capture non-linear relationships.

Example: If you have a feature "x," create a new feature "x^2" or "x^3."

Interaction Terms:

Method: Combine two or more features to create new ones that might have a more significant impact on the target.

Example: If you have features for "length" and "width," create an interaction term like "area = length * width."

Feature Scaling:

Method: Scale numerical features to a specific range, making them more robust to variations in magnitude.

Example: Scale features between 0 and 1 or -1 and 1.

Log Transform:

Method: Apply the natural logarithm to features to handle skewed distributions and make the data more symmetric.

Example: Log-transform skewed variables like income or population.

Target Encoding (Mean Encoding):

Method: Encode categorical variables with the mean of the target variable for each category.

Example: Encode categorical variables based on the mean of the target variable for each category in a classification problem.

Time Features:

Method: Extract relevant features from date and time variables, such as day of the week, month, or year.

Example: For a timestamp feature, extract features like day of the week or time of the day.

Frequency Encoding:

Method: Encode categorical variables based on their frequency of occurrence.

Example: Encode categories based on how often they appear in the dataset.

One-Hot Encoding

consider a simple example where we have a dataset with a categorical variable "Color" that can take on three values: Red, Blue, and Green. We want to apply One-Hot Encoding to represent this categorical variable as binary vectors.

Original dataset:

	Color
1	Red
2	Blue
3	Green
4	Red

	Color
5	Blue

One-Hot Encoding involves creating binary columns for each category. Let's represent "Color" using One-Hot Encoding:

	Color_Red	Color_Blue	Color_Green
1	1	0	0
2	0	1	0
3	0	0	1
4	1	0	0
5	0	1	0

Here's the explanation:

The original "Color" column has three categories: Red, Blue, and Green.

We create three binary columns, one for each category (Color_Red, Color_Blue, Color_Green).

If an example has a certain color, the corresponding binary column is set to 1; otherwise, it is set to 0.

For instance:

The first row has "Red" in the original "Color" column, so Color_Red is set to 1, and Color_Blue and Color_Green are set to 0.

The second row has "Blue," so Color_Blue is set to 1, and the other binary columns are set to 0.

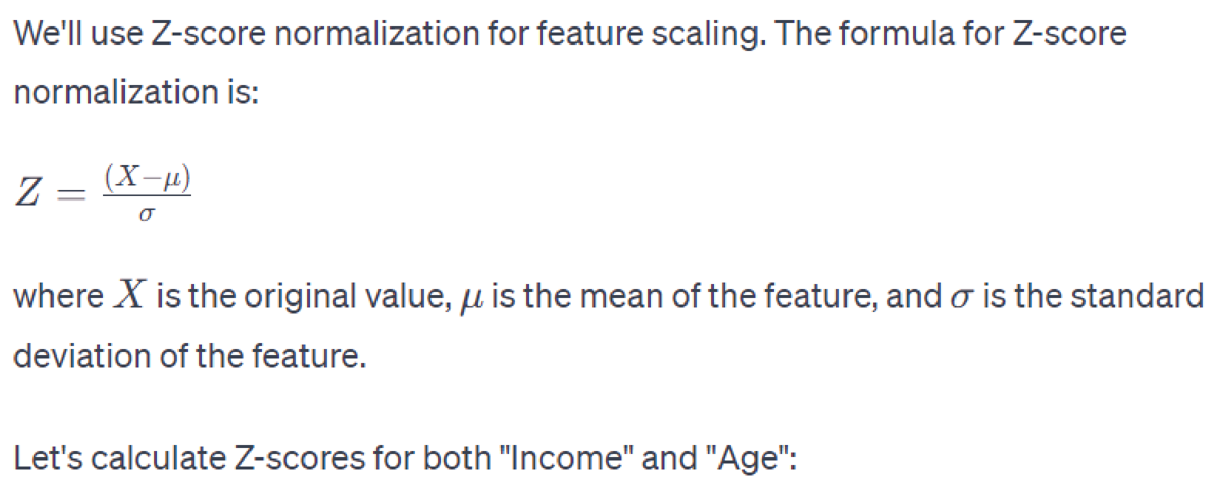
Feature Scaling

we have a dataset with two numerical features: "Income" and "Age." We want to apply feature scaling to bring these features to a common scale.

Original dataset:

	Income ($)	Age (years)
1	50000	30
2	80000	40
3	120000	35
4	60000	25
5	90000	45

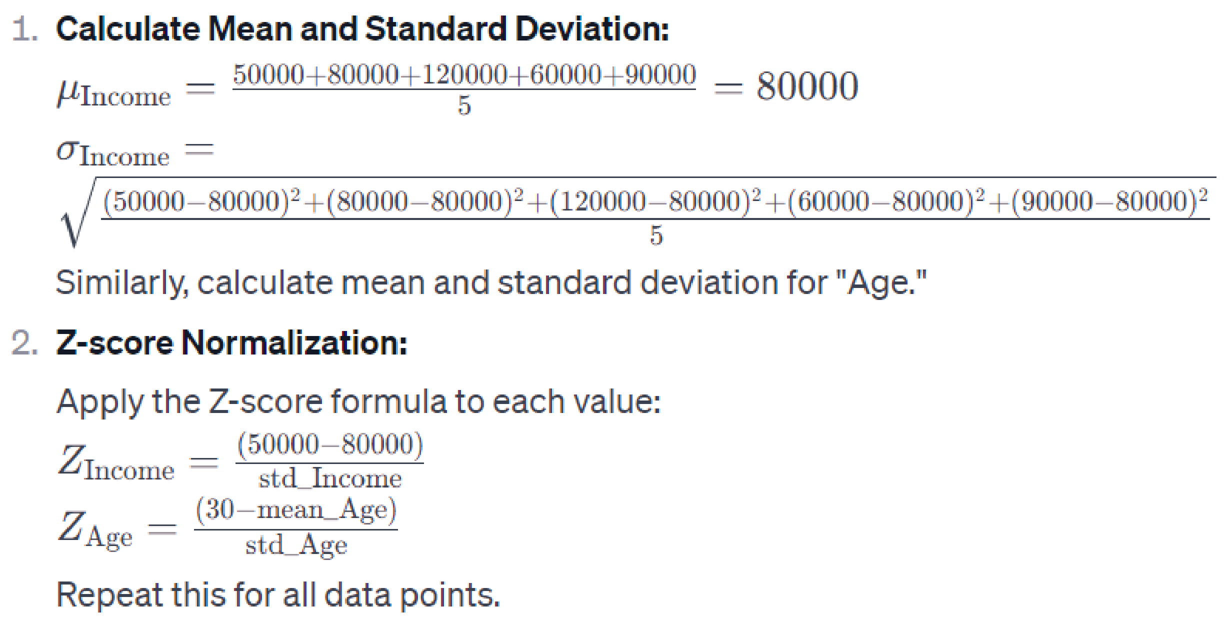
We'll use Z-score normalization for feature scaling. The formula for Z-score normalization is:


We'll use Z-score normalization for feature scaling. The formula for Z-score normalization is:

$$Z = \frac{(X-\mu)}{\sigma}$$

where X is the original value, μ is the mean of the feature, and σ is the standard deviation of the feature.

Let's calculate Z-scores for both "Income" and "Age":



1. **Calculate Mean and Standard Deviation:**

   $$\mu_{\text{Income}} = \frac{50000+80000+120000+60000+90000}{5} = 80000$$

   $$\sigma_{\text{Income}} = \sqrt{\frac{(50000-80000)^2+(80000-80000)^2+(120000-80000)^2+(60000-80000)^2+(90000-80000)^2}{5}}$$

   Similarly, calculate mean and standard deviation for "Age."

2. **Z-score Normalization:**

   Apply the Z-score formula to each value:

   $$Z_{\text{Income}} = \frac{(50000-80000)}{\text{std\_Income}}$$

   $$Z_{\text{Age}} = \frac{(30-\text{mean\_Age})}{\text{std\_Age}}$$

   Repeat this for all data points.

Calculate Accuracy, Precision, Recall and the F1-score w.r.t below confusion matrix

N=195	Predicted (YES)	Predicted (NO)
Actual (YES)	150	10
Actual (NO)	20	100

Accuracy = (TP + TN) / (TP + TN + FP + FN) = (150 + 100) /(150+100+20+10)= **0.89**

Recall= TP/ (TP+FN) = 150/(150+10) = **0.93**

Precision: TP/(TP+FP)= 150/(150+20) = **0.88**

F-measure = (2*Recall*Precision)/(Recall+Presision) = (2*0.93*0.88)/(0.93+0.88) = **0.90**

Q. For the given table calculate the Mean square error, root mean square and R-square error.

AGE	FAILURES	PREDICTION
10	15	26
20	30	32
40	40	44

50	55	50
70	75	62
90	90	74

`

Mean square error=98.5

root mean square=9.9

R-Square=0.85

Find a linear regression equation for the following two sets of data:

x	2	4	6	8
y	3	7	5	10

Sol: To find the linear regression equation we need to find the value of Σx, Σy, Σx

2

2

and Σxy

Construct the table and find the value

x	**y**	**x²**	**xy**
2	3	4	6
4	7	16	28
6	5	36	30
8	10	64	80
Σx = 20	Σy = 25	Σx² = 120	Σxy = 144

The formula of the linear equation is y=a+bx. Using the formula we will find the value of a and b

a= $\frac{\left(\sum_{Y}\right)\left(\sum_{X^2}\right)-\left(\sum_{X}\right)\left(\sum_{XY}\right)}{n\left(\sum_{x^2}\right)-\left(\sum_{x}\right)^2}$

Now put the values in the equation

$$a=\frac{25\times 120-20\times 144}{4\times 120-400}$$

a= $\frac{120}{80}$

a=1.5

b= $\frac{n\left(\sum_{XY}\right)-\left(\sum_{X}\right)\left(\sum_{Y}\right)}{n\left(\sum_{x^2}\right)-\left(\sum_{x}\right)^2}$

Put the values in the equation

$$b = \frac{4 \times 144 - 20 \times 25}{4 \times 120 - 400}$$

b= $\frac{76}{80}$

b=0.95

Hence we got the value of a = 1.5 and b = 0.95

The linear equation is given by

Y = a + bx

Now put the value of a and b in the equation

Hence equation of linear regression is y = 1.5 + 0.95x

Evaluation Metrics for Regression

Dataset:

$$X = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]$$
$$Y_{\text{true}} = [2, 4, 5, 4, 5, 8, 9, 10, 12, 11]$$
$$Y_{\text{pred}} = [1.5, 3, 4, 4.5, 5, 7, 8, 9.5, 11, 10.5]$$

Here, X represents the independent variable, Y_{true} represents the true values of the dependent variable, and Y_{pred} represents the predicted values by a linear regression model.

1. **Mean Absolute Error (MAE):**
   $MAE = \frac{1}{n}\sum_{i=1}^{n} |Y_{\text{true},i} - Y_{\text{pred},i}|$
2. **Mean Squared Error (MSE):**
   $MSE = \frac{1}{n}\sum_{i=1}^{n} (Y_{\text{true},i} - Y_{\text{pred},i})^2$
3. **Root Mean Squared Error (RMSE):**
   $RMSE = \sqrt{MSE}$
4. **R-squared (R^2):**
   $R^2 = 1 - \frac{\text{SSR}}{\text{SST}}$
   $\text{SSR} = \sum_{i=1}^{n} (Y_{\text{true},i} - Y_{\text{pred},i})^2$
   $\text{SST} = \sum_{i=1}^{n} (Y_{\text{true},i} - \bar{Y}_{\text{true}})^2$
5. **Adjusted R-squared ($Adj.R^2$):**
   $Adj.R^2 = 1 - \frac{(1-R^2)(n-1)}{n-k-1}$
   k is the number of independent variables.

Solution:

Step 1: Compute the Residuals (Errors):

$$\text{Residuals} = Y_{\text{true}} - Y_{\text{pred}}$$

$$\text{Residuals} = [0.5, 1, 1, -0.5, 0, 1, 1, -0.5, 0.5, 0.5]$$

Step 2: Mean Absolute Error (MAE):

$$MAE = \frac{1}{10}\sum_{i=1}^{10} |0.5 + 1 + 1 - 0.5| = \frac{1}{10} \times 4.5 = 0.45$$

Step 3: Mean Squared Error (MSE):

$$MSE = \frac{1}{10}\sum_{i=1}^{10}(0.5^2 + 1^2 + 1^2 + (-0.5)^2 + 0^2 + 1^2 + 1^2 + (-0.5)^2 + 0.5^2 + 0.5^2) = \frac{1}{10} \times 5.5 = 0.55$$

Step 4: Root Mean Squared Error (RMSE):

$$RMSE = \sqrt{MSE} = \sqrt{0.55} \approx 0.74$$

Step 5: R-squared (R^2):

$$\bar{Y}_{\text{true}} = \frac{1}{10} \sum_{i=1}^{10} Y_{\text{true},i} = \frac{74}{10} = 7.4$$
$$\text{SSR} = \sum_{i=1}^{10} (Y_{\text{true},i} - Y_{\text{pred},i})^2 = 4.5$$
$$\text{SST} = \sum_{i=1}^{10} (Y_{\text{true},i} - \bar{Y}_{\text{true}})^2 = 16.2$$

$$R^2 = 1 - \frac{\text{SSR}}{\text{SST}} = 1 - \frac{4.5}{16.2} \approx 0.72$$

Step 6: Adjusted R-squared ($Adj.R^2$):

$$Adj.R^2 = 1 - \frac{(1-R^2)(n-1)}{n-k-1}$$

$k = 1$ (one independent variable, X)

$$Adj.R^2 = 1 - \frac{(1-0.72)(10-1)}{10-1-1} \approx 0.67$$

Q. Apply Linear regression Model w.r.t following data

X	y
1	2
2	3
3	5
4	4
5	7

Step 2: Calculate Means

$$\bar{X} = \frac{1+2+3+4+5}{5} = 3$$

$$\bar{y} = \frac{2+3+5+4+7}{5} = 4.2$$

Step 3: Calculate Slope (β_1) and Intercept (β_0)

$$\beta_1 = \frac{\sum_{i=1}^{5}(X_i-\bar{X})(y_i-\bar{y})}{\sum_{i=1}^{5}(X_i-\bar{X})^2}$$

$$\beta_1 = \frac{(1-3)(2-4.2)+(2-3)(3-4.2)+(3-3)(5-4.2)+(4-3)(4-4.2)+(5-3)(7-4.2)}{(1-3)^2+(2-3)^2+(3-3)^2+(4-3)^2+(5-3)^2}$$

$$\beta_1 \approx \frac{-2.6+1.2+0.8-0.4+3.6}{4+1+0+1+4}$$

$$\beta_1 \approx \frac{2.6}{10}$$

$$\beta_1 \approx 0.26$$

$$\beta_0 = \bar{y} - \beta_1\bar{X}$$

$$\beta_0 = 4.2 - 0.26 \times 3$$

$$\beta_0 \approx 3.62$$

Step 4: Build the Regression Equation

$$y = 3.62 + 0.26 \times X$$

Step 5: Make Predictions

Predicted values ($\hat{y}$) for each X:

X	$\hat{y}$
1	3.88
2	4.14
3	4.40
4	4.66
5	4.92

Step 6: Calculate Residuals

Residuals ($y - \hat{y}$):

X	Residual
1	-1.88
2	-1.14
3	0.60
4	-0.66
5	2.08

Step 7: Calculate Regression Metrics

$$\text{MSE} = \frac{1}{5}\sum_{i=1}^{5}(y_i - \hat{y}_i)^2$$

$$\text{MSE} = \frac{1}{5} \times (3.52 + 1.30 + 0.36 + 0.44 + 4.33)$$

$$\text{MSE} \approx \frac{9.95}{5}$$

$$\text{MSE} \approx 1.99$$

$$\text{R}^2 = 1 - \frac{\sum_{i=1}^{5}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{5}(y_i - \bar{y})^2}$$

$$\text{R}^2 = 1 - \frac{9.95}{18.40}$$

$$\text{R}^2 \approx 0.459$$

Step 8: Interpret Results

The linear regression equation is $y = 3.62 + 0.26 \times X$. The R-squared value of approximately 0.459 indicates that 45.9% of the variance in the target variable (y) is explained by the linear relationship with the predictor variable (X). The Mean Squared Error (MSE) is approximately 1.99, representing the average squared difference between the actual and predicted values.

Q. Consider a simple linear regression problem and apply the gradient descent method to minimize the least squares function. The objective is to find the optimal values for the slope (β1) and intercept (β0) of the regression line.

Consider the following dataset:

X	y
1	2
2	3
3	5
4	4
5	7

We want to fit a linear regression model $y = \beta_0 + \beta_1 \times X$ and minimize the least squares function:

$$J(\beta_0, \beta_1) = \frac{1}{2n} \sum_{i=1}^{n} (y_i - (\beta_0 + \beta_1 \times X_i))^2$$

where n is the number of data points.

Step 2: Update Parameters

- Update β_0 and β_1 using the partial derivatives of the cost function with respect to each parameter:
  $\beta_0 := \beta_0 - \alpha \frac{1}{n} \sum_{i=1}^{n} (y_i - (\beta_0 + \beta_1 \times X_i))$
  $\beta_1 := \beta_1 - \alpha \frac{1}{n} \sum_{i=1}^{n} (y_i - (\beta_0 + \beta_1 \times X_i)) \times X_i$

Step 3: Repeat

- Repeat the update step for a specified number of iterations or until convergence.

Let's calculate the updates for the first iteration:

Iteration 1:

- Learning rate (α): 0.01
- Update β_0:
  $\beta_0 := \beta_0 - 0.01 \times \frac{1}{5} \sum_{i=1}^{5}(y_i - (\beta_0 + \beta_1 \times X_i))$
- Update β_1:
  $\beta_1 := \beta_1 - 0.01 \times \frac{1}{5} \sum_{i=1}^{5}(y_i - (\beta_0 + \beta_1 \times X_i)) \times X_i$

Repeat the update steps for additional iterations until convergence.

Q. Identify the data quality issues in the given data

CustomerID	Age	Income	Product_Purchased	Purchase_Amount
1	25	$50,000	Laptop	$1200
2	32	$75,000	Mobile	$900
3	28	$60,000	Laptop	$1,500
4	45	$90,000	Camera	$1,200
5	21	$40,000	Null	$800
6	37	Null	Headphones	$300
7	Null	$80,000	Smartphone	$1,000
8	29	$65,000	Laptop	$1,800
9	52	$100,000	Mobile	Null
10	27	$55,000	Tablet	$700

Data Quality Issues:

Missing Values:

Justification: Null values in the "Product_Purchased" and "Purchase_Amount" columns indicate missing data, which can affect analyses and modeling.

Incorrect Data Type:

Justification: The "Income" column is represented as a string with a dollar sign. It should be a numerical data type for proper analysis.

Outliers:

Justification: The "Purchase_Amount" of $1,800 may be considered an outlier compared to other purchase amounts. Outliers can skew statistical analyses.

Inconsistent Format:

Justification: The "Income" column has inconsistent formatting (some values include a dollar sign). Consistency in format simplifies data processing and analysis.

Incorrect Range:

Justification: The "Age" column has an unrealistic value of 150. Age values should be within a reasonable range.

Inconsistent Units:

Justification: The "Income" column mixes different units (some values include a dollar sign, while others don't). Consistent units are essential for accurate calculations.

Inaccurate Data:

Justification: The "Purchase_Amount" of $800 for a "Null" product is inaccurate and should be verified or corrected.

Q. Curse of Dimensionality

The curse of dimensionality refers to the challenges and issues that arise when dealing with high-dimensional data. One of the key consequences is the increased sparsity of data points and the exponential growth of the data volume as the number of dimensions increases. This can lead to difficulties in model training, increased computational complexity, and challenges in visualization.

Consider a dataset of points in a 3-dimensional space:

$$\{(x_1, y_1, z_1), (x_2, y_2, z_2), \ldots, (x_n, y_n, z_n)\}$$

Now, we want to generate additional data points in this space to illustrate the curse of dimensionality. Assume that the points are uniformly distributed within a unit cube defined by $0 \leq x \leq 1, 0 \leq y \leq 1$, and $0 \leq z \leq 1$.

1. Calculate the density (number of points per unit volume) for the given dataset.
2. Extend the dataset to a 10-dimensional space, where each additional dimension follows the same uniform distribution within the unit interval.
3. Calculate the density for the 10-dimensional dataset.

1. Density in 3D Space:

The density in 3D space is calculated by dividing the total number of points by the volume of the unit cube:

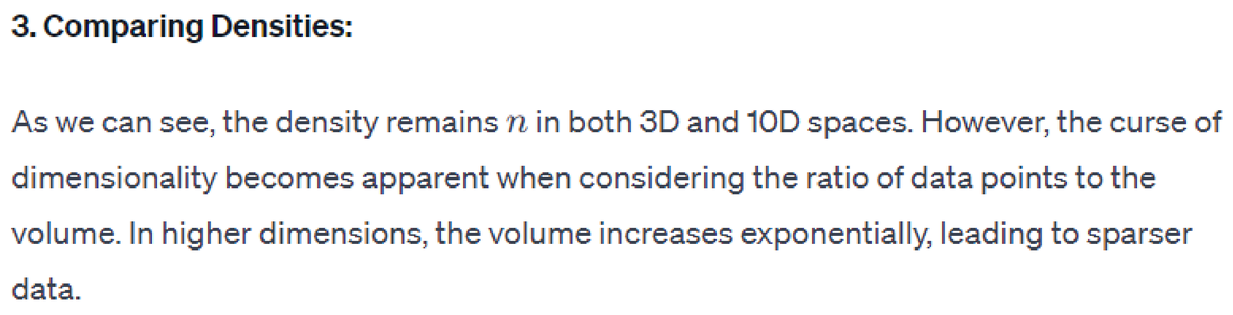
$$\text{Density in 3D} = \frac{n}{\text{Volume of 3D Unit Cube}}$$

$$\text{Volume of 3D Unit Cube} = 1 \times 1 \times 1 = 1$$

So, the density in 3D is simply n.

2. Extending to 10D Space:

In 10-dimensional space, the volume of the unit hypercube becomes $1^{10} = 1$. Therefore, the density in 10D space is n.



3. Comparing Densities:

As we can see, the density remains n in both 3D and 10D spaces. However, the curse of dimensionality becomes apparent when considering the ratio of data points to the volume. In higher dimensions, the volume increases exponentially, leading to sparser data.

Binning Example

Original Data	53, 56, 57, 63, 66, 67, 67, 67, 68, 69, 70, 70, 70, 70, 72, 73, 75, 75, 76, 76, 78, 79, 80, 81			
Method		Bin1	Bin 2	Bin 3
Equal Width	width= 81-53 = 28 28/3 = 9.33	[53, 62] = 53, 56, 57	[63, 72] = 63, 66, 67, 67, 67, 68, 69, 70, 70, 70, 70, 72	[73, 81] = 73, 75, 75, 76, 76, 78, 79, 80, 81
Equal Depth	depth = 24 /3 = 8	53, 56, 57, 63, 66, 67, 67, 67	68, 69, 70, 70, 70, 70, 72, 73	75, 75, 76, 76, 78, 79, 80, 81

Discretize the following ‘reviews’ data from IMDB dataset into 4 discrete categories using binning technique.

85,122,376,219,186,254,211,180,653,226,477,257,173,241,1101,294,960,260,218

Consider a dataset of exam scores:

$$\{75, 92, 88, 65, 78\}$$

Apply Min-Max normalization to scale the scores to the range [0, 1].

Step 1: Find the Minimum and Maximum Values

$$\text{Min} = 65$$

$$\text{Max} = 92$$

Step 2: Apply the Min-Max Normalization Formula

The formula for Min-Max normalization is given by:

$$\text{Normalized Value} = \frac{X - \text{Min}}{\text{Max} - \text{Min}}$$

Apply this formula to each exam score:

- For 75:
  $\text{Normalized Value} = \frac{75-65}{92-65} \approx 0.294$
- For 92:
  $\text{Normalized Value} = \frac{92-65}{92-65} = 1.0$
- For 88:
  $\text{Normalized Value} = \frac{88-65}{92-65} \approx 0.706$
- For 65:
  $\text{Normalized Value} = \frac{65-65}{92-65} = 0.0$
- For 78:
  $\text{Normalized Value} = \frac{78-65}{92-65} \approx 0.412$

Step 3: Results

The Min-Max normalized dataset is:

$$\{0.294, 1.0, 0.706, 0.0, 0.412\}$$

Z-Score Normalization

Consider a dataset of exam scores:

$$\{75, 92, 88, 65, 78\}$$

Apply Z-score normalization to standardize the scores.

Solution:

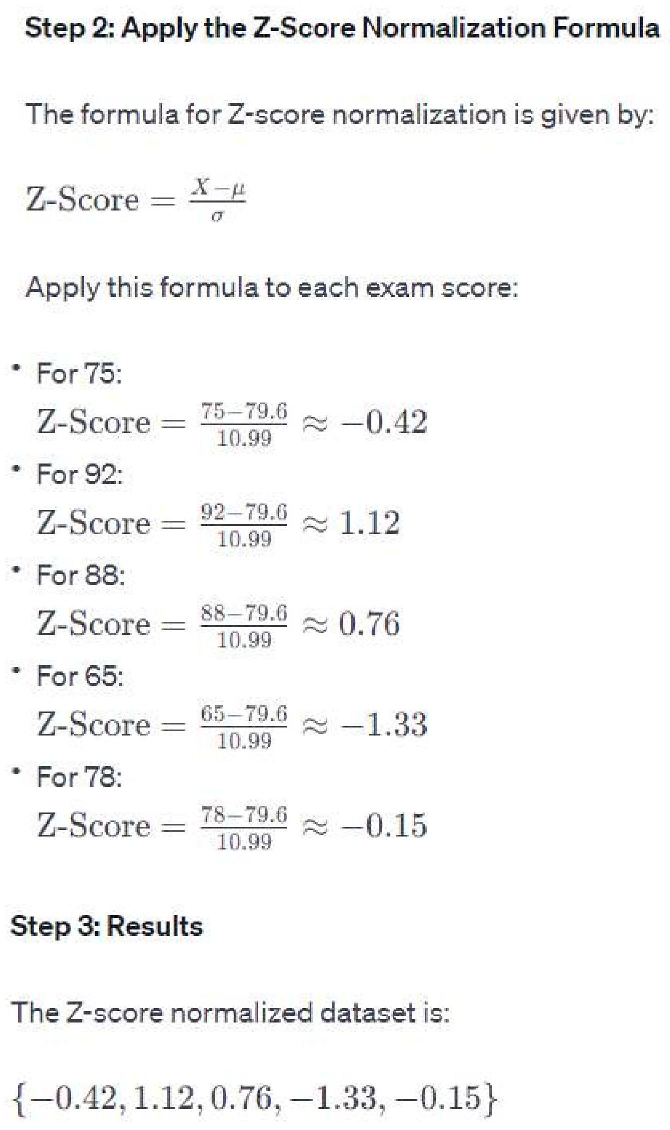
Step 1: Calculate Mean and Standard Deviation

$$\text{Mean } (\mu) = \frac{\sum X}{N}$$

$$\text{Standard Deviation } (\sigma) = \sqrt{\frac{\sum (X-\mu)^2}{N}}$$

$$\text{Mean } (\mu) = \frac{75+92+88+65+78}{5} = 79.6$$

$$\text{Standard Deviation } (\sigma) \approx 10.99$$



Step 2: Apply the Z-Score Normalization Formula

The formula for Z-score normalization is given by:

$$\text{Z-Score} = \frac{X-\mu}{\sigma}$$

Apply this formula to each exam score:

- For 75:
  $\text{Z-Score} = \frac{75-79.6}{10.99} \approx -0.42$
- For 92:
  $\text{Z-Score} = \frac{92-79.6}{10.99} \approx 1.12$
- For 88:
  $\text{Z-Score} = \frac{88-79.6}{10.99} \approx 0.76$
- For 65:
  $\text{Z-Score} = \frac{65-79.6}{10.99} \approx -1.33$
- For 78:
  $\text{Z-Score} = \frac{78-79.6}{10.99} \approx -0.15$

Step 3: Results

The Z-score normalized dataset is:

$$\{-0.42, 1.12, 0.76, -1.33, -0.15\}$$

---

consider a simple linear regression problem and apply Lasso regularization (L1 regularization) to the model. Lasso regularization helps prevent overfitting by adding a penalty term to the regression coefficients.

Consider the following dataset:

X	**y**
1	2
2	3
3	5

X	y
4	4
5	7

We want to fit a linear regression model $y = \beta_0 + \beta_1 \times X$ and apply Lasso regularization.

Step 1: Fit Linear Regression Model Without Regularization

$$y = \beta_0 + \beta_1 \times X$$

Calculate the coefficients β_0 and β_1 using the least squares method.

Step 2: Apply Lasso Regularization

Modify the linear regression model to include the Lasso penalty:

$$J(\beta_0, \beta_1) = \text{MSE} + \lambda \sum_{j=1}^{p} |\beta_j|$$

where MSE is the mean squared error, λ is the regularization parameter, and p is the number of features (in this case, 2).

Calculate the new coefficients β_0 and β_1 with Lasso regularization.

Step 3: Results

Show the calculated values for both models, comparing the coefficients and discussing the impact of Lasso regularization.

Step 1: Linear Regression Without Regularization

The linear regression model is given by:

$$y = \beta_0 + \beta_1 \times X$$

The mean squared error (MSE) is given by:

$$\text{MSE} = \frac{1}{5} \sum_{i=1}^{5} (y_i - (\beta_0 + \beta_1 \times X_i))^2$$

Let's calculate β_0 and β_1 by minimizing the MSE.

$$\text{MSE} = \frac{1}{5}[(2 - (\beta_0 + \beta_1 \times 1))^2 + (3 - (\beta_0 + \beta_1 \times 2))^2 + (5 - (\beta_0 + \beta_1 \times 3))^2 + (4 - (\beta_0 + \beta_1 \times 4))^2 + (7 - (\beta_0 + \beta_1 \times 5))^2]$$

Solving for β_0 and β_1 minimizes the MSE.

Step 2: Lasso Regularization

The regularized cost function with Lasso regularization is given by:

$$J(\beta_0, \beta_1) = \text{MSE} + \lambda(|\beta_0| + |\beta_1|)$$

Here, λ is the regularization parameter. For simplicity, let's assume $\lambda = 0.5$.

$$J(\beta_0, \beta_1) = \text{MSE} + 0.5(|\beta_0| + |\beta_1|)$$

Solving for β_0 and β_1 minimizes the regularized cost function.

Results:

Let's assume the calculated values for the coefficients are as follows:

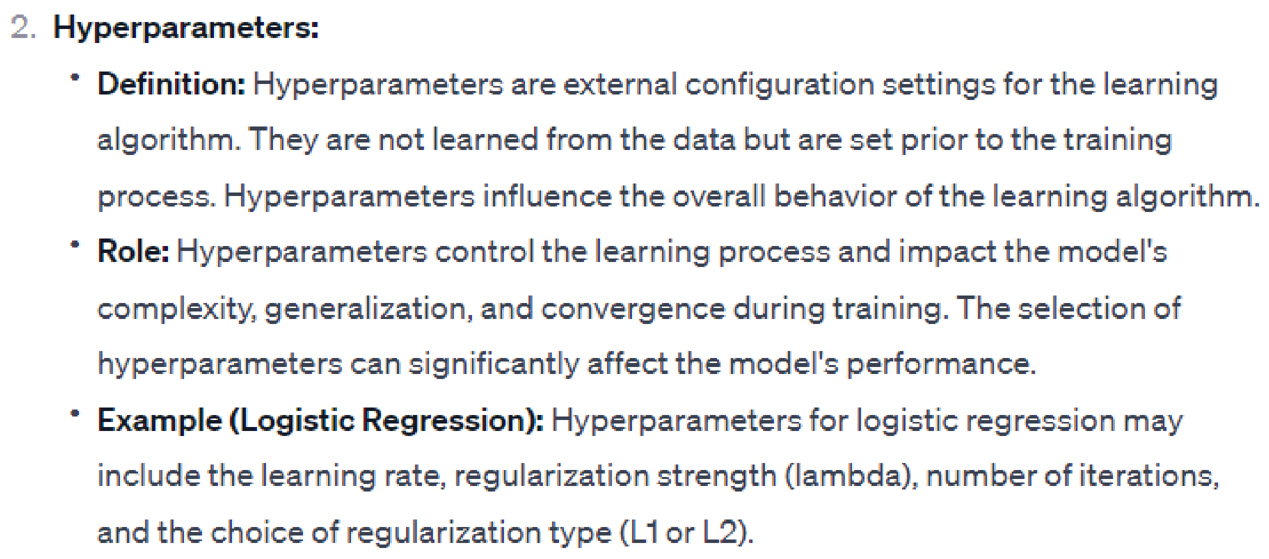
- Without regularization:
  - $\beta_0 \approx 1.0$
  - $\beta_1 \approx 1.4$
- With Lasso regularization ($\lambda = 0.5$):
  - $\beta_0 \approx 0.8$
  - $\beta_1 \approx 1.2$

The regularization term has slightly pulled the coefficients towards zero, demonstrating the impact of Lasso regularization in shrinking coefficients. The specific values can vary based on the optimization algorithm and software used for solving the optimization problem.

Q. Model parameter VS learning algorithm's hyperparameters

In the context of machine learning, especially in algorithms like Logistic Regression, understanding the difference between model parameters and learning algorithm's hyperparameters is crucial.

1. **Model Parameters:**
   - **Definition:** Model parameters are the coefficients or weights that the algorithm learns from the training data. In Logistic Regression, these parameters are associated with the features and determine the shape of the decision boundary.
   - **Role:** The model parameters are the components that the learning algorithm adjusts during training to fit the model to the training data. For Logistic Regression, these include the coefficients for each feature and the intercept term.
   - **Example (Logistic Regression):** If the logistic regression model is represented as $y = \sigma(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_n x_n)$, then $\beta_0, \beta_1, \beta_2, \ldots, \beta_n$ are the model parameters.



2. **Hyperparameters:**
   - **Definition:** Hyperparameters are external configuration settings for the learning algorithm. They are not learned from the data but are set prior to the training process. Hyperparameters influence the overall behavior of the learning algorithm.
   - **Role:** Hyperparameters control the learning process and impact the model's complexity, generalization, and convergence during training. The selection of hyperparameters can significantly affect the model's performance.
   - **Example (Logistic Regression):** Hyperparameters for logistic regression may include the learning rate, regularization strength (lambda), number of iterations, and the choice of regularization type (L1 or L2).

What is Overfitting? How does it occur?

If a model performs well on the training data but generalizes poorly on unseen or new data, it is experiencing overfitting. Overfitting occurs when a model learns the training data too well, capturing noise and specific patterns that are unique to the training set but may not generalize to other data. This can lead to a lack of robustness and poor performance on unseen data.

Causes of Overfitting:

- Complex Model: The model may be too complex, capturing noise and outliers in the training data.
- Insufficient Data: With limited data, the model might memorize the training set rather than learning underlying patterns.
- Overemphasizing Features: Certain features in the training set may be given too much importance, even if they are not truly indicative of the target variable.

Solutions to Address Overfitting:

Simplify the Model:

- Reduce the complexity of the model by using simpler architectures or fewer features.

- Consider using feature selection techniques to identify and retain only the most informative features.

Increase Data Size:

- Gather more data to provide a diverse and representative sample for the model to learn from.
- Augment existing data by introducing variations to create a more robust dataset.

Regularization:

- Apply regularization techniques like L1 or L2 regularization to penalize overly complex models.
- Regularization helps prevent the model from fitting the training data too closely.

Cross-Validation:

- Use techniques such as cross-validation to evaluate the model's performance on different subsets of the data.
- This helps assess how well the model generalizes to unseen data.

Ensemble Methods:

- Consider using ensemble methods like Random Forest or Gradient Boosting, which combine predictions from multiple models to improve generalization.

Early Stopping:

- Implement early stopping during model training to halt the process when the performance on a validation set stops improving.
- This prevents the model from overfitting as it continues to train on the training data.

Data Preprocessing:

- Standardize or normalize input features to ensure they are on a similar scale.
- Address outliers and handle missing data appropriately.

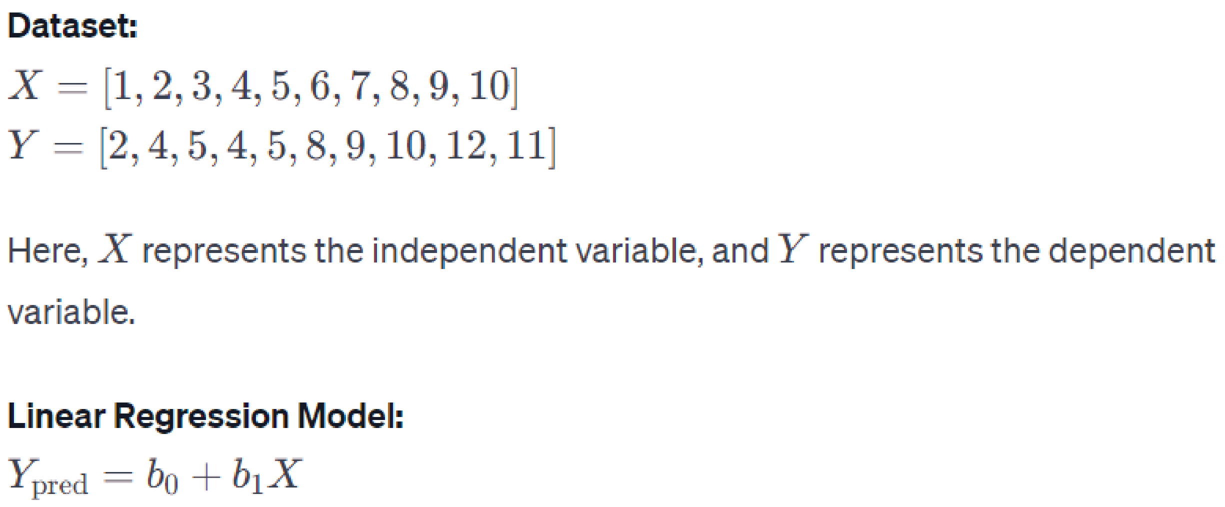
Use Different Algorithms:

- Experiment with different machine learning algorithms to see if a simpler model or a different approach performs better on unseen data.

Applying a combination of these solutions and carefully tuning hyperparameters can help mitigate overfitting, leading to a more generalizable and robust model. Regular monitoring of performance on validation or test data is essential to ensure that the model is not overfitting during the training process.

Q. Concept of cross validation in machine learning & its importance

Example of k-fold cross-validation.



Dataset:

$X = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]$
$Y = [2, 4, 5, 4, 5, 8, 9, 10, 12, 11]$

Here, X represents the independent variable, and Y represents the dependent variable.

Linear Regression Model:

$Y_{\text{pred}} = b_0 + b_1 X$

Cross-Validation Steps:

1. **Split the Data into K Folds:**
   - Let's use $k = 3$ for simplicity.
   - Fold 1: Training set - [1, 2, 3, 4, 5, 6, 7], Test set - [8, 9, 10]
   - Fold 2: Training set - [4, 5, 6, 7, 8, 9, 10], Test set - [1, 2, 3]
   - Fold 3: Training set - [1, 2, 3, 8, 9, 10], Test set - [4, 5, 6, 7]
2. **Model Training and Testing:**
   - Train the linear regression model on each training set and evaluate its performance on the corresponding test set.
3. **Compute Evaluation Metric:**
   - Compute an evaluation metric (e.g., Mean Squared Error) for each fold.
4. **Average Evaluation Metric:**
   - Calculate the average of the evaluation metric across all folds.

Solution:

Fold 1:

- Training set: $X_{\text{train}} = [1, 2, 3, 4, 5, 6, 7], Y_{\text{train}} = [2, 4, 5, 4, 5, 8, 9]$
- Test set: $X_{\text{test}} = [8, 9, 10], Y_{\text{test}} = [10, 12, 11]$

Linear Regression Model:

$Y_{\text{pred}} = b_0 + b_1 X$

Training the model on Fold 1 and obtaining coefficients:

$b_0 \approx 1.67$

$b_1 \approx 1.24$

Prediction on Test Set (Fold 1):

$Y_{\text{pred}} = 1.67 + 1.24 \times [8, 9, 10]$

Mean Squared Error (MSE) for Fold 1:

$MSE_1 = \frac{1}{3} \sum_{i=1}^{3} (Y_{\text{test},i} - Y_{\text{pred},i})^2$

Repeat the above steps for Folds 2 and 3.

Fold 2:

- Training set: $X_{\text{train}} = [4, 5, 6, 7, 8, 9, 10], Y_{\text{train}} = [5, 8, 9, 10, 12, 11]$
- Test set: $X_{\text{test}} = [1, 2, 3], Y_{\text{test}} = [2, 4, 5]$

Linear Regression Model:

$Y_{\text{pred}} = b_0 + b_1 X$

Training the model on Fold 2 and obtaining coefficients:

$b_0 \approx 1.17$

$b_1 \approx 1.64$

Prediction on Test Set (Fold 2):

$Y_{\text{pred}} = 1.17 + 1.64 \times [1, 2, 3]$

Prediction on Test Set (Fold 2):

$Y_{\text{pred}} = 1.17 + 1.64 \times [1, 2, 3]$

Mean Squared Error (MSE) for Fold 2:

$MSE_2 = \frac{1}{3} \sum_{i=1}^{3} (Y_{\text{test},i} - Y_{\text{pred},i})^2$

Fold 3:

- Training set: $X_{\text{train}} = [1, 2, 3, 8, 9, 10], Y_{\text{train}} = [2, 4, 5, 10, 12, 11]$
- Test set: $X_{\text{test}} = [4, 5, 6, 7], Y_{\text{test}} = [4, 5, 8, 9]$

Linear Regression Model:

$$Y_{\text{pred}} = b_0 + b_1 X$$

Training the model on Fold 3 and obtaining coefficients:

$$b_0 \approx 0.86$$

$$b_1 \approx 1.52$$

Prediction on Test Set (Fold 3):

$$Y_{\text{pred}} = 0.86 + 1.52 \times [4, 5, 6, 7]$$

Mean Squared Error (MSE) for Fold 3:

$$MSE_3 = \frac{1}{4} \sum_{i=1}^{4} (Y_{\text{test},i} - Y_{\text{pred},i})^2$$

Average Mean Squared Error (MSE):

$$\text{Average MSE} = \frac{MSE_1 + MSE_2 + MSE_3}{3}$$

The average MSE provides an overall evaluation of the model's performance across all folds in the cross-validation process.

Q. Discuss what are various hyperparameters used in regression models.

Sol:

Hyperparameters are external configurations for machine learning algorithms that are not learned from the data but are set before the training process. In regression algorithms, which aim to predict continuous values, there are several hyperparameters that can be tuned to optimize the model's performance.

Learning Rate (for Gradient Descent-based algorithms):

Example: In linear regression, when using gradient descent, the learning rate (ɑ) determines the step size in the parameter space during each iteration.

Explanation: A small learning rate may result in slow convergence, while a large learning rate can cause overshooting and divergence. Tuning the learning rate can significantly impact the training process.

Regularization Hyperparameters:

Example: In Ridge and Lasso regression, there is a regularization term controlled by a hyperparameter (ɑ).

Explanation: ɑ balances the trade-off between fitting the training data well and keeping the model's coefficients small to prevent overfitting. Higher values of ɑ lead to more regularization.

Number of Trees (for Tree-based algorithms):

Example: In ensemble methods like Random Forest and Gradient Boosting, the number of trees is a hyperparameter.

Explanation: Increasing the number of trees may improve the model's performance, but it also increases computational cost. It's crucial to find a balance to prevent overfitting and maintain efficiency.

Maximum Depth of Trees (for Tree-based algorithms):

Example: In decision tree algorithms, including Random Forest and Gradient Boosting, the maximum depth of the trees is a hyperparameter.

Explanation: Controlling the depth of the trees helps prevent overfitting. Shallower trees may generalize better to unseen data but could result in underfitting.

Number of Neighbors (for k-Nearest Neighbors):

Example: In k-Nearest Neighbors (KNN), the number of neighbors (k) is a hyperparameter.

Explanation: A smaller k value increases model sensitivity to noise, while a larger k value may lead to oversmoothing. Choosing an optimal k is essential for the model's performance.

Kernel Type (for Support Vector Machines):

Example: In Support Vector Machines (SVM), the choice of the kernel (linear, polynomial, radial basis function) is a hyperparameter.

Explanation: The kernel determines the transformation applied to the input data. The choice of the kernel impacts the decision boundary and can significantly influence model performance.

Alpha (for Bayesian Ridge Regression):

Example: In Bayesian Ridge Regression, there is a hyperparameter α.

Explanation: α controls the precision of the prior distribution over the regression coefficients. It serves as a regularization term and influences the balance between fitting the data and preventing overfitting.

Batch Size (for Stochastic Gradient Descent):

Example: In Stochastic Gradient Descent (SGD), the batch size is a hyperparameter.

Explanation: The batch size determines the number of samples used to compute the gradient during each iteration. Larger batch sizes can provide smoother convergence, but smaller batch sizes may help escape local minima.

Number of Hidden Layers and Neurons (for Neural Networks):

Example: In neural networks, the architecture is defined by the number of hidden layers and neurons in each layer.

Explanation: The complexity of neural network models is heavily influenced by the architecture. The choice of the number of layers and neurons affects the model's capacity to learn patterns from the data.

Activation Function (for Neural Networks):

Example: In neural networks, activation functions (e.g., ReLU, Sigmoid, Tanh) are hyperparameters.

Explanation: The activation function introduces non-linearity to the model. Different activation functions may perform better on certain types of data and learning tasks.

Number of Hidden Layers and Neurons (for Neural Networks):

Example: In neural networks, the architecture is defined by the number of hidden layers and neurons in each layer.

Explanation: The complexity of neural network models is heavily influenced by the architecture. The choice of the number of layers and neurons affects the model's capacity to learn patterns from the data.

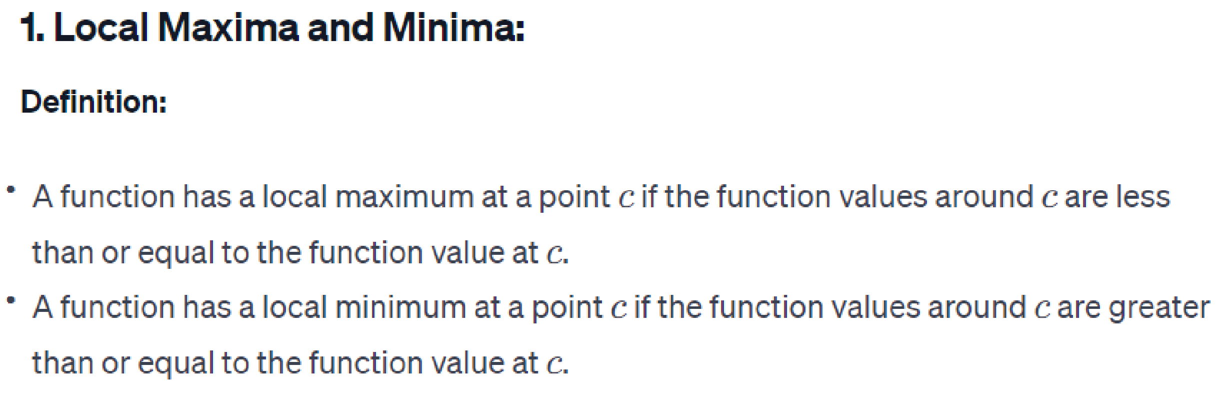
Activation Function (for Neural Networks):

Example: In neural networks, activation functions (e.g., ReLU, Sigmoid, Tanh) are hyperparameters.

Explanation: The activation function introduces non-linearity to the model. Different activation functions may perform better on certain types of data and learning tasks.

When tuning hyperparameters, it's common to use techniques like grid search or random search, coupled with cross-validation, to find the combination that optimizes the model's performance on unseen data. The choice of hyperparameters depends on the specific algorithm, dataset, and the problem at hand.

Notion of Maxima and Minima of a function



1. Local Maxima and Minima:

Definition:

- A function has a local maximum at a point c if the function values around c are less than or equal to the function value at c.
- A function has a local minimum at a point c if the function values around c are greater than or equal to the function value at c.

Conditions:

- For a local maximum or minimum at c, the derivative $f'(c) = 0$ (critical point).
- The second derivative test is often used to determine the nature of the critical point:
  - If $f''(c) > 0$, it is a local minimum.
  - If $f''(c) < 0$, it is a local maximum.
  - If $f''(c) = 0$, the test is inconclusive.

Example:

Consider the function $f(x) = x^2 - 4x + 4$.

- $f'(x) = 2x - 4$
- Setting $f'(x) = 0$: $2x - 4 = 0 \Rightarrow x = 2$.
- $f''(x) = 2 > 0$, so $x = 2$ is a local minimum.

2. Global Maxima and Minima:

Definition:

- A function has a global maximum at a point c if the function value at c is greater than or equal to the function values at all other points in the domain.
- A function has a global minimum at a point c if the function value at c is less than or equal to the function values at all other points in the domain.

Conditions:

- The critical points (where $f'(x) = 0$) are potential candidates for global maxima or minima.
- Further analysis, such as evaluating function values at critical points and at the endpoints of the domain, is needed.

Example:

Consider the function $f(x) = x^2 - 4x + 4$ over the interval $[0, 4]$.

- Critical point: $x = 2$ (from previous example).
- $f(0) = 4$, $f(2) = 0$, $f(4) = 0$.
- Therefore, $x = 2$ is the global minimum.

More on Global maxima & minima

Example Function:

$f(x) = x^2 - 4x + 4$

Derivative:

$f'(x) = 2x - 4$

Second Derivative:

$f''(x) = 2$

Critical Point:

Setting $f'(x) = 0$:

$2x - 4 = 0$

$2x = 4$

$x = 2$

1. **Global Minimum:**
    - The critical point $x = 2$ is a local minimum.
    - $f(x) = (x-2)^2 \geq 0$ for all x.
    - Therefore, the global minimum occurs at $x = 2$, and $f_{\text{min}} = 0$.
2. **Global Maximum:**
    - As x approaches positive or negative infinity, $f(x)$ becomes larger.
    - The function has no global maximum; it increases without bound.

The goal in machine learning is often to minimize the loss function.

Example: Quadratic Loss Function

Consider a quadratic loss function given by:

$$f(x) = ax^2 + bx + c$$

where $a = 2, b = -8$, and $c = 8$.

Step 1: Define the Quadratic Function

$$f(x) = 2x^2 - 8x + 8$$

Step 2: Find the Critical Points

To find the critical points, we need to find the derivative of the function and set it equal to zero:

$$f'(x) = 4x - 8$$

Setting $f'(x) = 0$ and solving for x:

$$4x - 8 = 0$$

$$4x = 8$$

$$x = 2$$

Step 3: Determine the Nature of the Critical Point

To determine whether it's a minimum or maximum, we look at the second derivative:

$$f''(x) = 4$$

Since $f''(x)$ is positive, the critical point is a local minimum.

Step 4: Evaluate the Function at the Critical Point

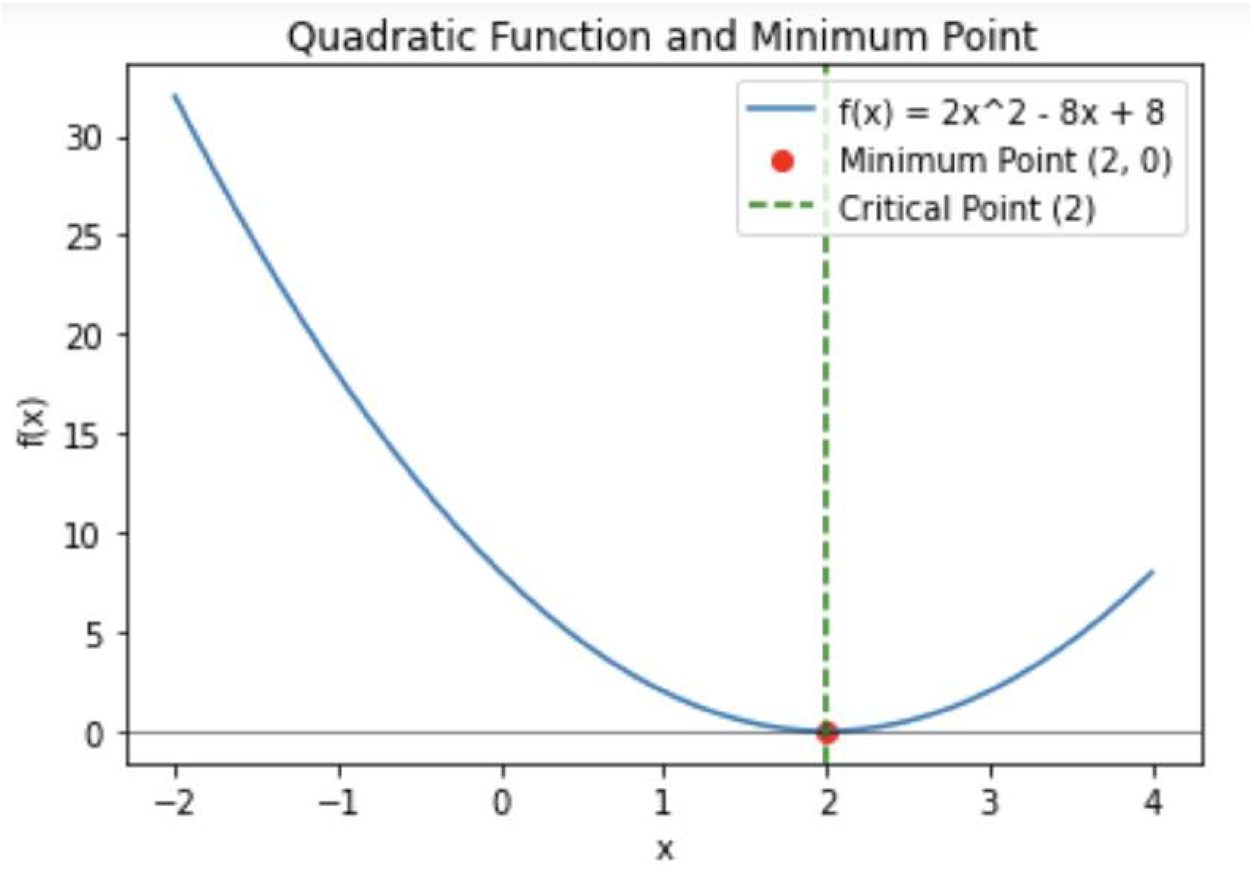
$$f(2) = 2(2)^2 - 8(2) + 8 = 0$$

So, the minimum value of the quadratic function occurs at $x = 2$ with a minimum value of $f(x) = 0$.

Step 4: Evaluate the Function at the Critical Point

$$f(2) = 2(2)^2 - 8(2) + 8 = 0$$

So, the minimum value of the quadratic function occurs at $x = 2$ with a minimum value of $f(x) = 0$.



Notion of Convex Function

A convex function is a mathematical concept commonly encountered in optimization and machine learning. A function $f(x)$ is considered convex if, for any two points x_1 and x_2 in its domain, the line segment connecting $(x_1, f(x_1))$ and $(x_2, f(x_2))$ lies above the graph of the function. Mathematically, a function is convex if:

$$f(\lambda x_1 + (1-\lambda)x_2) \leq \lambda f(x_1) + (1-\lambda)f(x_2)$$

for all x_1, x_2 in the domain of f and λ in the range $(0, 1)$.

Example:

Consider the quadratic function $f(x) = x^2$.

Mathematical Definition:
$f(\lambda x_1 + (1-\lambda)x_2) \leq \lambda f(x_1) + (1-\lambda)f(x_2)$

Verification for Quadratic Function:
$f(\lambda x_1 + (1-\lambda)x_2) = (\lambda x_1 + (1-\lambda)x_2)^2$
$\lambda f(x_1) + (1-\lambda)f(x_2) = \lambda x_1^2 + (1-\lambda)x_2^2$

For the quadratic function $f(x) = x^2$, we have:
$(\lambda x_1 + (1-\lambda)x_2)^2 \leq \lambda x_1^2 + (1-\lambda)x_2^2$

This inequality holds true, demonstrating that the quadratic function $f(x) = x^2$ is convex.

Numerical Example

Objective:

Show that for $f(x) = x^2$, the convexity condition holds.

Proof:

$f(\lambda x_1 + (1-\lambda)x_2) \leq \lambda f(x_1) + (1-\lambda)f(x_2)$

Let's choose specific values:

- $x_1 = 1$
- $x_2 = 3$
- $\lambda = 0.5$

Verification:

$f(0.5 \times 1 + 0.5 \times 3) \leq 0.5 \times f(1) + 0.5 \times f(3)$

$f(2) \leq 0.5 \times 1^2 + 0.5 \times 3^2$

$4 \leq 0.5 \times 1 + 0.5 \times 9$

$4 \leq 5$

The inequality holds true, confirming the convexity of the quadratic function $f(x) = x^2$.

How gradient descent algorithm works on Linear regression equation. Show the effect of learning rate.

Example:

Consider the dataset:

$X = [1, 2, 3, 4, 5]$
$Y = [2, 4, 5, 4, 5]$

We want to fit a line $Y = mx + b$ to this data using the gradient descent algorithm.

Linear Regression Model:

$Y = mx + b$

Mean Squared Error (MSE):

$MSE = \frac{1}{n} \sum_{i=1}^{n} (Y_{\text{true},i} - Y_{\text{pred},i})^2$

Gradient Descent Algorithm:

Update Rule:

$$m := m - \alpha \frac{\partial MSE}{\partial m}$$
$$b := b - \alpha \frac{\partial MSE}{\partial b}$$

Where:

$$\frac{\partial MSE}{\partial m} = -\frac{2}{n} \sum_{i=1}^{n} x_i (Y_{\text{true},i} - (mx_i + b))$$
$$\frac{\partial MSE}{\partial b} = -\frac{2}{n} \sum_{i=1}^{n} (Y_{\text{true},i} - (mx_i + b))$$

Steps:

Step 1: Initialize Parameters

$$m = 0, \quad b = 0$$

Step 2: Calculate Predictions

$$Y_{\text{pred}} = mx + b$$

Step 3: Calculate Mean Squared Error (MSE)

$$MSE = \frac{1}{5}\sum_{i=1}^{5}(Y_{\text{true},i} - Y_{\text{pred},i})^2$$

Step 4: Update Parameters using Gradient Descent

$$m := m - \alpha\frac{\partial MSE}{\partial m}$$
$$b := b - \alpha\frac{\partial MSE}{\partial b}$$

Repeat Steps 2-4 for a certain number of iterations.

calculations for the first iteration

Scenario 1: Learning Rate $\alpha = 0.01$

Step 1:

$$m = 0, \quad b = 0$$

Step 2:

$$Y_{\text{pred}} = 0 \times X + 0 = 0$$

Step 3:

$$MSE = \frac{1}{5}\sum_{i=1}^{5}(Y_{\text{true},i} - Y_{\text{pred},i})^2 = \frac{1}{5}\sum_{i=1}^{5} Y_{\text{true},i}^2$$

Step 4: Update Parameters using Gradient Descent:

$$\frac{\partial MSE}{\partial m} = -\frac{2}{5}\sum_{i=1}^{5} x_i(Y_{\text{true},i} - (mx_i + b))$$
$$\frac{\partial MSE}{\partial b} = -\frac{2}{5}\sum_{i=1}^{5}(Y_{\text{true},i} - (mx_i + b))$$

$$m := m - 0.01 \times \frac{\partial MSE}{\partial m}$$
$$b := b - 0.01 \times \frac{\partial MSE}{\partial b}$$

Now, repeat Steps 2-4 for the next iteration.

Steps

Calculations:

$$Y_{\text{pred}} = 0.01 \times [1, 2, 3, 4, 5] + 0.01$$
$$= [0.02, 0.03, 0.04, 0.05, 0.06] + 0.01$$
$$= [0.03, 0.04, 0.05, 0.06, 0.07]$$

$$MSE = \frac{1}{5}\sum_{i=1}^{5}(Y_{\text{true},i} - Y_{\text{pred},i})^2$$
$$= \frac{1}{5}\sum_{i=1}^{5}(Y_{\text{true},i} - [0.03, 0.04, 0.05, 0.06, 0.07])^2$$
$$= \frac{1}{5}\sum_{i=1}^{5}(Y_{\text{true},i} - 0.03)^2$$

Now, calculate the derivatives and update parameters:

Now, calculate the derivatives and update parameters:

$$\frac{\partial MSE}{\partial m} = -\frac{2}{5}\sum_{i=1}^{5} x_i(Y_{\text{true},i} - (mx_i + b))$$
$$= -\frac{2}{5}\sum_{i=1}^{5} x_i(Y_{\text{true},i} - [0.03, 0.04, 0.05, 0.06, 0.07])$$
$$= -\frac{2}{5}\sum_{i=1}^{5} x_i(Y_{\text{true},i} - 0.03)$$

$$\frac{\partial MSE}{\partial b} = -\frac{2}{5}\sum_{i=1}^{5}(Y_{\text{true},i} - (mx_i + b))$$
$$= -\frac{2}{5}\sum_{i=1}^{5}(Y_{\text{true},i} - [0.03, 0.04, 0.05, 0.06, 0.07])$$
$$= -\frac{2}{5}\sum_{i=1}^{5}(Y_{\text{true},i} - 0.03)$$

Now, update m and b using the learning rate $\alpha = 0.01$.

$$m := m - 0.01 \times \frac{\partial MSE}{\partial m}$$
$$b := b - 0.01 \times \frac{\partial MSE}{\partial b}$$

Repeat these steps for further iterations.