studocu

Unit 2 ML Notes ml r22

Machine Learning (Jawaharlal Nehru Technological University, Hyderabad)

Scan to open on Studocu

Studocu is not sponsored or endorsed by any college or university
Downloaded by Vaishnavi Bhan (vaishnavibhan8157@gmail.com)

UNIT-II

Multi-Layer Perceptron (MLP), Going Forwards,Going Backwards,Backpropagation Error,Multi-Layer Perceptron (MLP) in Practice,Examples of using the Multi-LayerPerceptron (MLP)-overview, Deriving Back Propagation, Radial Basis Functions and Splines, Concepts -- RBF (Radial Basis Function) Network, Curse of Dimensionality, Interpretations and Basis Functions, Support Vector Machines (SVM)

Type-1: Multi-Layer Perceptron (MLP), Going Forwards, Going Backwards:

Multi-layer perceptron (MLP)

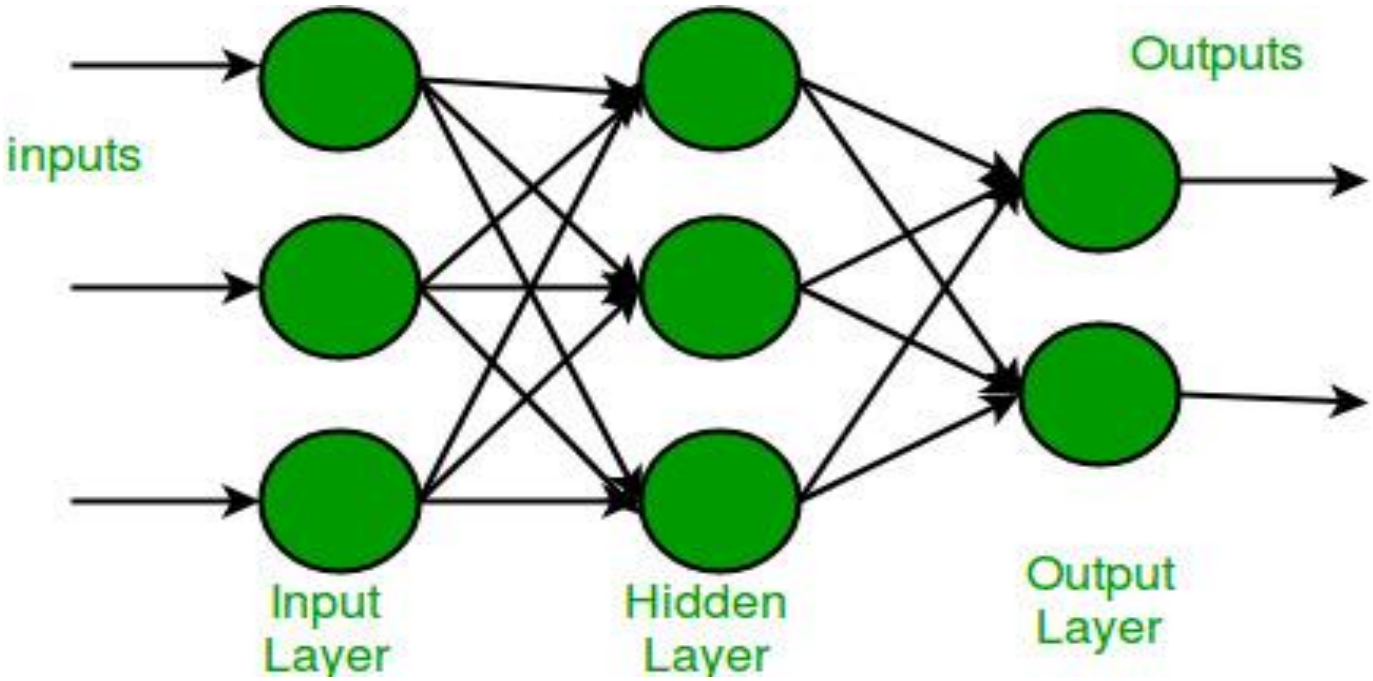
A multi-layer perceptron (MLP) is a neural network that has at least three layers: an input layer, an hidden layer and an output layer. Each layer operates on the outputs of its preceding layer:
A multilayer perceptron (MLP) is a perceptron that teams up with additional perceptron, stacked in several layers, to solve complex problems. Each perceptron in the first layer on the left (the input layer), sends outputs to all the perceptron in the second layer (the hidden layer), and all perceptron in the second layer send outputs to the final layer on the right (the output layer).

Multi-Layer Perceptron Architecture:

It is a type of neural network with an architecture consisting of input, hidden, and output layers of interconnected neurons. It is capable of learning complex patterns and performing tasks such as classification and regression by adjusting its parameters through training.

Let’s explore the architecture of an MLP in detail:

This document is available on studocu
Downloaded by Vaishnavi Bhan (vaishnavibhan8157@gmail.com)



Input Layer: The input layer is where the MLP and dataset first engage with one another. A feature in the incoming data is matched to each neuron in this layer. For instance, each neuron might represent the intensity value of a pixel in picture categorization. These unprocessed input values are to be distributed to the neurons in the next hidden layers by the input layer.

Hidden Layers: MLPs have a hidden layer or layers that are present between the input and output layers. The main computations happen at these layers. Every neuron in a hidden layer analyzes the data that comes from the neurons in the layer above it. In the same buried layer, neurons do not interact directly with one another but rather indirectly via weighted connections. The hidden layer transformation allows the network to learn intricate links and representations in the data. The intricacy of the task might affect the depth (number of hidden layers) and width (number of neurons in each layer).

Output Layer: The MLP's neurons in the output layer, the last layer, generate the model's predictions. The structure of this layer is determined by the particular task at hand. The probability score for binary classification may be generated by a single neuron with a sigmoid activation function. Multiple neurons, often with softmax activation, can give probabilities to each class in a multi-class classification system. When doing regression tasks, the output layer frequently just has a single neuron that can forecast a continuous value.

Each neuron applies an activation function to the weighted total of its inputs, whether it is in the input, hidden, or output layer. The sigmoid, hyperbolic tangent (tanh), and rectified linear unit (ReLU) are often used activation functions. The MLP modifies connection (synapse) weights during training using backpropagation and optimization methods like gradient descent. In order to reduce the discrepancy between projected and actual outputs, this method aids the network in learning and fine-tuning its parameters. MLPs are appropriate for a variety of machine learning and deep learning problems, from straightforward to extremely complicated, due to their flexibility in terms of the number of hidden layers, neurons per layer, and choice of activation functions.

In a **Multi-Layer Perceptron (MLP)**, the concepts of "going forward" and "going backward" refer to the two key processes involved in training the network: **forward propagation** and **backpropagation**.

Downloaded by Vaishnavi Bhan (vaishnavibhan8157@gmail.com)

1. Forward Propagation (Going Forward):

It refers to the process in artificial neural networks where inputs are passed through the network layer by layer to generate a prediction or output. This process is called "going forward" because the data flows from the input layer through the hidden layers and eventually to the output layer.

Working procedure:

1. **Input Layer**: The input data (features) is fed into the network. Each input neuron represents one feature from the dataset.
2. **Weights and Biases**: Each connection between neurons has an associated weight, and each neuron has an associated bias. These weights and biases determine the importance of the connections and neurons.
3. **Activation Function**: For each neuron in the hidden layers and the output layer, the input data is multiplied by weights, summed with the bias, and then passed through an activation function. This activation function introduces non-linearity into the model, allowing the neural network to learn complex patterns.
4. **Layer-wise Computation**: The output from one layer becomes the input to the next layer. This process is repeated until the data reaches the output layer.
5. **Output Layer**: The final layer produces the predicted output (e.g., classification, regression values).

The steps in forward propagation can be expressed mathematically as follows:

For a single neuron $z=W \cdot X+b$

W = weights vector

X = input vector

b = bias

z is passed through an activation function $a=f(z)$

The process ends with the prediction being compared to the actual target in the training phase, which is used to calculate the error.

2. Backpropagation (Going Backward):

Backpropagation is the process used in training neural networks to adjust the weights and biases in order to minimize the error (or loss) between the predicted output and the actual output.This process is called "going backward " because the data flows from the output layer to the input layer:

Steps of Backpropagation:

a) **Error Calculation (Loss Function)**:

This document is available on studocu
Downloaded by Vaishnavi Bhan (vaishnavibhan8157@gmail.com)

- After forward propagation, the network produces a prediction. The difference between the predicted value and the actual value is measured using a **loss function** (e.g., Mean Squared Error for regression or Cross-Entropy for classification).
- The loss L represents how far the network's prediction is from the ground truth.

b) **Compute the Gradient of the Loss**:

- The goal of backpropagation is to adjust the weights and biases in such a way that the loss is minimized. This is done by computing the **gradients** (partial derivatives) of the loss function with respect to each weight and bias in the network.
- Gradients tell us the direction and magnitude in which the weights need to be updated to reduce the loss.

c)**Chain Rule of Calculus**:

- Backpropagation uses the **chain rule** from calculus to efficiently compute these gradients. The chain rule allows us to compute the derivative of the loss with respect to weights by breaking down the process layer by layer.
- For each layer, we compute how much the current layer's weights and biases contributed to the error and then propagate that error backward through the network, adjusting each layer accordingly.

d)**Weight and Bias Update**:

- Once the gradients are calculated, **Gradient Descent** (or a variant like Stochastic Gradient Descent or Adam) is used to update the weights and biases in the network.
- The update rule for a weight W is

  $W_{new} = W_{old} - \eta \partial L / \partial W$

  Where

  η is the learning rate (a hyperparameter that controls the size of the update step),

  $\partial L / \partial W$ is the gradient of the loss with respect to the weight.

- Similarly, biases are updated:

  $b_{new} = b_{old} - \eta \partial L / \partial b$

e) **Repeat**:

- This process is repeated for each batch of data in the training set. Over time, the weights and biases converge toward values that minimize the error.

Breakdown by Layer:

- **Output Layer**: The error is first calculated at the output layer, based on the difference between the predicted output and the actual target. The gradient of this error is calculated and propagated backward.

Downloaded by Vaishnavi Bhan (vaishnavibhan8157@gmail.com)

- **Hidden Layers**: The error is propagated backward through the hidden layers. For each hidden layer, the gradients are calculated with respect to the weights and biases of that layer using the chain rule. This process continues until the input layer is reached.

Type-2: Backpropagation Error:

Backpropagation error refers to the process of calculating and distributing the error or loss through a neural network during training. The goal is to adjust the weights in such a way that the output becomes more accurate, reducing the overall error. This process is crucial for optimizing the performance of the network.

Working procedure:

1. **Initial Forward Pass**:
   - The input is passed through the network in a process called forward propagation.
   - The network produces an output (a prediction), which is compared to the actual target value using a loss function (such as Mean Squared Error for regression or Cross-Entropy Loss for classification).
   - This loss is a measure of the error between the predicted and true output.
2. **Error Calculation**:
   - The difference between the predicted output and the true output (label) represents the **error** or **loss**.
   - The goal of backpropagation is to minimize this loss.
3. **Backward Pass (Backpropagation)**:
   - The backpropagation algorithm calculates the **gradient** of the loss function with respect to each weight in the network using the **chain rule** of calculus.
   - This gradient tells us how much the loss would change if a specific weight were adjusted slightly. It gives the **direction** in which the weights should be adjusted to reduce the loss.
4. **Error Propagation (Layer by Layer)**:
   - Backpropagation starts at the output layer (where the final prediction is made) and works backward through the hidden layers.
   - In each layer, the error is **propagated backward** using the chain rule, where the gradient of the loss with respect to the output of the layer is computed. This is used to compute the gradient of the loss with respect to the inputs to that layer (which are the outputs of the previous layer).
   - This continues until the error reaches the input layer.
5. **Weight Update**:
   - Once the gradients are computed, an optimization algorithm (e.g., Stochastic Gradient Descent, Adam) uses them to update the weights and biases in the direction that reduces the error.
   - The weights are updated by subtracting a fraction of the gradient (controlled by a learning rate) from the current weight value.

Example: Error Propagation

If we have a neural network where:

- y is the true label.
- $\hat{y}$ is the predicted output from the network.
- L is the loss function (e.g., $L=(\hat{y}-y)^2$).

This document is available on studocu
Downloaded by Vaishnavi Bhan (vaishnavibhan8157@gmail.com)

The backpropagation algorithm computes:

1. $\partial L/\partial \hat{y}$ (how the loss changes with the predicted output).
2. $\partial \hat{y}/\partial W$ (how the output changes with the weight WWW).
3. Using the chain rule: $\partial L/\partial W = \partial L/\partial \hat{y} \cdot \partial \hat{y}/\partial W$
4. Finally, the weight WWW is updated as:

$W = W - \eta \partial L/\partial W$ where η is the learning rate.

Note:

1) **Error signals** are passed back from the output layer through the hidden layers.
2) Each neuron in the network learns to adjust its contribution to minimize the overall error.
3) Backpropagation efficiently tunes the weights to ensure that the model becomes more accurate with each iteration of training.

Type-3: Multi-Layer Perceptron (MLP) in Practice

A **Multi-Layer Perceptron (MLP)** is a type of neural network commonly used in practice for solving supervised learning problems, particularly in classification and regression tasks. Its architecture is simple yet effective, consisting of multiple fully connected layers of neurons that can learn complex relationships in the data. Here's how an MLP is structured and used in practical scenarios:

1. MLP Architecture

An MLP typically consists of the following components:

- **Input Layer**: This layer receives the input features (e.g., pixel values for images, sensor data, or tabular data). Each neuron in this layer represents one feature of the input.
- **Hidden Layers**: These are intermediate layers where the actual learning occurs. Each hidden layer consists of multiple neurons, and every neuron in one layer is connected to every neuron in the next layer. The MLP can have one or more hidden layers depending on the complexity of the task.
- **Output Layer**: The final layer, which produces the predicted output. For a classification task, the output might be one or more classes, while for regression, it's typically a continuous value.

2. Activation Functions

Activation functions introduce non-linearity to the model, allowing the MLP to learn complex patterns. Some commonly used activation functions include:

- **ReLU (Rectified Linear Unit)**: $f(x)=\max(0, x)$, used in hidden layers, helps to avoid the vanishing gradient problem.
- **Sigmoid**: Used in the output layer for binary classification, maps values between 0 and 1.
- **Softmax**: Used for multi-class classification, providing probabilities for each class.
- **Tanh**: Like sigmoid but maps values between -1 and 1.

3. Training an MLP

Downloaded by Vaishnavi Bhan (vaishnavibhan8157@gmail.com)

MLPs are trained using **backpropagation** and an optimization algorithm like **Gradient Descent**. The goal is to adjust the weights of the neurons in such a way that the loss (difference between the predicted and actual output) is minimized.

Steps in Training:

1. **Forward Propagation**: Input data is passed through the layers, and the network computes a prediction based on the current weights.
2. **Loss Calculation**: The difference between the predicted output and the actual target is measured using a loss function. For example:

Cross-Entropy Loss for classification tasks.

Mean Squared Error (MSE) for regression tasks.

3. **Backpropagation**: The error is propagated back through the network, calculating the gradients of the loss with respect to the weights. This uses the **chain rule** to compute the error contribution of each weight.
4. **Weight Update**: The optimization algorithm (e.g., **Stochastic Gradient Descent** or **Adam**) updates the weights using the gradients and learning rate. The update rule is:

$W_{new} = W_{old} - \eta \partial L / \partial W$

Where:

W is the weight matrix,

η is the learning rate,

$\partial L / \partial W$ is the gradient of the loss with respect to the weight.

5. **Iteration**: This process is repeated for multiple epochs until the model converges to a low loss value.

---

4. MLP in Practice

(a) Classification Tasks:

MLPs are widely used for tasks where the goal is to categorize input data into predefined classes. Some common use cases include:

- **Image Classification**: Classifying handwritten digits (e.g., MNIST dataset).
- **Text Classification**: Sentiment analysis of text (positive or negative).
- **Medical Diagnosis**: Predicting the presence or absence of disease based on medical data.

For example, in the case of handwritten digit recognition (e.g., MNIST dataset), the MLP takes pixel values of an image as input, passes it through several hidden layers, and outputs the predicted digit (0-9).

This document is available on studocu
Downloaded by Vaishnavi Bhan (vaishnavibhan8157@gmail.com)

(b) Regression Tasks:

In regression tasks, MLPs predict continuous values. Examples include:

- **Predicting house prices** based on features like the number of rooms, size, and location.
- **Stock price prediction** using historical data.

(c) Time Series Forecasting:

Though not as effective as Recurrent Neural Networks (RNNs), MLPs can still be used for time-series forecasting. By using a sliding window approach, historical data is passed as input, and the model predicts future values (e.g., predicting sales, energy consumption).

(d) Natural Language Processing (NLP):

MLPs can be used for simple NLP tasks like sentiment analysis and text classification. However, more advanced architectures like CNNs, RNNs, and Transformers are preferred for tasks like machine translation and language modelling.

5. Tuning Hyperparameters

The performance of an MLP depends on several hyperparameters. In practice, tuning these parameters is crucial for achieving good results:

- **Learning Rate**: Controls how big of a step the optimizer takes during weight updates. A small learning rate can lead to slow convergence, while a large one can cause the model to overshoot minima.
- **Batch Size**: The number of samples used to update the model weights at each iteration. Small batch sizes can provide noisy gradient estimates, while large batch sizes may lead to slower learning.
- **Number of Hidden Layers and Neurons**: The depth and width of the network determine its capacity. More hidden layers or neurons allow the model to learn more complex patterns but may lead to overfitting if the model becomes too complex.
- **Dropout**: A regularization technique that randomly drops neurons during training to prevent overfitting.
- **Number of Epochs**: The number of times the entire dataset is passed through the network. Training for too many epochs can lead to overfitting.

6. Challenges and Limitations

While MLPs are versatile, they do come with some challenges:

- **Overfitting**: MLPs can easily overfit, especially on small datasets or with too many neurons. Regularization techniques like dropout and early stopping are often used to combat overfitting.
- **Computational Cost**: As the number of neurons and layers increases, so does the computational complexity, which can slow down training.
- **Feature Engineering**: MLPs require meaningful input features. Unlike CNNs, which can learn spatial features from raw images, MLPs often need well-structured data.

Downloaded by Vaishnavi Bhan (vaishnavibhan8157@gmail.com)

- **Not Ideal for Sequential Data**: MLPs do not perform well with sequential data like time series or text because they do not capture temporal dependencies. RNNs and LSTMs are better suited for such tasks.

7.MLP Applications in the Real World

MLPs have been successfully used in many real-world applications:

- **Finance**: Credit scoring, fraud detection, and stock price prediction.
- **Healthcare**: Predicting patient outcomes, diagnosing diseases based on medical data.
- **Marketing**: Customer segmentation, churn prediction, and recommendation systems.
- **Manufacturing**: Predictive maintenance and anomaly detection in equipment.

Note:

In practice, an MLP is implemented by defining its architecture, compiling it with appropriate loss functions and optimizers, training it using backpropagation, and evaluating it on unseen data. Hyperparameter tuning, regularization techniques, and early stopping are essential for optimizing performance.

Type-4: Examples of using the Multi-LayerPerceptron (MLP)-overview

Here are some common examples of using **Multi-Layer Perceptron (MLPs)** across different domains, along with a brief overview of their task, data inputs, and outputs:

1. Image Classification

Example: MNIST Handwritten Digit Classification

MLPs can be used for simple image classification tasks where the input images are represented as flattened vectors of pixel values.

- **Task**: Classifying handwritten digits (0-9) from the **MNIST** dataset.
- **Input**: 28x28 pixel grayscale images (flattened into a 784-dimensional vector).
- **Output**: 10 classes (digits from 0 to 9).

Steps:

1. Flatten the 2D image into a 1D vector.
2. Pass the vector through multiple hidden layers with activation functions like ReLU.
3. Use the **softmax** function in the output layer to predict the digit class.

MLPs are effective for small-scale image datasets like MNIST, but for more complex images, **Convolutional Neural Networks (CNNs)** are preferred.

2. Text Classification

Example: Spam Detection in Emails

This document is available on studocu
Downloaded by Vaishnavi Bhan (vaishnavibhan8157@gmail.com)

MLPs are used for classifying text into predefined categories. One common example is spam detection in emails.

- **Task**: Classify an email as "spam" or "not spam".
- **Input**: Email text, converted into a numerical format using techniques like **Bag-of-Words** or **TF-IDF**.
- **Output**: Binary classification (spam or not spam).

Steps:

1. Convert email text into numerical features.
2. Use an MLP with one or more hidden layers to map the input text to a binary output.
3. Use a **sigmoid** function for binary classification at the output layer.

MLPs work well with structured text data, but for sequence-based tasks like language translation, **Recurrent Neural Networks (RNNs)** or **Transformers** are better suited.

3. Regression Tasks

Example: House Price Prediction

In regression tasks, MLPs predict continuous values based on input features. House price prediction is a classic regression problem.

- **Task**: Predict the price of a house based on features like the number of bedrooms, square footage, location, etc.
- **Input**: A set of numerical features (e.g., size, location, number of rooms).
- **Output**: A continuous value representing the predicted house price.

Steps:

1. Use the house's features as input to the MLP.
2. Pass the input through several hidden layers.
3. Use a linear activation function in the output layer to predict the continuous house price.

MLPs perform well for tabular data and can predict prices or other continuous outcomes effectively when trained on relevant features.

4. Medical Diagnosis

Example: Disease Prediction

MLPs can be applied to healthcare problems where the goal is to predict a disease based on patient data (e.g., symptoms, test results, and demographics).

- **Task**: Predict whether a patient has a certain disease based on medical test results.
- **Input**: A set of medical features (e.g., age, blood pressure, cholesterol levels).
- **Output**: Binary classification (presence or absence of the disease).

Downloaded by Vaishnavi Bhan (vaishnavibhan8157@gmail.com)

Steps:

1. Use patient features as input to the MLP.
2. The network's hidden layers learn complex patterns from the data.
3. The output layer uses a **sigmoid** function to predict the probability of disease.

MLPs are widely used in medical diagnosis for conditions like diabetes, heart disease, and cancer detection.

5. Time Series Prediction

Example: Stock Market Forecasting

MLPs can be applied to time series problems by using a sliding window of past data points to predict future values, though they are not ideal for sequential dependencies.

- **Task**: Predict the stock price of a company based on historical prices.
- **Input**: A window of historical stock prices.
- **Output**: The predicted future stock price (regression).

Steps:

1. Organize the time series data into sliding windows (e.g., past 10 days of stock prices as input).
2. Use the MLP to predict the next value in the series.
3. The output layer uses a linear activation function for regression.

MLPs can be effective for simple time series tasks, but models like **RNNs** and **LSTMs** are typically better for capturing temporal relationships.

6. Customer Segmentation

Example: Marketing Segmentation

MLPs are used in marketing to classify customers into different segments based on behaviour or demographics.

- **Task**: Segment customers based on their purchasing behaviour.
- **Input**: Features like purchase frequency, amount spent, and demographic information.
- **Output**: Cluster or classify customers into segments (e.g., high-spenders, budget buyers).

Steps:

1. Use customer behaviour data as input to the MLP.
2. Apply clustering algorithms or supervised classification to segment customers.
3. Use the output to tailor marketing strategies to different segments.

MLPs provide valuable insights into customer behaviour and help companies improve marketing and customer retention strategies.

7. Anomaly Detection

This document is available on studocu
Downloaded by Vaishnavi Bhan (vaishnavibhan8157@gmail.com)

Example: Fraud Detection in Banking

MLPs can be used to detect fraudulent transactions by identifying patterns that deviate from normal behaviour.

- **Task**: Detect fraudulent credit card transactions.
- **Input**: Features of the transaction (e.g., amount, location, time).
- **Output**: Binary classification (fraudulent or non-fraudulent).

Steps:

1. Train the MLP on historical transaction data (with labelled fraud cases).
2. Use the MLP to classify new transactions as fraudulent or non-fraudulent.
3. Use techniques like **oversampling** or **under sampling** to deal with the imbalance in the dataset.

MLPs are widely used in finance for detecting anomalies such as fraud or unusual patterns in transaction data.

8. Energy Consumption Prediction

Example: Predicting Energy Usage

MLPs can predict energy consumption based on historical data and environmental factors.

- **Task**: Predict the energy consumption of a building.
- **Input**: Historical energy usage data and external factors (e.g., temperature, weather conditions).
- **Output**: Predicted energy consumption (continuous value).

Steps:

1. Collect historical energy consumption and environmental data.
2. Train the MLP to predict future energy usage based on past data.
3. Use the MLP for real-time prediction and optimization of energy usage.

Energy companies and industries use MLPs to manage and optimize energy consumption effectively.

Type-5: Deriving Back Propagation:

- **Backpropagation is a key algorithm in training neural networks, used to minimize the error by updating the weights of the network based on the gradient of the loss function.**
- **Here's a step-by-step derivation of backpropagation for a simple feedforward neural network using the chain rule of calculus.**

We are considering a simple neural network with:

- **Input: X (features)**
- **Output: Y^ (predicted value)**
- **Target: Y (true value)**
- **Weights and biases: W and b**

Downloaded by Vaishnavi Bhan (vaishnavibhan8157@gmail.com)

- **Activation function: f (e.g., sigmoid, ReLU)**

1)Forward Propagation:

For a simple 2-layer (1 hidden layer) neural network:

a. Input layer to Hidden layer:

The hidden layer activation $Z^{(1)}$ is computed as:

$$Z^{(1)} = W^{(1)}X + b^{(1)}$$

Apply the activation function f to get the hidden layer output $A^{(1)}$

$$A^{(1)} = f(Z^{(1)})$$

b.Hidden layer to Output layer:

The output layer (pre-activation) $Z^{(2)}$ is computed as:

$$Z^{(2)} = W^{(2)}A^{(1)} + b^{(2)}$$

The output of the network (before applying any activation or softmax) is:

$$\hat{Y} = g(Z^{(2)})$$

where g is a final activation function (like softmax or sigmoid for classification problems).

2. **Loss Function:**

   The error or loss function measures the difference between the predicted output $\hat{Y}$ and the true target Y.

   let's use the squared error loss:

   $$L = \frac{1}{2}(\hat{Y} - Y)^2$$

3. **Backpropagation: Computing Gradients:**

   To update the weights using gradient descent, we need to compute the gradient of the loss with respect to each weight. This is where the chain rule comes in.

Step 1: Derivative of Loss with respect to Output Layer Activation

First, compute the gradient of the loss with respect to the predicted output $\hat{Y}$:

$$\partial L / \partial \hat{Y} = \hat{Y} - Y$$

This document is available on studocu
Downloaded by Vaishnavi Bhan (vaishnavibhan8157@gmail.com)

Step 2: Derivative with respect to Output Layer Weights

Now, to compute the gradient with respect to the output layer weights $W^{(2)}$ we need to apply the chain rule:

Gradient of the loss with respect to the pre-activation Z (2):

$$\partial L/\partial Z^{(2)} = \partial L/\partial Y\hat{} \cdot \partial Y\hat{}/\partial Z^{(2)}$$

For a softmax or sigmoid activation, you'd take the derivative of the specific activation function g

- **Gradient of the loss with respect to the output layer weights $W^{(2)}$**

$$\partial L/\partial W^{(2)} = \partial L/\partial Z^{(2)} \cdot \partial Z^{(2)}/\partial W^{(2)} = \partial L/\partial Z^{(2)} \cdot A^{(1)}$$

Step 3: Derivative with respect to Hidden Layer Weights

Next, compute the gradient with respect to the hidden layer weights $W^{(1)}$

This also requires applying the chain rule:

Gradient of the loss with respect to the hidden layer output $A^{(1)}$

$$\partial L/\partial A^{(1)} = \partial L/\partial Z^{(2)} \cdot \partial Z(2)/\partial A^{(1)} = \partial L/\partial Z^{(2)} \cdot W^{(2)}$$

Gradient of the loss with respect to the pre-activation $Z^{(1)}$ Of the hidden layer:

$$\partial L/\partial Z(1) = \partial L/\partial A(1) \cdot \partial A(1)/\partial Z(1) = \partial L/\partial A(1) \cdot f'(Z^{(1)})$$

Here, f′ is the derivative of the activation function f (e.g., sigmoid or ReLU)

Gradient of the loss with respect to the hidden layer weights $W^{(1)}$

$$\partial L/\partial W^{(1)} = \partial L/\partial Z^{(1)} \cdot \partial Z^{(1)}/\partial W^{(1)} = \partial L/\partial Z^{(1)} \cdot X$$

4. **Weight Updates:**

Once the gradients are computed, the weights can be updated using gradient descent.

example, for the weights of the hidden layer $W^{(1)}$

$$W^{(1)} \leftarrow W^{(1)} - \eta\, \partial L/\partial W^{(1)}$$

where η is the learning rate.

Similarly, the output layer weights $W^{(2)}$ and the biases $b^{(1)}$, $b^{(2)}$ are updated.

Type-6: Radial Basis Functions and Splines:

Downloaded by Vaishnavi Bhan (vaishnavibhan8157@gmail.com)

- Radial Basis Function is defined as the mathematical function that takes real-valued input and provides the real-valued output determined by the distance between the input value and a fixed point projected in space. This fixed point is positioned at an imaginary location within the spatial context.
- This type of transformation function is widely used in various machine learning and deep learning algorithms, such as SVM, Artificial Neural Networks, etc.
- In the context of machine learning and computational mathematics, RBFs are utilized as activation functions in neural networks or as a basic function for interpolation and approximation.
- The most used Radial Basis Function in machine learning is the Gaussian Radial Basis Function.

Example of Radial Basis Function:

- Let us now understand the Radial Basis Function with the help of an example. Now, suppose you want to improve the accuracy of option pricing by considering more sophisticated models that can capture non-linear relationships and varying volatility. Radial Basis Functions offer a suitable solution in this context.
- For simplicity, let's focus on predicting the option price based on the current stock price (S), time to expiration (T), and implied volatility (σ). These factors interact in a complex, non-linear manner. Traditional linear models may struggle to capture the complicated dependencies between these variables.
- In an RBF-based approach, each combination of stock price, time to expiration, and implied volatility is associated with a radial basis function. The radial basis function is high when the input values are close to the center (representing a specific combination of S, T, and σ) and decreases as the distance increases.
- One primary reason for the need of RBFs is their ability to efficiently capture complex, non-linear relationships within data. Unlike simpler linear models, RBFs can model complex patterns and dependencies by transforming input data into a high-dimensional space, where these relationships become more apparent.
- Radial Basis Functions (RBFs) play a crucial role in various fields, including machine learning, signal processing, and numerical analysis, due to their unique mathematical properties and versatile applications.
- This characteristic makes RBFs particularly valuable in tasks such as pattern recognition, interpolation, and approximation, where complex and non-linear relationships are prevalent.
- Radial Basis Functions in neural networks are almost like K-Nearest Neighbor models in terms of conceptuality, though the implementation of both models is quite different. The Radial Basis Function (RBF) operates by assigning weights to input vectors based on their distances from predefined centers or prototypes in the input space.
- The RBF calculates the Euclidean distance between the input vector and the center, squares it, divides it by $2\sigma^2$, and applies an exponential function. The result is a weighted output that reflects the proximity of the input to the center.

Types of Radial Basis Function

There are several types of Radial Basis Functions (RBFs), each with its own characteristics and mathematical formulations. Some common types include:

1.Gaussian Radial Basis Function: It has a bell-shaped curve and is often employed in various applications due to its simplicity and effectiveness.

It is represented as

This document is available on studocu
Downloaded by Vaishnavi Bhan (vaishnavibhan8157@gmail.com)

$$\phi(r) = \exp(-\frac{||x-c||^2}{2\sigma^2})$$

2. Multiquadric Radial Basis Function

It provides a smooth interpolation and is commonly used in applications like meshless methods and

radial basis function interpolation. It is defined as:

$$\phi(r) = \sqrt{1 + (\frac{||x-c||^2}{\sigma^2})}$$

3.Inverse Multiquadric Radial Basis Function:

This type of function is similar to the Multiquadric RBF but has the inverse in the denominator, resulting in a different shape. Here is the formula for this function:

$$\phi(r) = \frac{1}{\sqrt{1+(\frac{||x-c||^2}{\sigma^2})}}$$

4.Thin Plate Spline Radial Basis Function:

The Thin Plate Spline RBF is defined as $\phi(r)= r^2\log(r)$ is the Euclidean distance between the input and the center. This RBF is often used in applications involving thin-plate splines, which are used for surface interpolation and deformation.

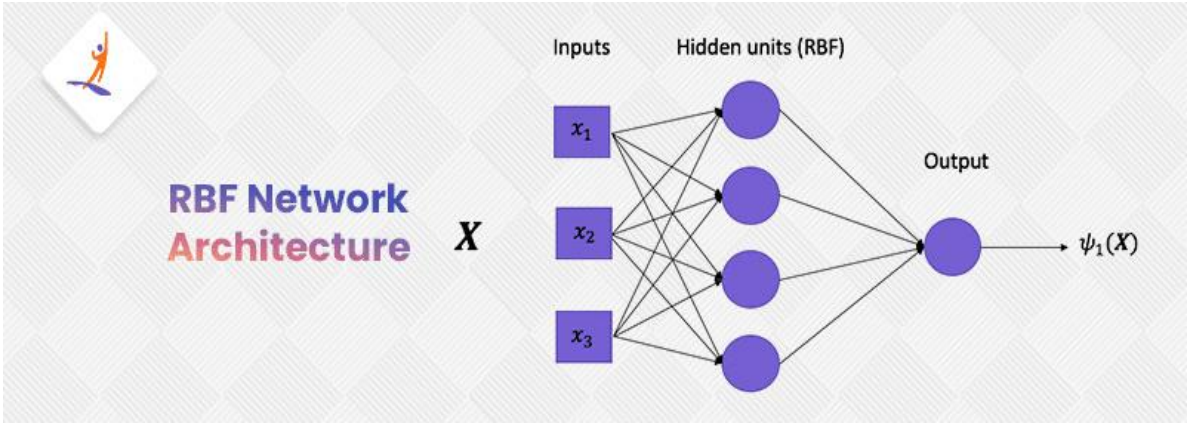
5.Cubic Radial Basis Function:

The Cubic RBF is defined as $\phi(r)= r^3$ where r is the Euclidean distance. It has cubic polynomial behavior and is sometimes used in interpolation.

RBF Network Architecture:

The architecture of an RBFN generally consists of three layers: an input layer, a hidden layer with radial basis functions, and an output layer. Here's a breakdown of the architecture:

Downloaded by Vaishnavi Bhan (vaishnavibhan8157@gmail.com)



Input Layer: The input layer comprises nodes that depict the features or dimensions of the input data, with each node corresponding to an element in the input vector.

Hidden Layer: The hidden layer consists of nodes connected to radial basis functions, where each node functions as a prototype or center for an RBF.

Output Layer: The output layer generates the final network output, usually calculated as a linear combination of the activations originating from the hidden layer.

Advantages of Radial Basis Function:

The advantages of the Radial Basis Function are:

- **Nonlinearity**: RBFs can capture complex nonlinear relationships between variables, making them effective for modeling nonlinear data.
- **Versatility**: They can be used in various machine learning algorithms, such as SVMs, neural networks, and regression models.
- **Flexibility**: RBFs can approximate functions with high precision, especially in multidimensional spaces.

Splines

- Splines are a class of piecewise polynomial functions used in interpolation and approximation, particularly when you want to create smooth curves that pass through a set of points.
- The idea behind splines is to divide the input domain into intervals and fit a low-degree polynomial to each interval while ensuring that the overall curve is smooth.

Types of Splines:

1. **Linear Spline:**
   - A linear spline fits straight lines between adjacent data points.
   - Not very smooth, as the first derivative is not continuous at the knot points (the points where the pieces meet).
2. **Cubic Spline:**
   - A cubic spline uses piecewise cubic polynomials between each pair of points, ensuring that both the first and second derivatives are continuous at the knot points.
   - This smoothness makes cubic splines widely used in practice.

This document is available on studocu
Downloaded by Vaishnavi Bhan (vaishnavibhan8157@gmail.com)

3. **B-Splines (Basis Splines):**
   - B-splines are a more general form of splines where the curve is represented as a linear combination of basis functions.
   - They offer local control over the shape of the curve, meaning that changing one control point affects only a part of the spline.
4. **Natural Spline:**
   - A special case of cubic spline where the second derivative is set to zero at the endpoints, making it "natural."

Mathematical Form:

A cubic spline is defined piecewise as:

$S(x)=\{a_1(x-x_1)^3+b_1(x-x_1)^2+c_1(x-x_1)+d_1 \text{for } x_1 \le x < x_2 a_2(x-x_2)^3+b_2(x-x_2)^2+c_2(x-x_2)+d_2 \text{for } x_2 \le x < x_3\}$

The coefficients are determined by solving a system of equations that enforce the smoothness conditions (continuity of the first and second derivatives).

Applications:

- **Data fitting**: Splines are often used to fit smooth curves through data points, especially in cases where the data is noisy.
- **Computer graphics**: Splines, especially B-splines and NURBS (Non-Uniform Rational B-Splines), are widely used in computer graphics for modeling smooth shapes and surfaces.
- **Signal processing**: Splines can be used to approximate signals with smooth functions, useful for filtering or signal compression.

Example (Cubic Spline Interpolation):

Suppose you have a set of data points $(x_1,y_1),(x_2,y_2),\ldots,(x_n,y_n)$. The goal is to find a cubic spline

that interpolates the points**.**

Comparison of Radial Basis Functions and Splines

Aspect	Radial Basis Functions (RBF)	Splines
Basic Idea	Use radial functions based on distance from centers	Use piecewise polynomials on intervals
Continuity	Globally continuous (smooth everywhere)	Typically continuous at knots (piecewise smooth)
Flexibility	Can easily handle scattered data in high dimensions	Primarily for 1D and 2D data interpolation
Smoothness	Controlled by the choice	Smoothness ensured

Downloaded by Vaishnavi Bhan (vaishnavibhan8157@gmail.com)

Aspect	Radial Basis Functions (RBF)	Splines
Control	of RBF (e.g., Gaussian spread)	by matching derivatives
Local vs Global	Changes in one point can affect the whole interpolation	Localized control over shape (especially B-splines)
Applications	Function approximation, machine learning (RBF networks)	Curve fitting, computer graphics, data smoothing

Note:1. Radial Basis Functions (RBF): Used in high-dimensional spaces for smooth interpolation and function approximation, with applications in machine learning (e.g., RBF networks).

2. Splines: Primarily used for piecewise polynomial approximation in low-dimensional spaces (1D or 2D), widely employed in graphics and data fitting due to their smoothness and local control.

Type-7: Concepts-- RBF (Radial Basis Function)

An RBF (Radial Basis Function) Network is a type of artificial neural network that uses radial basis functions as activation functions. It is primarily used for classification, regression, and function approximation tasks. The RBF network typically consists of three layers: the input layer, the hidden layer, and the output layer. Unlike traditional multi-layer perceptrons (MLPs) that use sigmoid or ReLU activation functions, RBF networks rely on functions that respond to the distance between input data points and "centers" or "prototypes."

Key Concepts in RBF Networks:

1. Radial Basis Function (RBF):

- An RBF is a real-valued function whose value depends on the distance from a center point.
- Common choices of RBFs include:

Gaussian (most common):

$$\phi(r) = \exp(-\frac{||x-c||^2}{2\sigma^2})$$

Multiquadric:

$$\phi(r) = \sqrt{1 + (\frac{||x-c||^2}{\sigma^2})}$$

Inverse Multiquadric:

This document is available on studocu
Downloaded by Vaishnavi Bhan (vaishnavibhan8157@gmail.com)

$$\phi(r) = \frac{1}{\sqrt{1+(\frac{||x-c||^2}{\sigma^2})}}$$

Thin Plate Spline: $\phi(r)=r2\log(r)$

Here, r is the Euclidean distance between the input vector xxx and the center ccc, and σ is a parameter controlling the width (or spread) of the function.

2. Network Structure:

An RBF network has three layers:

1. Input Layer:
   - Each neuron corresponds to one input feature. No computations are performed in this layer; it simply passes the input features to the hidden layer.
2. Hidden Layer:
   - The hidden layer consists of neurons that use RBFs as activation functions. Each hidden neuron has a centercic_ici and computes the distance between the input vector xxx and the center.
   - The neuron then applies the chosen RBF to this distance.
   - The number of neurons in this layer is typically chosen based on the complexity of the problem.
3. Output Layer:
   - The output is a weighted linear combination of the RBFs from the hidden layer.
   - If the network is used for classification, the output can be fed into a softmax function to assign class probabilities.
   - For regression tasks, the output is a continuous value based on the weighted sum of the activations in the hidden layer.

3. Mathematical Model:

Given an input vector x, the output $f(x)$ of the RBF network is given by:

$$\mathbf{f(x)=\sum_1^m wi(\phi \| x - ci \|)}$$

where:

wi are the weights applied to the RBF outputs,

$\phi(\|x-ci\|)$ is the radial basis function .

m is the number of hidden units (centers).

Downloaded by Vaishnavi Bhan (vaishnavibhan8157@gmail.com)

For classification, the output layer will produce a vector of class probabilities, and the prediction is made by selecting the class with the highest probability.

4. Training an RBF Network:

Training an RBF network involves two main steps:

1. Determining the centers and widths (parameters of the RBFs):
    - K-means clustering is often used to select centers from the input data, as it finds representative points (centroids) for different regions in the input space.
    - The width σ\sigmaσ of the RBF can be chosen as a function of the distance between centers (e.g., the average distance between clusters) or determined through cross-validation.
2. Training the output layer weights:
    - Once the centers and widths are fixed, the output weights wiw_iwi are learned.
    - Since the hidden layer behaves like a fixed feature extractor, learning the output weights is a linear regression problem and can be solved using the least squares method or gradient descent.

In the case of classification, if there are multiple classes, each class is assigned an output neuron, and the problem is reduced to finding the correct output weights.

5. Advantages of RBF Networks:

- Fast training: Since only the output weights need to be learned once the centers are chosen, RBF networks often train faster than backpropagation-based neural networks like MLPs.
- Interpolation capability: RBF networks perform well on interpolation tasks due to their localized response to input points. This is particularly useful in cases where input data is sparse or non-linearly separable.
- Simple structure: The division between the hidden and output layers simplifies the training process, as the training for the output layer is typically linear.

6. Disadvantages of RBF Networks:

- Scalability: Choosing the right number of centers and efficiently training the network becomes challenging for large datasets or high-dimensional input spaces.
- Center selection: The performance of the RBF network heavily depends on the choice of centers. If centers are poorly chosen, the network may not generalize well to unseen data.
- Global generalization: RBF networks perform well when data is locally distributed but may struggle with more global relationships between data points compared to deeper architectures.

7. Applications of RBF Networks:

- Function Approximation: RBF networks are commonly used in situations where you want to model a smooth function from scattered or noisy data points.
- Classification: The use of RBFs allows the network to form decision boundaries that are non-linear and can adapt well to complex data distributions.

This document is available on studocu
Downloaded by Vaishnavi Bhan (vaishnavibhan8157@gmail.com)

- Time Series Prediction: RBF networks can be used to predict future values of a time series by approximating the underlying function that generates the data.
- Control Systems: In robotic control, RBF networks are used to model the relationship between control inputs and system behavior due to their ability to approximate highly non-linear functions.

Example of an RBF Network in Classification:

1. Input: A dataset with features (e.g., 2D points).
2. Hidden Layer:
    - Centers (cluster centroids) are chosen using a clustering algorithm like k-means.
    - The hidden layer applies the RBF (e.g., Gaussian) based on the distance from each center to the input points.
3. Output Layer:
    - The output is a weighted sum of the RBF outputs.
    - For classification, a softmax layer may be applied to produce probabilities for different classes.
4. Prediction:
    - The class with the highest probability is selected as the predicted class.

Key Points:

- RBF Networks are a type of neural network that uses radial basis functions in the hidden layer.
- The network consists of an input layer, a hidden layer where RBFs are applied, and an output layer that performs a weighted sum of the hidden layer outputs.
- Training involves choosing centers (often using clustering) and learning output weights (using linear regression or gradient descent).
- RBF Networks are well-suited for interpolation, function approximation, and classification tasks in low to moderate-dimensional spaces. However, they can struggle with large or complex datasets

Type-8: Curse of Dimensionality:

The Curse of Dimensionality refers to the various phenomena that arise when analyzing and organizing data in high-dimensional spaces. As the number of dimensions (or features) increases, the volume of the space increases exponentially, which presents numerous challenges for data analysis, machine learning, and optimization. The term was popularized by Richard Bellman in the 1960s in the context of dynamic programming.

Key Aspects of the Curse of Dimensionality:

1. Exponential Growth of Volume

- In high-dimensional spaces, the number of data points required to densely populate the space grows exponentially with the number of dimensions.
- For example, consider a 1D space (line) of length 1, a 2D space (square) of area 1, and a 3D space (cube) of volume 1. As you increase dimensions, filling the space with a fixed number of points (e.g., grid or random sampling) becomes infeasible because the space "grows" exponentially.

Example:

Downloaded by Vaishnavi Bhan (vaishnavibhan8157@gmail.com)

- A grid with 10 points per dimension in 1D requires 10 points.
- In 2D, it needs 10^2=100 points.
- In 10 dimensions, it needs 10^{10}=10,000,000,00010 points.
- \

This exponential growth in the amount of data required makes high-dimensional problems computationally expensive.

2. Data Sparsity

- As the number of dimensions increases, data points become increasingly sparse in the feature space. Even with large datasets, the relative density of data points decreases.
- In high dimensions, most of the data points tend to lie near the boundaries of the space, as opposed to the center. This sparsity makes it difficult to generalize well because there isn't enough data to cover the space uniformly.

3. Distance Metrics in High Dimensions

- Many machine learning algorithms (e.g., k-nearest neighbour's, clustering, etc.) rely on distance metrics (like Euclidean distance) to measure similarity between data points.
- In high-dimensional spaces, distances between any two points tend to become very similar, making it difficult to differentiate between points based on distance.

Example:

- In low dimensions, the distance between two random points might vary significantly.
- In high dimensions, the distance between two random points converges to a nearly constant value, making it difficult for algorithms to distinguish between "close" and "far" points.

4. Volume Concentration

- A phenomenon related to data sparsity and distance metrics is the concentration of volume. In high-dimensional spaces, most of the volume is concentrated in the corners or near the boundaries of the space.
- As a result, data points tend to be located far from each other, and intuitive geometric concepts from low dimensions no longer hold. For instance, in high-dimensional spaces, most of the data points tend to cluster near the surface of a high-dimensional sphere rather than in the interior.

5. Increased Computational Complexity

- Machine learning models in high-dimensional spaces require more computational resources due to the increased number of features. Training and inference time grow as the number of dimensions increases.
- For algorithms that rely on pairwise comparisons (e.g., nearest neighbors, distance-based methods), the number of pairwise computations also grows rapidly, adding to the computational burden.

6. Overfitting in High Dimensions

This document is available on studocu
Downloaded by Vaishnavi Bhan (vaishnavibhan8157@gmail.com)

- In high-dimensional spaces, models can overfit very easily. Overfitting occurs when the model captures noise and irrelevant patterns from the data rather than the underlying trend.
- With more features (dimensions), it's easier to find correlations between the input features and the output labels, even if these correlations are spurious. Therefore, high-dimensional models tend to fit the training data too well but fail to generalize to unseen data.

7. Dimensionality Reduction

- To combat the curse of dimensionality, various dimensionality reduction techniques are employed to reduce the number of features while retaining as much information as possible:
    - Principal Component Analysis (PCA): A linear method that reduces dimensionality by projecting the data onto the directions of maximum variance.
    - t-SNE and UMAP: Non-linear dimensionality reduction methods that are useful for visualizing high-dimensional data.
    - Autoencoders: Neural network-based models that can learn compressed representations of high-dimensional data.
    - Feature Selection: Instead of creating new features, select only the most important features that have a significant impact on the output.

Practical Examples of the Curse of Dimensionality

1. K-Nearest Neighbors (KNN):
    - KNN relies on distance to measure the similarity between points. In high-dimensional spaces, distances between all points become nearly equal, which makes it difficult to find meaningful neighbors.
    - This reduces the effectiveness of the algorithm in high dimensions, as the nearest neighbors are no longer representative of the local structure of the data.
2. Clustering Algorithms (e.g., K-Means):
    - In high-dimensional spaces, clusters become harder to form because the distances between points become more uniform, making it challenging to assign points to meaningful clusters.
    - The performance of clustering algorithms degrades as the dimensionality of the data increases.
3. Linear Models (e.g., Linear Regression):
    - Linear models, such as linear regression, require more data as the number of dimensions increases. The model tries to find a hyperplane in high-dimensional space, but without sufficient data, the solution may overfit the training data.
    - This can lead to poor generalization and increased error on new data.

Strategies to Overcome the Curse of Dimensionality:

1. Feature Engineering:
    - Carefully crafting features or using domain knowledge to reduce the number of dimensions can help avoid the curse of dimensionality.
    - For example, combining features or removing irrelevant or redundant features can make the model more robust.
2. Dimensionality Reduction Techniques:
    - Principal Component Analysis (PCA): This transforms high-dimensional data into a lower-dimensional space while preserving the most important variance in the data.

Downloaded by Vaishnavi Bhan (vaishnavibhan8157@gmail.com)

- Autoencoders: These neural networks can learn a compressed representation of the input data, helping reduce the dimensionality.
- Feature Selection: Methods like forward selection, backward elimination, or regularization (e.g., L1 regularization) can help choose a subset of features that matter most.

3. Regularization:
   - Regularization techniques like L1 (Lasso) and L2 (Ridge) regularization help constrain the model, preventing it from overfitting in high-dimensional spaces by penalizing the magnitude of the model's coefficients.
   - These methods help by encouraging sparsity in the model, effectively reducing the number of features used.
4. Use of Non-Euclidean Metrics:
   - Instead of relying on Euclidean distance, some models use different distance metrics or kernels that are better suited to high-dimensional spaces (e.g., cosine similarity or Mahalanobis distance).
5. Adding More Data:
   - A simple but often difficult-to-achieve solution is to gather more data. As dimensionality increases, the volume of the space grows, so having more data helps fill the space and makes learning easier.

Note:

The Curse of Dimensionality refers to the challenges that arise when working with high-dimensional data, including data sparsity, increased computational complexity, and difficulty in generalizing models. Techniques like dimensionality reduction, feature selection, regularization, and careful feature engineering are commonly used to mitigate these challenges. Understanding and addressing the curse of dimensionality is crucial for developing efficient and accurate machine learning models, especially when dealing with large datasets with many features.

Type-9: Interpretations and Basis Functions:

Interpretations and Basis Functions refer to two fundamental concepts in function approximation and machine learning, particularly in the context of models like Radial Basis Function (RBF) networks, splines, and other linear models. These concepts explain how complex functions or data can be approximated or represented using a combination of simpler functions (basis functions).

1. Basis Functions:

Basis functions are a set of functions used to represent other functions, especially in the context of function approximation, interpolation, or regression. In machine learning, they transform the input data into a higher-dimensional space where it becomes easier to model complex relationships.

Common Types of Basis Functions:

- Polynomial Basis Functions:
  - These are used in polynomial regression.
  - Example: $f(x)=w0+w1x+w2x2+\cdots+wnxn$
  - Here, each xn is a basis function.
- Radial Basis Functions (RBFs):
  - These are functions that depend on the distance between an input and a fixed center.

This document is available on studocu
Downloaded by Vaishnavi Bhan (vaishnavibhan8157@gmail.com)

  - Example: Gaussian RBF: $\phi(x) = \exp\left(-\frac{\|x - c\|^2}{2\sigma^2}\right)$, where c is the center and σ is the spread parameter.
- Fourier Basis:
  - Used in Fourier series for representing periodic functions.
  - Example: $f(x) = a_0 + \sum(a_n \cos(nx) + b_n \sin(nx))$, where $\cos(nx)$ and $\sin(nx)$ are basis functions.
- Wavelet Basis:
  - These are localized functions that can capture both time (or spatial) and frequency information.
  - Example: Haar wavelet or Daubechies wavelets, which are useful for capturing sudden changes in data.
- Spline Basis Functions:
  - These are piecewise polynomial functions, often used for smooth interpolation between data points.
  - Example: B-splines or cubic splines, where the function is divided into segments, each represented by a different polynomial.

How Basis Functions Are Used:

- In models like linear regression or kernel methods, the input data is transformed using a set of basis functions, which are then combined to form a complex function.
- The choice of basis functions is crucial because it determines how well the model can capture the underlying patterns in the data.
- Basis functions provide flexibility in modeling, enabling the model to represent non-linear relationships in the data.

Example: Linear Combination of Basis Functions

Let $\{\phi_1(x), \phi_2(x) \ldots\ldots \phi_n(x)\}$ be a set of basis functions. Any function $f(x)$ can be approximated as a linear combination of these basis functions:

$$f(x) \approx w_1\phi_1(x) + w_2\phi_2(x) + \cdots + w_n\phi_n(x)$$

where $w_1, w_2, \ldots.. w_n$ are the weights (or coefficients) to be learned by the model.

2. Interpretations of Basis Functions:

The interpretation of basis functions depends on the context in which they are used and how they transform the data. Here are a few interpretations of basis functions in various contexts:

a. Function Approximation:

- In function approximation, basis functions serve as building blocks that allow us to approximate complex functions. Instead of directly modeling a complex function, we represent it as a weighted sum of simpler functions (basis functions).
- For example, polynomial basis functions can approximate smooth, continuous functions, while spline basis functions are well-suited for representing functions with localized features or discontinuities.

b. Radial Basis Function Networks:

Downloaded by Vaishnavi Bhan (vaishnavibhan8157@gmail.com)

- In RBF networks, basis functions (usually Gaussian or other radial functions) represent localized regions of the input space. Each RBF responds strongly to inputs close to its center and weakly to those far away.
- The interpretation here is that each basis function can be seen as capturing a local feature of the data, and the combination of these local features gives a global representation of the function.

c. Kernel Methods (SVMs, Gaussian Processes):

- In kernel methods, a kernel function implicitly defines a basis function in a high-dimensional space. The kernel represents the similarity between input points without explicitly computing the transformation into the high-dimensional space.
- For instance, in support vector machines (SVMs), the kernel trick allows for the use of complex, non-linear decision boundaries by mapping data to a higher-dimensional space where a linear decision boundary can separate the data.

d. Fourier Series:

- In the context of Fourier series, basis functions like sine and cosine allow periodic functions to be expressed as a sum of simple oscillatory components.
- Each basis function in this case represents a different frequency, and the coefficients represent the amplitude of each frequency in the overall function. The interpretation is that any periodic signal can be decomposed into a set of oscillatory (sinusoidal) components.

e. Wavelets:

- Wavelets allow for localized basis functions that can capture both time and frequency characteristics in the data. This is particularly useful for non-stationary signals (e.g., signals that change over time).
- In wavelet analysis, the basis functions are scaled and shifted versions of a "mother wavelet," allowing them to represent both large-scale (low-frequency) and small-scale (high-frequency) features of the data.

3. Why Basis Functions Are Important:

- Non-linearity: By transforming the input data using basis functions, we can model non-linear relationships in the data using linear models (e.g., linear regression, SVM).
- Flexibility: Basis functions provide the model with flexibility, allowing it to fit a wide range of functional forms without requiring deep architectures or complex optimization techniques.
- Interpretability: In some cases, the coefficients of basis functions can provide insight into the underlying structure of the data. For instance, in Fourier analysis, the coefficients indicate the contribution of different frequencies in the data.

4. Example: Polynomial Regression Using Basis Functions

Suppose we want to fit a model to a set of data points (x_i, y_i) using polynomial regression. We can use polynomial basis functions:

- Basis functions: $\phi 1(x)=1, \phi 2(x)=x, \phi 3(x)=x_2, \ldots . \phi_n(x)=xn-1$.

Summary:

This document is available on studocu
Downloaded by Vaishnavi Bhan (vaishnavibhan8157@gmail.com)

- Basis functions are a set of functions used to represent or approximate more complex functions. They transform the input data into a higher-dimensional space where the relationships become easier to model.
- Different types of basis functions (polynomial, radial, Fourier, wavelet) are used in various applications, such as regression, classification, signal processing, and function approximation.
- Interpretations of basis functions depend on the context: they can represent local features in RBF networks, oscillatory components in Fourier analysis, or wave-like patterns in wavelet analysis.
- The ability to approximate complex functions using a set of basis functions is crucial in machine learning, particularly in models like RBF networks, kernel methods, and linear regression.

Type-10: Support Vector Machines (SVM):

Support Vector Machines (SVM) are a class of supervised machine learning algorithms primarily used for classification tasks, but they can also be adapted for regression (called Support Vector Regression or SVR). The key idea behind SVM is to find a hyperplane that best separates the data points into different classes, while maximizing the margin between the classes.

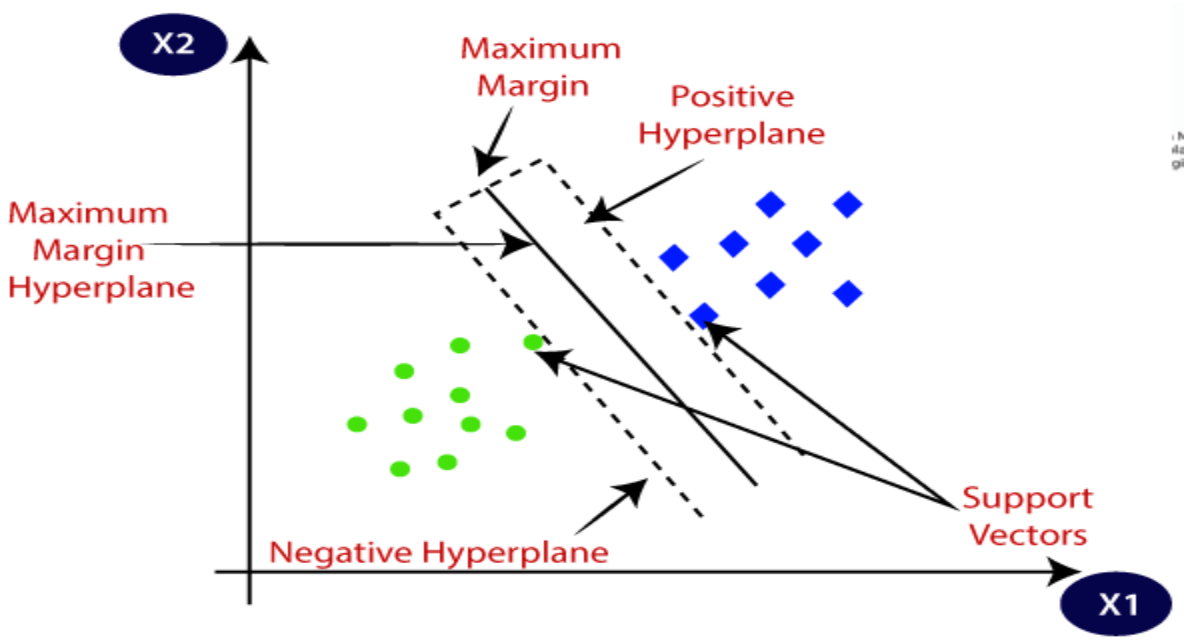
Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems.

However, primarily, it is used for Classification problems in Machine Learning. The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future.

This best decision boundary is called a hyperplane. SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called as support vectors, and hence algorithm is termed as Support Vector Machine.

Consider the below diagram in which there are two different categories that are classified using a decision boundary or hyperplane:

Downloaded by Vaishnavi Bhan (vaishnavibhan8157@gmail.com)



SVM can be of two types:

1. **Linear SVM:**

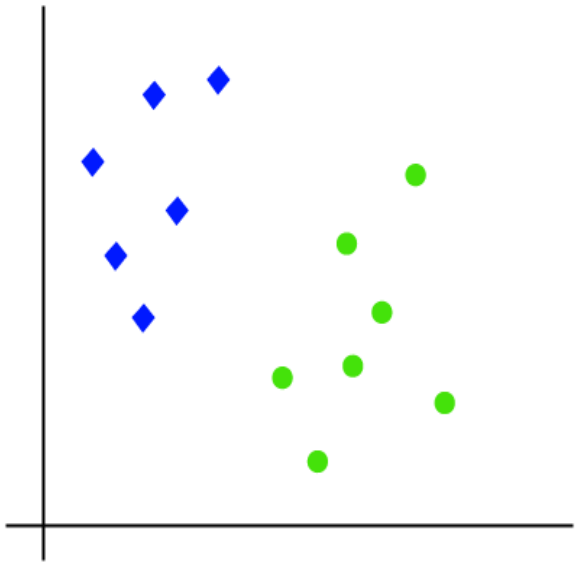
Linear SVM is used for linearly separable data, which means if a dataset can be classified into two classes by using a single straight line, then such data is termed as linearly separable data, and classifier is used called as Linear SVM classifier.

The working of the SVM algorithm can be understood by using an example. Suppose we have a dataset that has two tags (green and blue), and the dataset has two features x1 and x2. We want a classifier that can classify the pair (x1, x2) of coordinates in either green or blue.

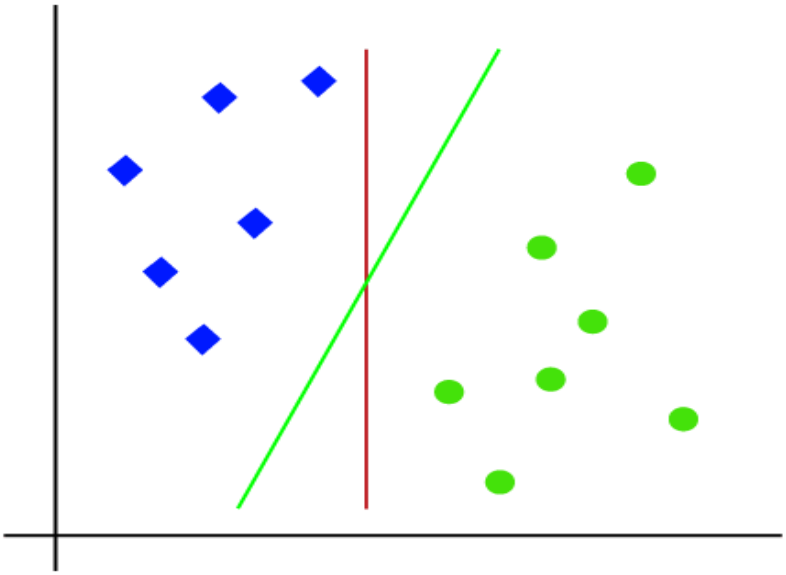
Consider the below image. It is 2-d space so by just using a straight line, we can easily separate these two classes. But there can be multiple lines that can separate these classes.

Consider the below image:

This document is available on studocu
Downloaded by Vaishnavi Bhan (vaishnavibhan8157@gmail.com)



SVM – Input Space



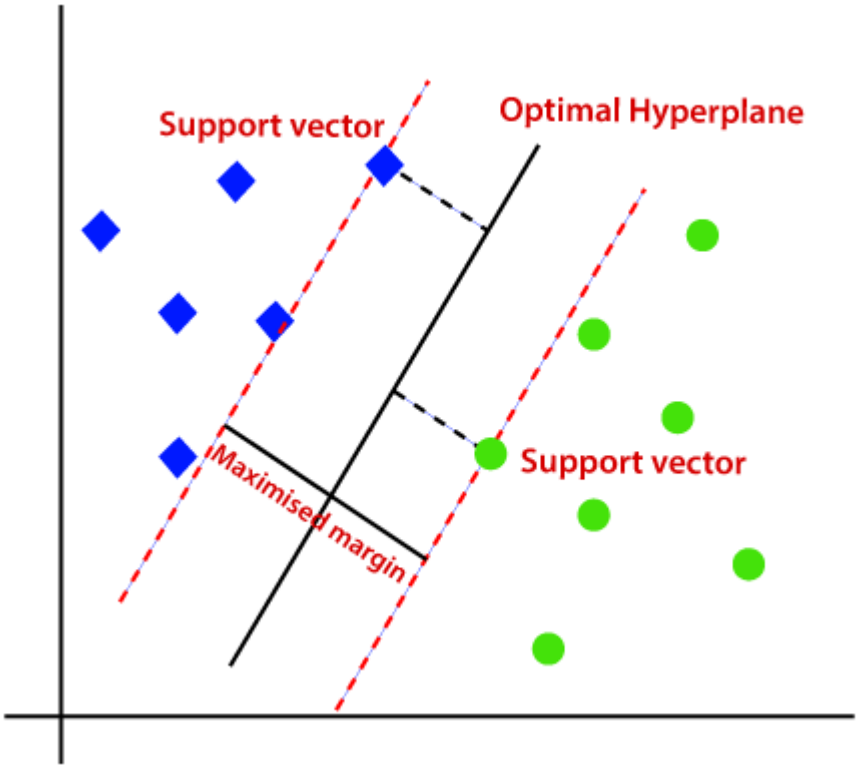
SVM – Linear Classification

Hence, the SVM algorithm helps to find the best line or decision boundary; this best boundary or region is called as a hyperplane. SVM algorithm finds the closest point of the lines from both the classes. These points are called support vectors.

The distance between the vectors and the hyperplane is called as margin.

And the goal of SVM is to maximize this margin. The hyperplane with maximum margin is called the optimal hyperplane.

Downloaded by Vaishnavi Bhan (vaishnavibhan8157@gmail.com)

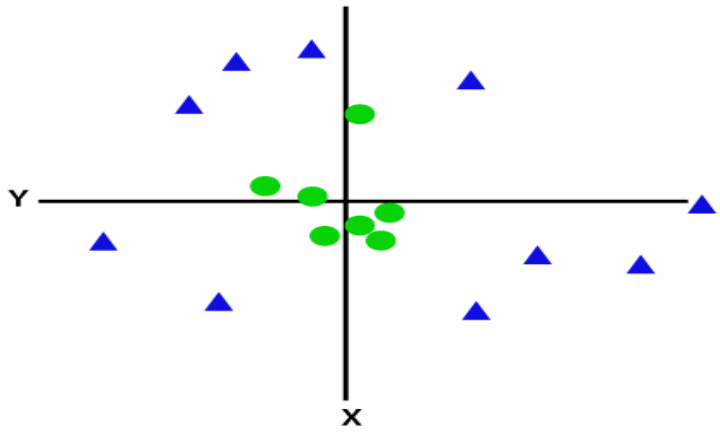


2.Non-linear SVM:

Non-Linear SVM is used for non-linearly separated data, which means if a dataset cannot be classified by using a straight line, then such data is termed as non-linear data and classifier used is called as Non-linear SVM classifier

If data is linearly arranged, then we can separate it by using a straight line, but for non-linear data, we cannot draw a single straight line.

Consider the below image:



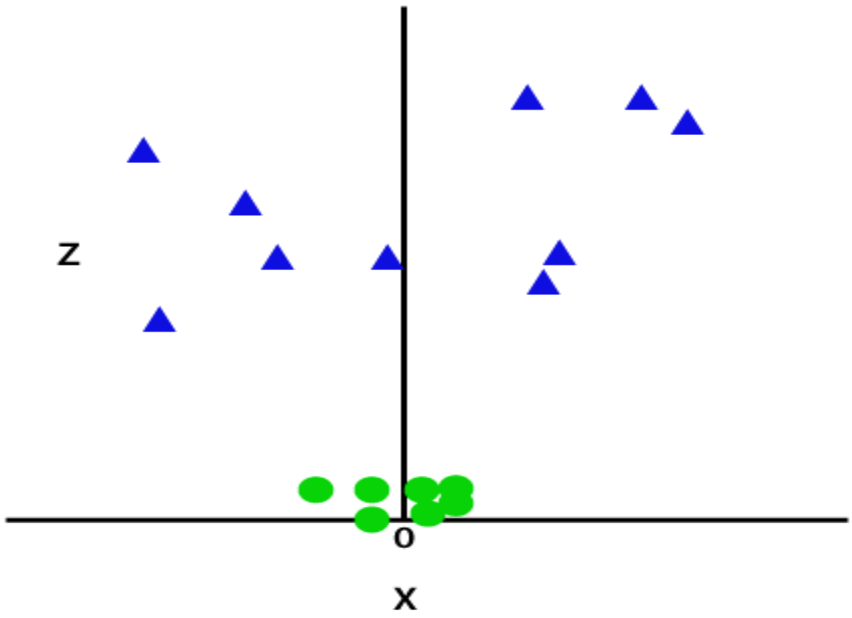
So, to separate these data points, we need to add one more dimension. For linear data, we have used two dimensions x and y, so for non-linear data, we will add a third-dimension z.

This document is available on studocu
Downloaded by Vaishnavi Bhan (vaishnavibhan8157@gmail.com)

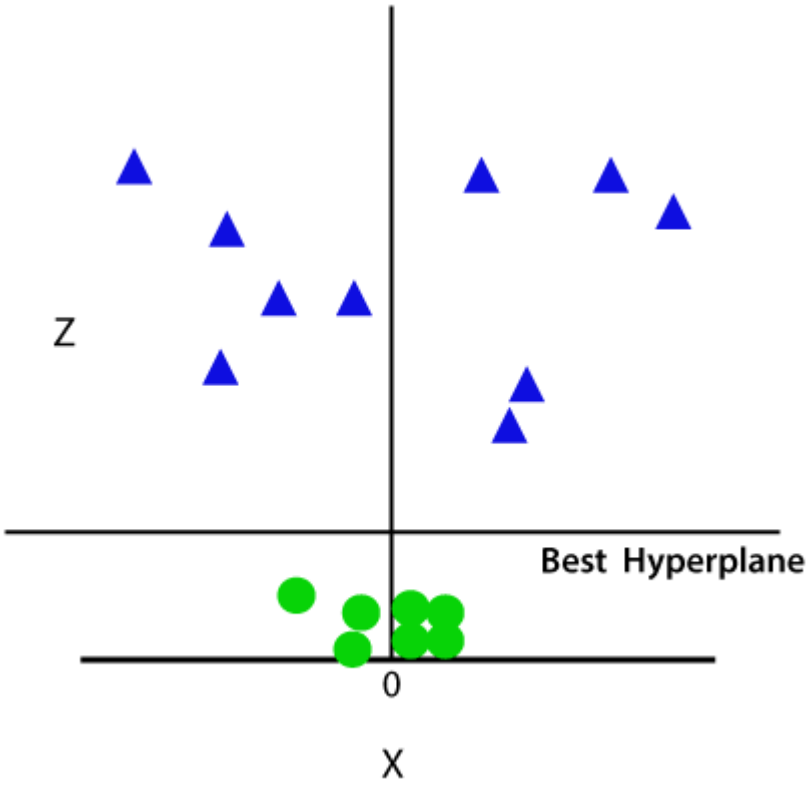
It can be calculated as:

$z=x^2+y^2$

By adding the third dimension, the sample space will become as below image:

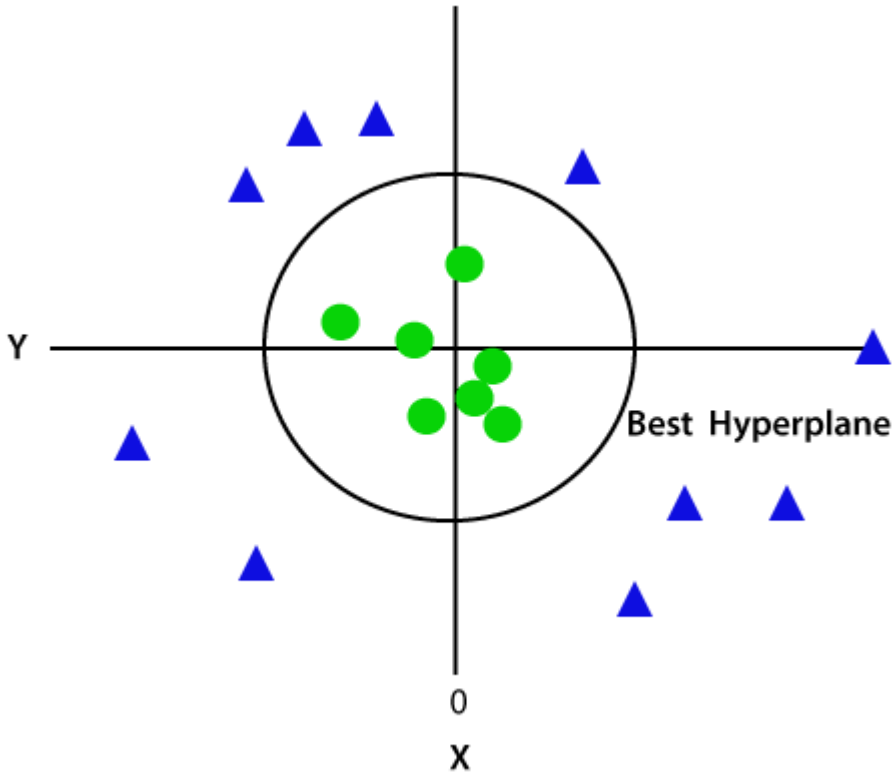


So now, SVM will divide the datasets into classes in the following way. Consider the below image:



Downloaded by Vaishnavi Bhan (vaishnavibhan8157@gmail.com)

Since we are in 3-d Space, hence it is looking like a plane parallel to the x-axis. If we convert it in 2d space with z=1, then it will become as:



Hence, we get a circumference of radius 1 in case of non-linear data.

Note:

Support Vectors:

The data points or vectors that are the closest to the hyperplane and which affect the position of the hyperplane are termed as Support Vector. Since these vectors support the hyperplane, hence called a Support vector.

Key Concepts of SVM:

1. Hyperplane:

- A hyperplane in an nnn-dimensional space is a flat affine subspace of one dimension less than the space itself. For example, in 2D, it's a line; in 3D, it's a plane.
- In SVM, the goal is to find the optimal hyperplane that separates the data points of different classes. This hyperplane can be linear or non-linear, depending on the data.

This document is available on studocu
Downloaded by Vaishnavi Bhan (vaishnavibhan8157@gmail.com)

2. Margin:

- The margin is the distance between the hyperplane and the nearest data points from each class. These closest data points are known as support vectors.
- SVM seeks to maximize this margin, making the decision boundary more robust and generalizable to new data points.

3. Support Vectors:

- Support vectors are the data points that lie closest to the hyperplane and have a direct impact on its position and orientation.
- These points are critical because the SVM model is defined by them. If we remove or move a support vector, the decision boundary could change.

4. Maximizing the Margin:

- The SVM model aims to find the hyperplane that maximizes the margin between the two classes. A larger margin often leads to better generalization and performance on unseen data.
- Mathematically, this is formulated as an optimization problem where we minimize the norm of the weight vector subject to the constraint that all points are correctly classified (or classified within a tolerable margin).

5. Linear vs. Non-linear SVM:

- Linear SVM: If the data is linearly separable, SVM can directly find a hyperplane that separates the classes.
- Non-linear SVM: If the data is not linearly separable, SVM uses the kernel trick to transform the input space into a higher-dimensional space, where a hyperplane can separate the data. This is a key strength of SVM.

SVM for Regression (SVR):

Support Vector Regression (SVR) is an adaptation of SVM for regression problems, where the goal is to predict continuous values.

- The key idea in SVR is to fit a function within a margin of tolerance (called ϵ\epsilon) to the training data.
- Data points that fall within this margin are not penalized, while points outside the margin incur a penalty based on their distance from the margin.

Advantages of SVM:

1. Effective in High-Dimensional Spaces: SVM works well even when the number of features exceeds the number of samples.
2. Memory Efficiency: Only a subset of the training data (the support vectors) is used in the final decision function, making SVM efficient.
3. Flexibility with Kernels: The use of kernels allows SVM to handle non-linearly separable data by implicitly mapping it into higher-dimensional space.

Downloaded by Vaishnavi Bhan (vaishnavibhan8157@gmail.com)

4. Robust to Overfitting: SVM is less prone to overfitting, especially in high-dimensional spaces, as it focuses on maximizing the margin rather than fitting all data points exactly.

Disadvantages of SVM:

1. Choice of Kernel and Parameters: The performance of SVM heavily depends on the choice of kernel and tuning of parameters like CCC and γ\gammaγ. Selecting the right combination often requires cross-validation.
2. Training Time: SVM can be computationally intensive, especially with large datasets. The training time scales quadratically with the number of samples, which makes it less suitable for very large datasets.
3. Interpretability: SVM models, especially with non-linear kernels, can be difficult to interpret compared to simpler models like decision trees.

Example:

Let's say we have two classes of points, and we want to separate them using SVM. If the data is linearly separable, SVM will find a straight line (in 2D) or a hyperplane (in higher dimensions) that separates the two classes with the maximum margin. If the data is not linearly separable, the kernel trick will map the data into a higher-dimensional space, where a hyperplane can be found to separate the classes.

Note:

- Support Vector Machines (SVM) are powerful supervised learning models for classification and regression tasks.
- The key idea is to find a hyperplane that maximizes the margin between classes, using support vectors.
- The kernel trick allows SVM to handle non-linearly separable data by implicitly mapping it to a higher-dimensional space.
- SVM is widely used for its robustness in high-dimensional spaces and flexibility through different kernels. However, it can be computationally expensive for large datasets, and selecting the right parameters can be challenging.

This document is available on studocu
Downloaded by Vaishnavi Bhan (vaishnavibhan8157@gmail.com)