Classification- logistic regression & Naïve Bayes

Explain logistic regression.

Logistic Regression

Logistic regression is a supervised machine learning algorithm used for classification tasks where the goal is to predict the probability that an instance belongs to a given class or not.

What is Logistic Regression?

- Logistic regression is used for binary classification where we use a sigmoid function to map the input features to a probability value between 0 and 1.
- It predicts the probability that an instance belongs to a particular class, making it suitable for binary classification problems.
- Unlike linear regression, logistic regression predicts categorical or discrete values instead of continuous values.

Key Points:

1. **Predictive Output**: Logistic regression predicts the output of a categorical dependent variable, which can be binary (e.g., Yes or No, 0 or 1) or multi-class.
2. **Sigmoid Function**: Logistic regression uses the sigmoid function to map the predicted values to probabilities. This function ensures that the output probability remains between 0 and 1.
3. **Types of Logistic Regression**:
    - *Binomial*: Two possible types of dependent variables (e.g., Pass or Fail).
    - *Multinomial*: Three or more possible unordered types of dependent variables (e.g., categories like "cat", "dog", or "sheep").
    - *Ordinal*: Three or more possible ordered types of dependent variables (e.g., "low", "medium", or "high").
4. **Assumptions**: Logistic regression assumes independent observations, binary dependent variables, linearity between independent variables and log odds, no outliers, and a sufficiently large sample size.

Terminologies involved in Logistic Regression:

- *Independent variables*: Input characteristics or predictor factors.
- *Dependent variable*: Target variable being predicted.
- *Logistic function*: Maps independent variables to a probability value between 0 and 1.
- *Odds*: Ratio of something occurring to something not occurring.
- *Log-odds*: Natural logarithm of the odds.
- *Coefficient*: Estimated parameters of the logistic regression model.
- *Intercept*: Constant term representing the log odds when all independent variables are zero.
- *Maximum likelihood estimation*: Method used to estimate the coefficients of the

logistic regression model.

How does Logistic Regression work?

- Logistic regression transforms the linear regression function's continuous output into categorical values using a sigmoid function.
- The sigmoid function maps real-valued input features into a probability value between 0 and 1.
- Based on a chosen threshold value (typically 0.5), instances with predicted probabilities above the threshold are classified into one class, while those below are classified into the other.

Let the independent input features be:

$$X = \begin{bmatrix} x_{11} & \cdots & x_{1m} \\ x_{21} & \cdots & x_{2m} \\ \vdots & \ddots & \vdots \\ x_{n1} & \cdots & x_{nm} \end{bmatrix}$$

and the dependent variable is Y having only binary value i.e. 0 or 1.

$$Y = \begin{cases} 0 & \text{if } Class\ 1 \\ 1 & \text{if } Class\ 2 \end{cases}$$

then, apply the multi-linear function to the input variables X.

$$z = \left(\sum_{i=1}^{n} w_i x_i\right) + b$$

Here $xixi$ is the ith observation of X, $wi=[w1,w2,w3,\cdots,wm]$ is the weights or Coefficient, and b is the bias term also known as intercept. simply this can be represented as the dot product of weight and bias.

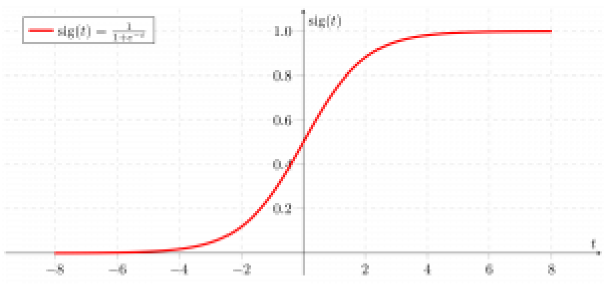
$$z = w \cdot X + b$$

whatever we discussed above is the linear regression.

Sigmoid Function

Now we use the sigmoid function where the input will be z and we find the probability between 0 and 1. i.e. predicted y.

$$\sigma(z) = \frac{1}{1-e^{-z}}$$



Sigmoid function

As shown above, the figure sigmoid function converts the continuous variable data into the probability i.e. between 0 and 1.

- $\sigma(z)$ tends towards 1 as $z \to \infty$
- $\sigma(z)$ tends towards 0 as $z \to -\infty$
- $\sigma(z)$ is always bounded between 0 and 1

where the probability of being a class can be measured as:

$$P(y=1) = \sigma(z)$$
$$P(y=0) = 1 - \sigma(z)$$

Logistic Regression Equation

The odd is the ratio of something occurring to something not occurring. it is different from probability as the probability is the ratio of something occurring to everything that could possibly occur. so odd will be:

$$\frac{p(x)}{1-p(x)} = e^z$$

Applying natural log on odd. then log odd will be:

$$\begin{aligned}
\log\left[\frac{p(x)}{1-p(x)}\right] &= z \\
\log\left[\frac{p(x)}{1-p(x)}\right] &= w \cdot X + b \\
\frac{p(x)}{1-p(x)} &= e^{w \cdot X + b} \quad \cdots \text{Exponentiate both sides} \\
p(x) &= e^{w \cdot X + b} \cdot (1 - p(x)) \\
p(x) &= e^{w \cdot X + b} - e^{w \cdot X + b} \cdot p(x)) \\
p(x) + e^{w \cdot X + b} \cdot p(x)) &= e^{w \cdot X + b} \\
p(x)(1 + e^{w \cdot X + b}) &= e^{w \cdot X + b} \\
p(x) &= \frac{e^{w \cdot X + b}}{1 + e^{w \cdot X + b}}
\end{aligned}$$

then the final logistic regression equation will be:

$$p(X;b,w) = \frac{e^{w\cdot X+b}}{1+e^{w\cdot X+b}} = \frac{1}{1+e^{-w\cdot X+b}}$$

Likelihood Function for Logistic Regression

The predicted probabilities will be:

- for y=1 The predicted probabilities will be: p(X;b,w) = p(x)
- for y = 0 The predicted probabilities will be: 1-p(X;b,w) = 1-p(x)

$$L(b,w) = \prod_{i=1}^{n} p(x_i)^{y_i}(1-p(x_i))^{1-y_i}$$

Taking natural logs on both sides

$$\begin{aligned}
\log(L(b,w)) &= \sum_{i=1}^{n} y_i \log p(x_i) \; + \; (1-y_i)\log(1-p(x_i)) \\
&= \sum_{i=1}^{n} y_i \log p(x_i) + \log(1-p(x_i)) - y_i \log(1-p(x_i)) \\
&= \sum_{i=1}^{n} \log(1-p(x_i)) + \sum_{i=1}^{n} y_i \log \frac{p(x_i)}{1-p(x_i} \\
&= \sum_{i=1}^{n} -\log 1 - e^{-(w\cdot x_i+b)} + \sum_{i=1}^{n} y_i(w\cdot x_i + b) \\
&= \sum_{i=1}^{n} -\log 1 + e^{w\cdot x_i+b} + \sum_{i=1}^{n} y_i(w\cdot x_i + b)
\end{aligned}$$

Gradient of the log-likelihood function

To find the maximum likelihood estimates, we differentiate w.r.t w,

$$\begin{aligned}
\frac{\partial J(l(b,w)}{\partial w_j} &= -\sum_{i=n}^{n} \frac{1}{1+e^{w\cdot x_i+b}} e^{w\cdot x_i+b} x_{ij} + \sum_{i=1}^{n} y_i x_{ij} \\
&= -\sum_{i=n}^{n} p(x_i;b,w)x_{ij} + \sum_{i=1}^{n} y_i x_{ij} \\
&= \sum_{i=n}^{n} (y_i - p(x_i;b,w))x_{ij}
\end{aligned}$$

What is Hypothesis representation in logistic regression?

The hypothesis function for logistic regression is represented using the sigmoid function, which maps the linear combination of the input features to a probability value between 0 and 1.

The hypothesis function for logistic regression is:

$$h_\theta(x) = g(\theta^T x)$$

Where:

- $h\theta(x)$ is the predicted probability that the output variable y is equal to 1 given the input features x and the model parameters θ
- g(z) is the sigmoid function, defined as: $g(z) = \frac{1}{1+e^{-z}}$
- $\theta^T x$ is the dot product of the model parameters θ and the input features x, representing the linear combination.

The sigmoid function maps any real-valued input z to a value between 0 and 1, which can be interpreted as the probability of the output variable y being 1.

Mathematically, the hypothesis function for logistic regression can be written as:

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

This hypothesis function allows logistic regression to model the probability of the binary output variable as a function of the input features. The model parameters θ are learned during the training process to minimize the cost function and make accurate predictions.

Explain decision boundary logistic regression.

Decision Boundary in Logistic Regression

In logistic regression, the decision boundary is a critical concept that defines the threshold for classifying instances into different classes based on the predicted probabilities. Here's a detailed explanation based on the provided sources:

1. **Sigmoid Function**:
    - Logistic regression uses the sigmoid function to map the linear combination of input features to a probability value between 0 and 1.
    - The sigmoid function is defined as:

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

2. **Decision Boundary**:
    - The decision boundary in logistic regression is the threshold value that separates the predicted probabilities into different classes.
    - Typically, if the predicted probability $h\vartheta(x)$ is greater than or equal to 0.5, the instance is classified as belonging to class 1; otherwise, it is classified as belonging to class 0.
3. **Geometric Interpretation**:
    - Geometrically, the decision boundary is a hyperplane that separates the feature space into regions corresponding to different classes.
    - The equation of the decision boundary is given by: $\theta^T x = 0$, where θ is the vector of model parameters and x is the input feature vector.
4. **Adjusting the Decision Boundary**:
    - The decision boundary can be adjusted based on the specific requirements of the problem or the desired trade-off between true positives and false positives.
    - By changing the threshold value from 0.5, the decision boundary can be shifted to optimize the model's performance for specific objectives.
5. **Visualization**:
    - In a two-dimensional feature space, the decision boundary in logistic regression is a line that separates the two classes.

For higher-dimensional feature spaces, the decision boundary becomes a hyperplane that separates the classes based on the predicted probabilities.

What is cost function for logistic regression?

In logistic regression, the cost function (also known as the loss function) is crucial for assessing the performance of the model and optimizing its parameters. Let's delve into the details of the cost function for logistic regression:

The cost function in logistic regression measures the disparity between the predicted probabilities and the actual binary labels. It quantifies how well the model is performing in terms of classifying instances into their correct categories.

Mathematical Formulation: The cost function for logistic regression is defined differently for the two possible classes (0 and 1). It is represented as:

$$\mathrm{Cost}(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if } y = 1 \\ -\log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$

Here:

- $h_\theta(x)$ represents the predicted probability that the instance belongs to class 1, calculated using the logistic function.
- y is the actual binary label of the instance

Explanation:

1. **Case 1 (y = 1)**:
    - If the true label y is 1 (indicating the positive class), the cost function penalizes the model proportionally to how far the predicted probability $h_\theta(x)$ is from 1.
    - As $h_\theta(x)$ approaches 1 (indicating high confidence in the positive class), the cost function tends towards 0.
    - However, as $h_\theta(x)$ deviates from 1 and approaches 0, the cost function increases exponentially.
2. **Case 2 (y = 0)**:
    - If the true label y is 0 (indicating the negative class), the cost function penalizes the model proportionally to how far the predicted probability $h_\theta(x)$ is from 0.
    - As $h_\theta(x)$ approaches 0 (indicating high confidence in the negative class), the cost function tends towards 0.
    - However, as $h_\theta(x)$ deviates from 0 and approaches 1, the cost function increases exponentially.

Purpose:

- The cost function serves as a measure of how well the model is fitting the training data.
- By minimizing the cost function, the model's parameters (weights) are adjusted to improve the accuracy of the predictions.
- Optimization techniques like gradient descent are commonly used to minimize the cost function and optimize the model's performance.

Explain Gradient Descent for Logistic Regression.

Gradient Descent is an optimization algorithm used to minimize the cost function in logistic regression. It is an iterative process that updates the model parameters to minimize the difference between the predicted probabilities and the actual outcomes. Here's how it works:

1. **Initialization**: The algorithm starts by initializing the model parameters, typically with random values.
2. **Cost Function**: The cost function for logistic regression is defined as the negative log-likelihood of the model. It measures the difference between the predicted probabilities and the actual outcomes.
3. **Gradient Calculation**: The gradient of the cost function is calculated with respect to each model parameter. This gradient represents the direction of the steepest descent of the cost function.
4. **Parameter Update**: The model parameters are updated by moving in the opposite direction of the gradient. This is done by subtracting the product of the learning rate and the gradient from the current parameter value.
5. **Iteration**: Steps 2-4 are repeated iteratively until the model converges or a stopping criterion is reached.

Key Points:

- **Learning Rate**: The learning rate controls the step size of each update. A high learning rate can lead to overshooting the minimum, while a low learning rate can lead to slow convergence.
- **Convergence**: The algorithm converges when the cost function stops decreasing or when a stopping criterion is reached.
- **Optimization**: Gradient Descent is used to optimize the model parameters to minimize the cost function. This is crucial in logistic regression as it helps in finding the best parameters that maximize the likelihood of the model.
- **Implementation**: Gradient Descent can be implemented in various programming languages, including Python, R, and MATLAB. The implementation typically involves defining the cost function, calculating the gradient, and updating the model parameters.

Explain Naïve Bayes Classifier

Naïve Bayes classifiers belong to a family of algorithms based on Bayes’ Theorem. Despite their simplified assumption of feature independence, they are widely employed in machine learning due to their efficiency and effectiveness. Let's dive into understanding the theory, implementation, advantages, and applications of Naïve Bayes classifiers.

- Naïve Bayes classifiers encompass a collection of classification algorithms rooted in Bayes’ Theorem. Although not a singular algorithm, they share a common principle—every pair of features being classified is presumed to be independent of each other. These classifiers are extensively used in various fields, including text classification, spam filtering, sentiment analysis, and medical diagnosis.

Why it's Called Naïve Bayes: The term "naïve" signifies the simplistic assumption made by these classifiers: that features are conditionally independent given the class label. This assumption simplifies computations but may not hold true in real-world scenarios. The "Bayes" part of the name pays homage to Reverend Thomas Bayes, an 18th-century statistician who formulated Bayes’ theorem.

Assumptions of Naïve Bayes:

1. **Feature Independence:** Naïve Bayes assumes that each feature is independent of the others, given the class label.
2. **Continuous Features:** Continuous features are assumed to follow a normal distribution within each class.
3. **Discrete Features:** Discrete features are assumed to have a multinomial distribution within each class.
4. **Equal Feature Importance:** All features are considered equally important in contributing to the prediction.
5. **No Missing Data:** The data should not contain any missing values.

Advantages of Naïve Bayes Classifier:

- **Ease of Implementation:** Naïve Bayes is simple to implement and computationally efficient.
- **Effective with Large Feature Sets:** It performs well even with a large number of features.
- **Suitable for Limited Training Data:** It works effectively even with limited training data.
- **Effective with Categorical Features:** Performs well when dealing with categorical features.

- **Assumptions for Numerical Features:** For numerical features, data is assumed to follow normal distributions.

Disadvantages of Naïve Bayes Classifier:

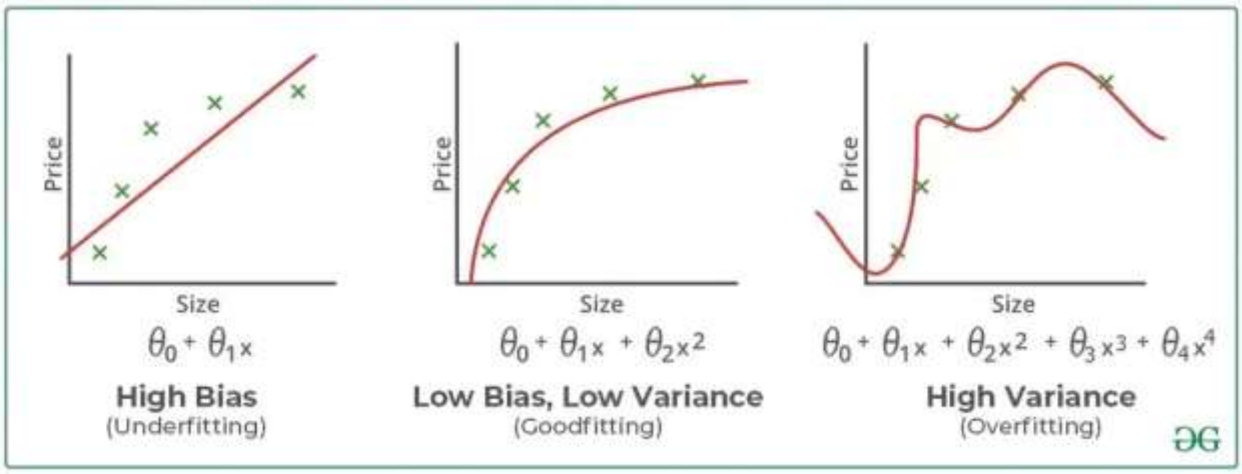
- **Assumption of Feature Independence:** The assumption of feature independence may not hold in real-world data.
- **Sensitivity to Irrelevant Attributes:** Naïve Bayes can be influenced by irrelevant attributes.
- **Zero Probability for Unseen Events:** It may assign zero probability to unseen events, leading to poor generalization.

Applications of Naïve Bayes Classifier:

- **Spam Email Filtering:** Classifies emails as spam or non-spam based on features.
- **Text Classification:** Used in sentiment analysis, document categorization, and topic classification.
- **Medical Diagnosis:** Helps predict the likelihood of a disease based on symptoms.
- **Credit Scoring:** Evaluates creditworthiness of individuals for loan approval.
- **Weather Prediction:** Classifies weather conditions based on various factors.

What is Over fitting & Under fitting

In machine learning, overfitting and underfitting are two common issues that can occur during the training of a model. These issues affect the model's performance and its ability to generalize to new, unseen data.



Overfitting

Overfitting occurs when a model is too complex and is able to fit the training data too closely. This means that the model is able to memorize the training data rather than learning the underlying patterns and relationships. As a result, the model performs well on the training data but poorly on new, unseen data.Overfitting can be caused by several factors, including:

1. **High model complexity**: Models with too many parameters or layers can easily overfit the training data.
2. **Insufficient training data**: If the training data is limited, the model may not have enough information to generalize well.
3. **High variance**: Models with high variance are more prone to overfitting.

Underfitting

Underfitting occurs when a model is too simple and is unable to capture the underlying patterns and relationships in the data. This means that the model is unable to fit the training data well and performs poorly on both the training and new data.Underfitting can be caused by several factors, including:

1. **Low model complexity**: Models with too few parameters or layers can easily underfit the training data.
2. **Insufficient training data**: If the training data is limited, the model may not have enough information to learn from.
3. **High bias**: Models with high bias are more prone to underfitting.

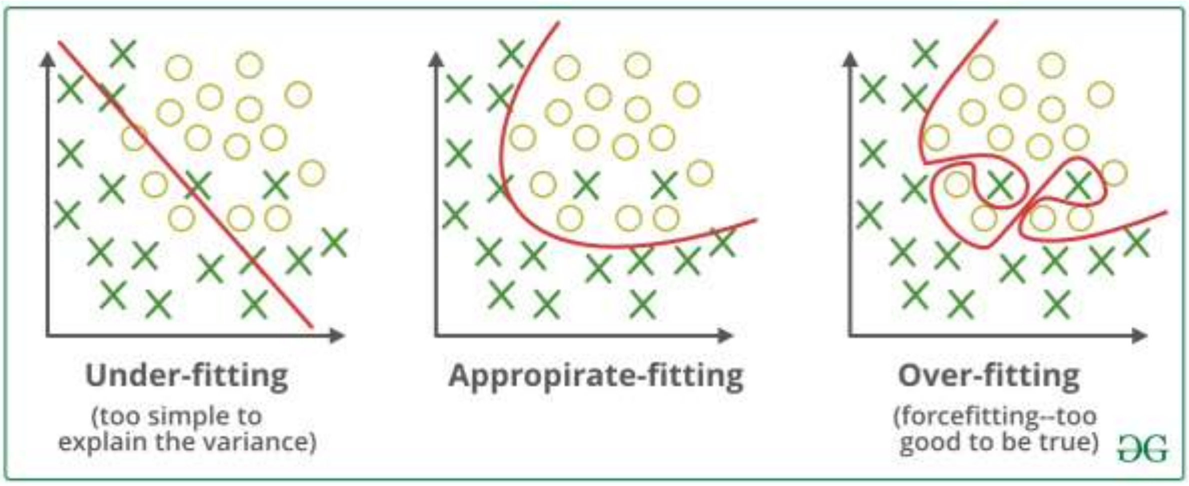
Consequences of Overfitting and Underfitting

Overfitting and underfitting can have significant consequences on the performance of a model. Overfitting can lead to poor performance on new data, while underfitting can result in a model that is unable to learn from the data.

Techniques to Avoid Overfitting and Underfitting

Several techniques can be used to avoid overfitting and underfitting, including:

1. **Regularization**: Regularization techniques, such as L1 and L2 regularization, can be used to reduce the complexity of a model and prevent overfitting.
2. **Early stopping**: Early stopping involves stopping the training process when the model's performance on the validation set starts to degrade.
3. **Data augmentation**: Data augmentation involves generating additional training data by applying transformations to the existing data. This can help to reduce overfitting.
4. **Model selection**: Model selection involves selecting the best model based on its performance on the validation set. This can help to avoid overfitting and underfitting.
5. **Cross-validation**: Cross-validation involves splitting the data into multiple folds and training the model on each fold. This can help to evaluate the model's performance on unseen data and avoid overfitting.


Under-fitting
(too simple to explain the variance)
Appropirate-fitting
Over-fitting
(forcefitting--too good to be true)

Explain instance based classifier.

An instance-based classifier is a type of machine learning algorithm that uses existing instances or examples to classify new, unseen data. This approach is also known as the "nearest neighbor" or "k-nearest neighbor" (k-NN) algorithm. Here's how it works:

1. **Training**: The algorithm is trained on a set of labeled instances or examples, where each instance is described by a set of features or attributes.
2. **Query**: When a new, unseen instance is presented, the algorithm calculates the distance between the query instance and each of the training instances.
3. **Classification**: The algorithm then classifies the query instance by determining which class is most common among the k nearest neighbors. This is done by counting the number of instances in each class among the k nearest neighbors and selecting the class with the highest count.
4. **k-NN**: The choice of k is crucial in determining the performance of the algorithm. A higher value of k can lead to more robust results but may also increase the computational complexity. A lower value of k can lead to more accurate results but may also be more susceptible to noise in the data.
5. **Distance Metrics**: The distance metric used to calculate the distance between instances can significantly impact the performance of the algorithm. Common distance metrics include Euclidean distance, Manhattan distance, and Minkowski distance.
6. **Advantages**: Instance-based classifiers have several advantages, including:
    - **Simple to implement**: The algorithm is relatively simple to implement, especially for small datasets.
    - **No need for feature engineering**: The algorithm does not require feature engineering or preprocessing of the data.
    - **Robust to noise**: The algorithm can be robust to noise in the data, as it relies on the majority vote of the nearest neighbors.
7. **Disadvantages**: Instance-based classifiers also have some disadvantages:
    - **Computational complexity**: The algorithm can be computationally expensive for large datasets.
    - **Sensitive to k**: The choice of k can significantly impact the performance of the algorithm.
    - **Sensitive to distance metric**: The choice of distance metric can also impact the performance of the algorithm.

Example

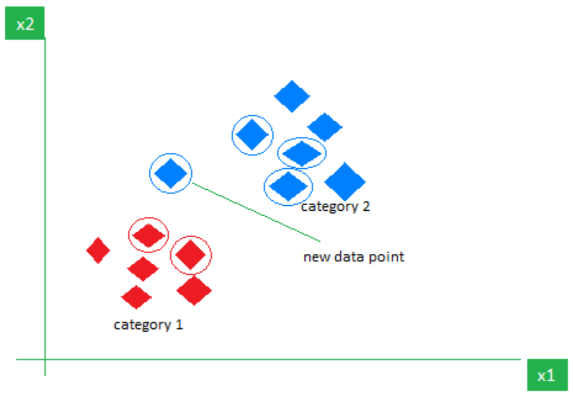
Suppose we have a dataset of images of different animals, where each image is described by features such as color, shape, and size. We want to classify a new image as either a cat or a dog. We can use an instance-based classifier by training the algorithm on a set of labeled images and then using the algorithm to classify the new image based on the distances between the new image and the training images.

Explain K- Nearest Neighbor Classifier

K-Nearest Neighbor (KNN) Classifier

The K-Nearest Neighbor (KNN) algorithm, developed by Evelyn Fix and Joseph Hodges in 1951 and later expanded by Thomas Cover, is a supervised machine learning method used for classification and regression tasks. Below, we explore its fundamentals, operational mechanisms, implementation, advantages, and disadvantages.

Definition: KNN is a fundamental classification algorithm in machine learning, belonging to the domain of supervised learning. It finds extensive application in pattern recognition, data mining, and intrusion detection. Unlike parametric algorithms like Gaussian Mixture Models (GMM), KNN is non-parametric, meaning it does not impose assumptions about the distribution of data. Instead, it utilizes prior training data to classify new data points into groups based on their attributes.



Intuition Behind KNN Algorithm: The intuition behind KNN lies in identifying clusters or groups within the data. Given a new, unclassified data point, KNN assigns it to a group based on the classes of its nearest neighbors. This process operates on the premise that a point's proximity to a cluster of similarly classified points increases the likelihood of it belonging to the same class.

Why Do We Need KNN Algorithm? KNN is valued for its simplicity, ease of implementation, and flexibility. It doesn't require assumptions about the data distribution and can handle both numerical and categorical data. Additionally, it's less sensitive to outliers compared to other algorithms. KNN works by finding the K nearest neighbors to a given data point based on a chosen distance metric (e.g., Euclidean distance), and then determining the class or value based on the majority vote or average of these neighbors.

Distance Metrics Used in KNN Algorithm: KNN relies on distance metrics to determine the proximity of data points. Common distance metrics include:

- **Euclidean Distance:** Cartesian distance between two points in a plane/hyperplane.
- **Manhattan Distance:** Sum of absolute differences between coordinates of points in n-dimensions.

- **Minkowski Distance:** A generalized distance metric that includes both Euclidean and Manhattan distance as special cases.

Choosing the Value of K: The value of K in KNN determines the number of neighbors considered during prediction. It's crucial to choose an appropriate K value based on the dataset's characteristics. Higher values of K may be suitable for datasets with more noise or outliers. Cross-validation methods can aid in selecting the optimal K value.

Operational Mechanism of KNN:

1. **Selecting the Optimal Value of K:** Choose the number of nearest neighbors (K) for prediction.
2. **Calculating Distance:** Measure similarity between the target and training data points using a chosen distance metric (e.g., Euclidean distance).
3. **Finding Nearest Neighbors:** Identify the K data points with the shortest distances to the target point.
4. **Voting for Classification or Taking Average for Regression:** For classification tasks, determine the class label by majority voting among the K nearest neighbors. For regression tasks, calculate the average or weighted average of values of the K nearest neighbors.

Advantages of KNN Algorithm:

- **Ease of Implementation:** Simple to implement with low algorithmic complexity.
- **Adaptability:** Easily adjusts to new examples added to the dataset, making it suitable for dynamic datasets.
- **Few Hyperparameters:** Requires minimal hyperparameters, mainly the choice of K and distance metric.

Disadvantages of KNN Algorithm:

- **Scalability:** Computationally expensive and resource-intensive, especially for large datasets.
- **Curse of Dimensionality:** Faces challenges in high-dimensional spaces, affecting classification accuracy.
- **Prone to Overfitting:** Vulnerable to overfitting due to high dimensionality, necessitating feature selection or dimensionality reduction techniques.

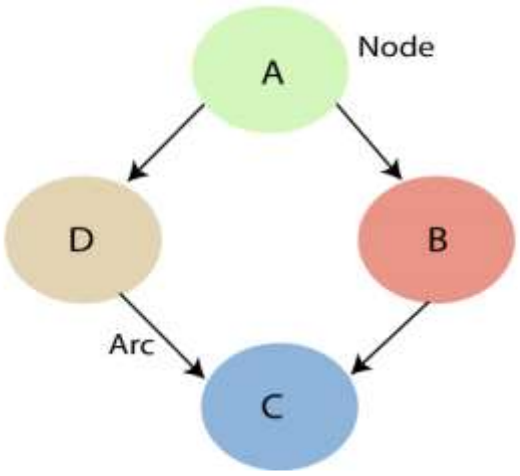
Explain Bayesian Network

Bayesian Network

A Bayesian network, also known as a Bayes network, belief network, decision network, or Bayesian model, is a probabilistic graphical model used to represent a set of variables and their conditional dependencies using a directed acyclic graph (DAG). It is a key technology in artificial intelligence for dealing with probabilistic events and solving problems involving uncertainty.

Components:

1. **Directed Acyclic Graph (DAG):** The graphical structure of a Bayesian network consists of nodes and directed links (arcs). Each node corresponds to a random variable, which can be either continuous or discrete. The directed arcs represent the causal relationships or conditional probabilities between random variables. The absence of a directed link between two nodes indicates that they are independent of each other.



2. **Table of Conditional Probabilities:** Along with the DAG, a Bayesian network includes conditional probability tables (CPTs) associated with each node. These tables specify the conditional probability distribution of each node given its parent nodes in the graph.

Operational Mechanism:

1. **Joint Probability Distribution:** Bayesian networks are based on joint probability distributions, which represent the probabilities of different combinations of variables. The joint probability distribution can be decomposed using conditional probabilities based on the structure of the Bayesian network.

2. **Conditional Probabilities:** The conditional probability of each node given its parent nodes is calculated using the conditional probability tables associated with each node in the network. This allows for inference and probabilistic reasoning about the variables in the network.

Example: Let's illustrate the concept of a Bayesian network with an example:

Scenario: Harry installed a new burglar alarm at his home to detect burglaries. The alarm responds reliably to burglaries but also responds to minor earthquakes. Harry's neighbors, David and Sophia, have taken the responsibility to inform Harry at work when they hear the alarm. However, David sometimes confuses the alarm with the phone ringing, and Sophia occasionally misses hearing the alarm due to listening to loud music.

Question: What is the probability of a burglary given that the alarm has been activated?

Solution: We can construct a Bayesian network to represent this scenario:

- Nodes: Burglary, Earthquake, Alarm, DavidCalls, SophiaCalls.
- Arcs: Burglary → Alarm, Earthquake → Alarm, Alarm → DavidCalls, Alarm → SophiaCalls.

We can then use the conditional probability tables associated with each node to compute the probability of a burglary given that the alarm has been activated, using Bayesian inference techniques.

Applications: Bayesian networks find applications in various fields, including:

- Prediction
- Anomaly detection
- Diagnostics
- Automated insight
- Reasoning
- Time series prediction
- Decision making under uncertainty