UNIT-III: Models Based on Decision Trees

Decision Trees for Classification

Decision trees are a popular supervised machine learning algorithm used for both classification and regression, where classification trees predict categorical outcomes by following a tree-like structure of decisions based on data features. Here's a more detailed explanation:

Key Concepts:

Tree Structure:

Decision trees are visualized as a tree, with:

Root Node: The starting point of the tree.

Internal Nodes: Represent features or attributes used for making decisions.

Branches: Represent possible outcomes or values of the feature.

Leaf Nodes: Represent the final classification or prediction.

Classification:

In classification, each leaf node represents a class label, and the algorithm classifies an instance by following the branches from the root to a leaf node.

Supervised Learning:

Decision trees are a type of supervised learning algorithm, meaning they learn from labeled data to make predictions.

Recursive Partitioning:

Decision trees work by recursively partitioning the data into subsets based on feature values, creating a tree structure that represents the decision rules.

Advantages:

Interpretability: Decision trees are relatively easy to understand and interpret, making them suitable for explaining predictions.

Handles both numerical and categorical data: Decision trees can handle both types of data without requiring much preprocessing.

Can handle high-dimensional data: Decision trees can handle a large number of features with good accuracy.

Disadvantages:

Overfitting: Decision trees can be prone to overfitting, meaning they learn the training data too well and perform poorly on new, unseen data.

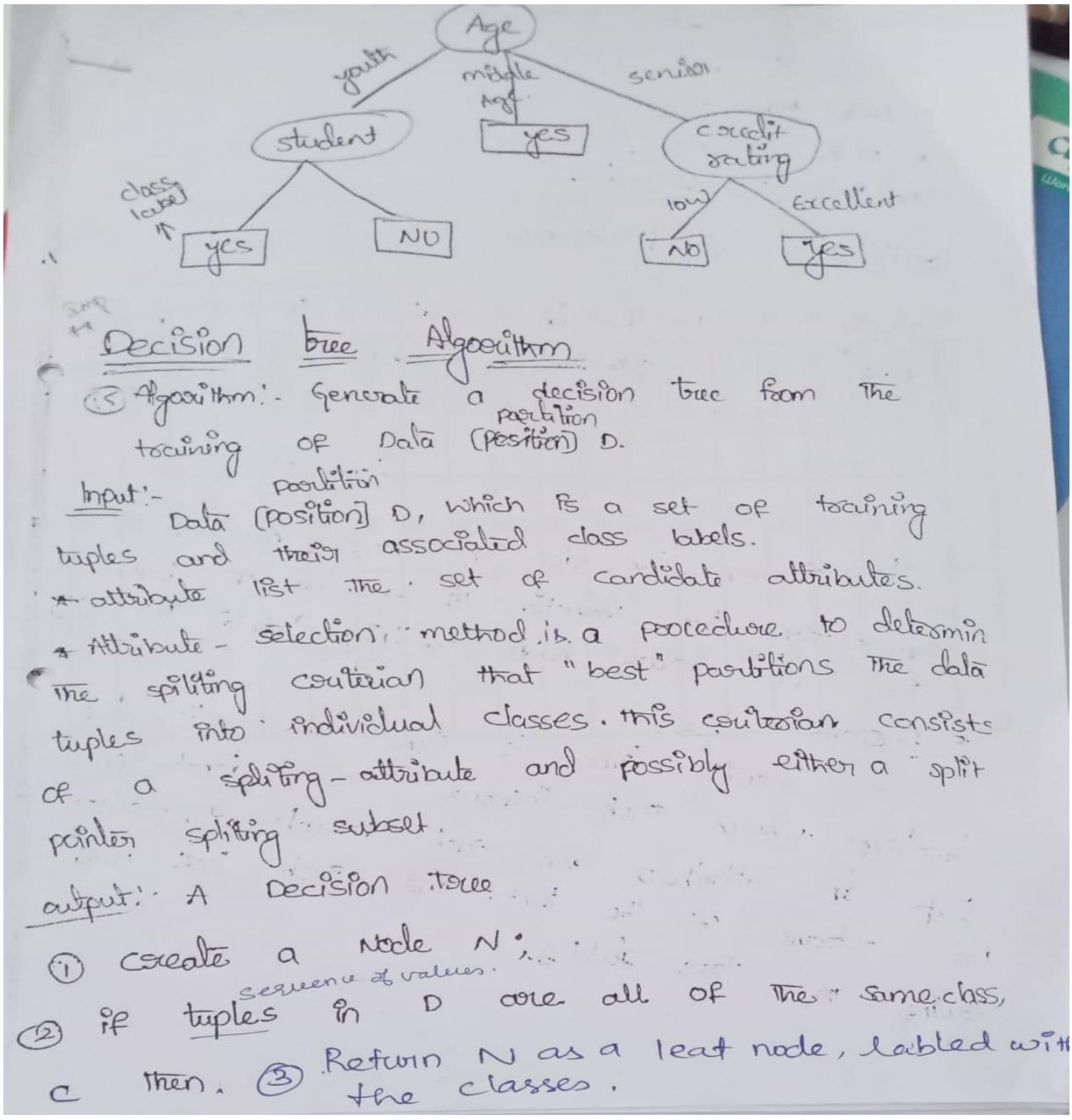
Sensitivity to small variations in data: Small changes in the training data can lead to significant changes in the tree structure.

Ensemble Methods:

To address the limitations of individual decision trees, ensemble methods like Random Forests and Gradient Boosting are often used, which combine multiple trees to improve accuracy and robustness.

Decision Tree Classifier is a class capable of performing multi-class classification on a dataset.

As with other classifiers, **DecisionTreeClassifier** takes as input two arrays: an array X, sparse or dense, of



classes.

if attribute-list is empty then

return N as a leaf node labeled with the majority class in D; // majority voting.

apply attribute-selection-method (D, attribute-list) to find the "best" splitting-criterion

label node N with splitting-criterion

if splitting-attribute is discrete-valued and multi-way splits allowed then

attribute-list ← attribute-list - splitting-attribute

for each outcome j of splitting-criterion.

let D_j be the set of data tuples in D satisfying outcome j;

if D_j is empty then

attach a leaf labeled with the majority class in D to node N;

else attach the node returned by generate decision-tree (D_j, attribute-list) to node N;

end for

return N;

Bayesian classification:-

Bayesian classifiers are statistical classifiers. They can predict class membership probabilities such as the probability that a given sample belongs to a particular class.

Bayesian classification is based on Bayes theorem.

Bayes theorem:

$$P(c|x) = \frac{P(x|c)\,P(c)}{P(x)}$$

find the posterior probability of a class conditional.

Algorithm:-

step 1:-
* let 'D' be a training dataset of data tuple associated with class labels.

step 2:-
* each tuple is represented by an n-dimensional vector $x = (x_1, x_2 \ldots x_n)$ where $x_1, x_2 \ldots x_n$ are values of attributes.

* Suppose that there are m no. of classes $C_1, C_2 \ldots C_m$ given a data tuple x the classifier is predict that $x \in$ the class having the highest posterior probability condition on x.

* This can be written mathematically where $j = 1, 2, \ldots m$ and $i \neq j$.

i.e., we maximize $P(C_i|x)$

Bayes theorem $P(C_i|x) = \frac{P(x|C_i)\,P(C_i)}{P(x)}$

step 3:-
As $P(x)$ is constant for all classes it is enough to maximize only that is numerical function as $P(x|C_i)$

If the class probability is not known

$P(C_1) = P(C_2) = \cdots = P(C_m)$

so it is enough to maximize only $P(x/C_i)$

step 4:-

$$P(x/C_i) = \prod_{k=1}^{m} P(x_k/C_i)$$

$$= P(x_1/C_i) \times P(x_2/C_i) \times \cdots \times P(x_m/C_i)$$

(i) If A_k categorical $P(x_k/C_i)$ is number of tuples of class C_i in 'D' having the value small S_k for A_k / no. of tuples of C_i in D.

(ii) A_k is continuous $g(x, \mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma C_i} \times e^{\frac{-(x-\mu)^2}{2\sigma^2}}$

where $g(x, \mu, \sigma)$ is the Gaussian (normal) density function for attribute A_k while μ and σ are the mean and standard deviation, respectively given the values for attributes A_k for training samples of class C.

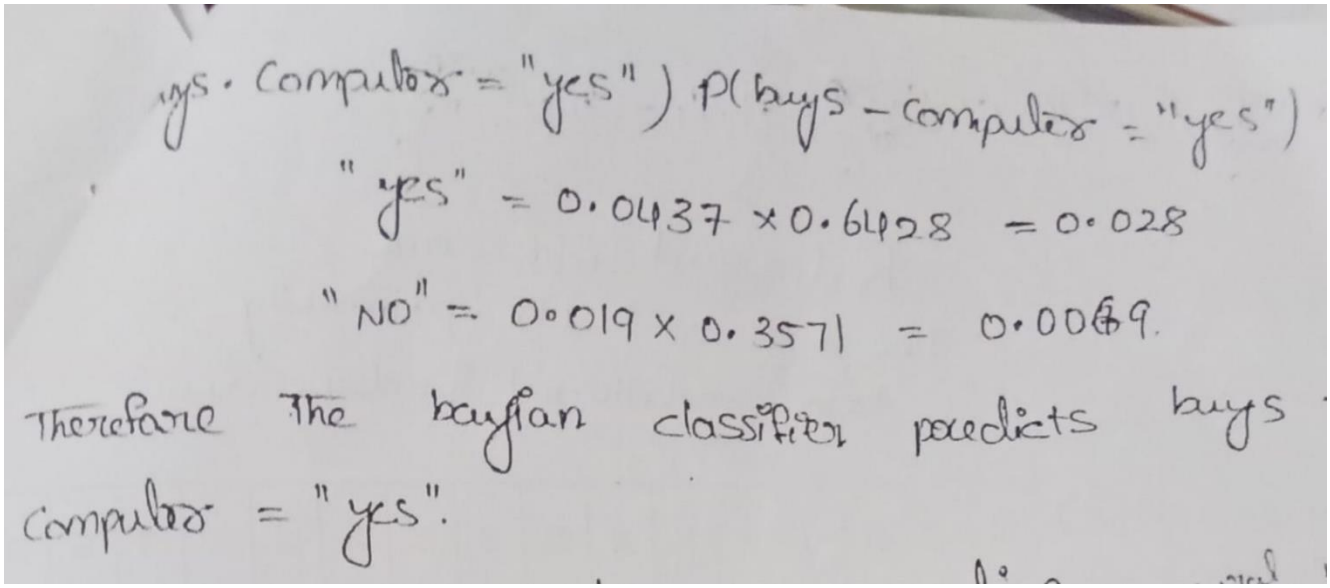
step 5:-

The Classifier predict the class label is C_i if and only if $P(x/C_i) P(C_i) > P(x/C_j) \cdot P(C_j)$ for $1 \le j \le m$, $j \ne i$

	Age	Income	student	credit rating	Computer
1	youth	high	NO	Avg	NO
2	"	"	"	Exce	"
3	middle	"	"	Avg	yes
4	senior	medium	"	"	"
5	senior	low	yes	"	"
6	senior	"	yes	Exce	NO
7	middle	"	yes	"	yes
8	youth	medium	yes NO	Avg	NO
9	youth	low	NO yes	"	yes
10	senior	medium	yes	"	"
11	youth	"	"	Exce	"
12	middle	"	NO.	Exce	"
13	middle	high	yes	Avg	yes
14	senior	medium	NO.	Exce	NO

$X = (age = "<= 30",\ income = "medium",\ student = "yes",$
$credit\ rating = "fair")$ predict the class labeled data tuple 'x' using bayesian classification algorithm.

Sol :- The prior probability of each class.

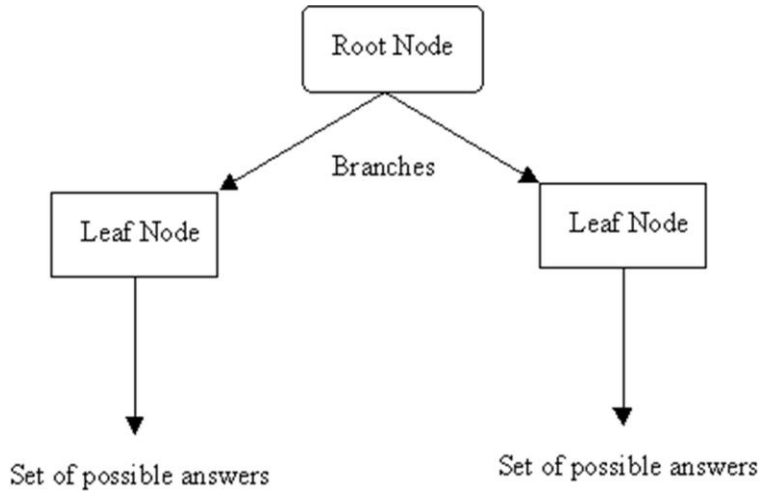


Impurity Measures

Impurity measures are used in Decision Trees just like squared loss function in linear regression. We try to arrive at as lowest impurity as possible by the algorithm of our choice. Impurity is presence of more than one class in a subset of data.

So all below mentioned measures differ in formula but align in goal. Watch till the end to know secret highlights of this topic.

Remember this



Make sure you understand that impurity measure is calculated for each leaf node, and its weighted average is the corresponding impurity measure for root node, based on which we say that this feature would become decision feature or not.

Let's take an example with Entropy and solve to see the exact formulation.

Entropy

The formula for impurity at leaf node isk

$$E(S) = \sum_{1}^{c} -p_i \, log_2 \, p_i$$

After taking weighted average for a feature, we need to check if this feature brings the most reduction in impurity. While using Entropy we do this by *Information Gain*

$$Information\ Gain = E(Y) - E(Y|X))$$

where
E(Y) should be Entropy before splitting the data over X E(Y|X) is Weighted Entropy after split over X Example: Consider the Contingency Table asdvv

Credit Rating	Liability		
	Normal	High	Total
Excellent	3	1	4
Good	4	2	6
Poor	0	4	4
Total	7	7	14

This should be read as simply Horizontal division is of Liability class labels (Normal and High), while vertical division is that of Credit Rating (Excellent, Good, Poor). So the number 3 in table implies, that out of total 14 companies there were 3 companies which got 'Excellent rating' and had 'Normal Liability'.

Calculation of Entropy for deciding if Credit Rating should be the first split. First calculate Entropy before splitting

$$E(\textit{Liability}) = -\frac{7}{14}\log_2\left(\frac{7}{14}\right) - \frac{7}{14}\log_2\left(\frac{7}{14}\right)$$

$$= -\frac{1}{2}\log_2\left(\frac{1}{2}\right) - \frac{1}{2}\log_2\left(\frac{1}{2}\right)$$

$$= 1$$

Next entropy over each Leaf node and then weighted average over credit rating split

$$E(\textit{Liability} \mid CR = \textit{Excellent}) = -\frac{3}{4}\log_2\left(\frac{3}{4}\right) - \frac{1}{4}\log_2\left(\frac{1}{4}\right) \approx 0.811$$

$$E(\textit{Liability} \mid CR = \textit{Good}) = -\frac{4}{6}\log_2\left(\frac{4}{6}\right) - \frac{2}{6}\log_2\left(\frac{2}{6}\right) \approx 0.918$$

$$E(\textit{Liability} \mid CR = \textit{Poor}) = -0\log_2(0) - \frac{4}{4}\log_2\left(\frac{4}{4}\right) = 0$$

$$\textit{Weighted Average}:$$

$$E(\textit{Liability} \mid CR) = \frac{4}{14} \times 0.811 + \frac{6}{14} \times 0.918 + \frac{4}{14} \times 0$$

$$= 0.625$$

Note greater the entropy, worse is the current feature for split at present level.
We now calculate information gain(Higher the better, or lower the conditional entropy)

$$\textit{Inf ormation Gain}:$$

$$IG(\textit{Liability}, CR) = E(\textit{Liability}) - E(\textit{Liability} \mid CR)$$

$$= 1 - 0.625$$

$$= 0.375$$

So we get 0.375 as the IG from Credit Rating as the metric for classification of data over liability status. If we had suppose stock price as a independent feature, we would have done the same thing for it as well. Then we would have compared the result for both, and one with higher information gain would have been our first decision variable for splitting.
Now
Impurity Reduction = G(Y) — G(Y|X))

Gini Index
The formula for leaf node is

$$G(S) = 1 - \sum_{1}^{c} p_i^2$$

After weighted average just like above, we calculate

$$Impurity\ Reduction = G(Y) - G(Y|X))$$

And one offering highest reduction is chosen as decision variable for splitting.

Classification Error

The formula of leaf node is

$$Classification\ Error = 1 - Max\ p_i$$

Often this is a rarely
used one. Another less
heard used ones are

Gain Ratio

The gain ratio "normalizes" the information gain

$$\textbf{Gain Ratio} = \frac{\textbf{Information gain } (\nabla)}{\textbf{Entropy}}$$

Impurity measures such as entropy and Gini Index tend to favor attributes that have large number of distinct values. Therefore Gain Ratio is computed which is used to determine the goodness of a split. Every splitting criterion has their own significance and usage according to their characteristic and attributes type.

Towing Criteria

The Gini Index may encounter problems when the domain of the target attribute is relatively wide. In this case it is possible to employ binary criterion called towing

$$\textbf{Twoing Criteria (t)} = \frac{P_L P_R}{4}\left(\sum \left(|p\ (i/t_L) - p\ (i/t_R)|\right)\right)^2$$

criteria.

This criterion is defined as:

Where, p (i/t) denote the fraction of records belonging to class i at a given node t

Twoing Criterion for Multiclass Problem

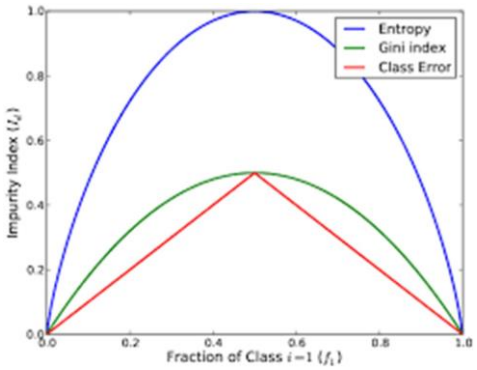
- Classes are numbered {1,2…,j}
- At each node group the classes into two subsets
  - e.g. In the 10 class gym example the two subsets identified by the twoing root node splitter were

  –C1={1,4,6,7,9,10} C2={2,3,5,8}
- Then the best split for separating these two groups is found
  - The process can be repeated for all possible groupings– best overall is the selected splitter
  - Same as GINI for a binary dependent variable

less I could find about it, have a look at this for more understanding.

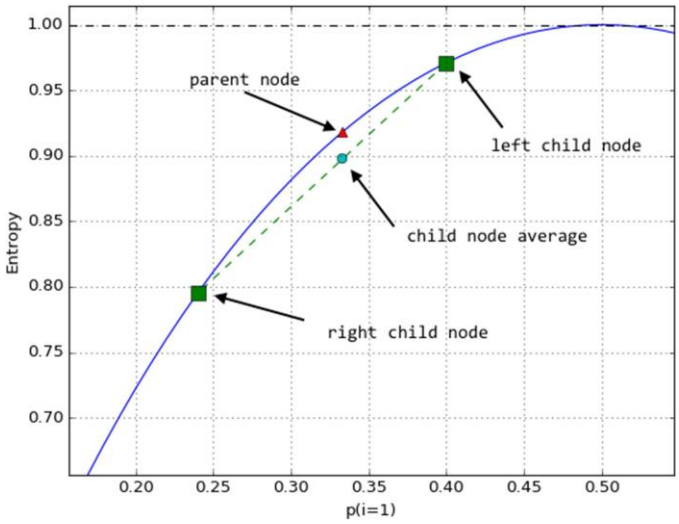
Highlights:

Binary classification: These are primarily used for Binary split, i.e. two leaf nodes, however when multilevel split is there, we can convert them to Binary split like eg. for color(R, G, B) as R or G, B or G, R or B as splitting decision Impurity Index(like Information Gain, Gini Index)



are concave functions, and we need to maximize the reduction in impurity. Note as below, graphically also they are Convex Functions.

3.Shapes of the above measures: Continuing from above figure the Impurity Index optimize the choice of feature for splitting but following different paths. Note Classification Error gives a straight-line curve as opposed to Entropy or Gini Index.

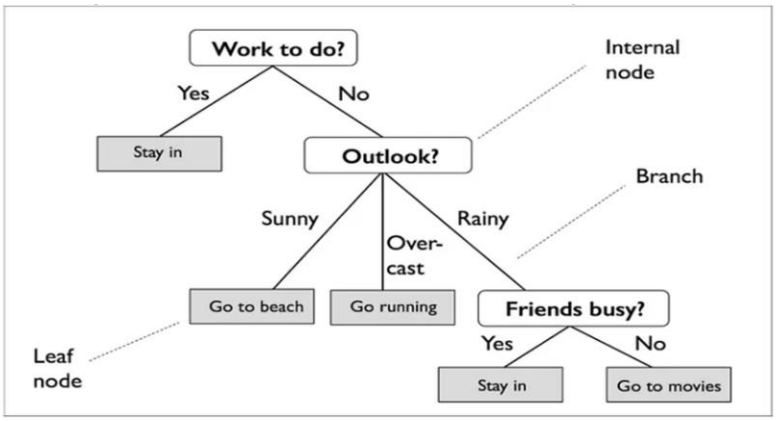


From this figure, we are trying to compare Entropy(or Gini) method with Classification Error. We are trying to compare Impurity before(Red mark)and after(Green dot) splitting. We want the vertical distance between these two points to maximize, so graphically it is quite intuitive that Classification error being straight Line leaves no space between these two points, however Entropy provides greater space over Gini Index.

4.Difference in use when numerical features instead of categorical: Categorical is easy to follow, while in Numerical our work is just to find average of two corresponding observations (arranged in ascending) and then check the split's entropy reduction taking each such average as a cutoff. The one providing the max reduction in impurity is chosen as cutoff value.

❖ REGRESSION BASED ON DECISION TREES:

Decision Tree Regression is a machine learning technique used to predict continuous numerical values by constructing a tree-like model. It works by splitting data based on features, creating nodes and branching paths until reaching leaf nodes that represent final predictions. These predictions can be the average value of the target variable within that leaf or a function mapping from the feature space to the target value.



Key Concepts:

Tree-like Structure:

The model is structured like a tree, with internal nodes representing features and decision points, and leaf nodes representing predictions.

Splitting Data:

The algorithm splits the data based on features, aiming to create subsets where the target variable has the least variability.

Leaf Node Predictions:

Predictions are made at the leaf nodes, which can be the average target value within that leaf or a function of the features.

How it Works:

1. **Data Splitting:**

The algorithm starts with a root node containing the entire dataset. It then iteratively splits the data based on the best feature and split point, aiming to maximize the reduction in variance or other suitable impurity measures.

2. **Node Creation:**

Each split creates new nodes (child nodes) and branches, representing the split conditions.

3. **Recursive Splitting:**

This process continues recursively until a stopping criterion is met, such as reaching a maximum tree depth or when no further splits significantly improve the model.

4. **Leaf Node Predictions:**

The final leaf nodes represent the predicted values for the corresponding data subsets.

Benefits:

Interpretability:

Decision trees are relatively easy to understand and interpret, making them useful for explaining prediction models.

Non-linear Relationships:

They can capture non-linear relationships in data, unlike linear regression models.

Feature Importance:

The algorithm can identify the importance of different features in making predictions.

Limitations:

Overfitting:

Decision trees can be prone to overfitting, especially if the tree is too deep or complex. Pruning or using ensemble methods can help mitigate this.

Sensitivity to Data:

They can be sensitive to changes in the data, and the model can be unstable.

Discrete Output:

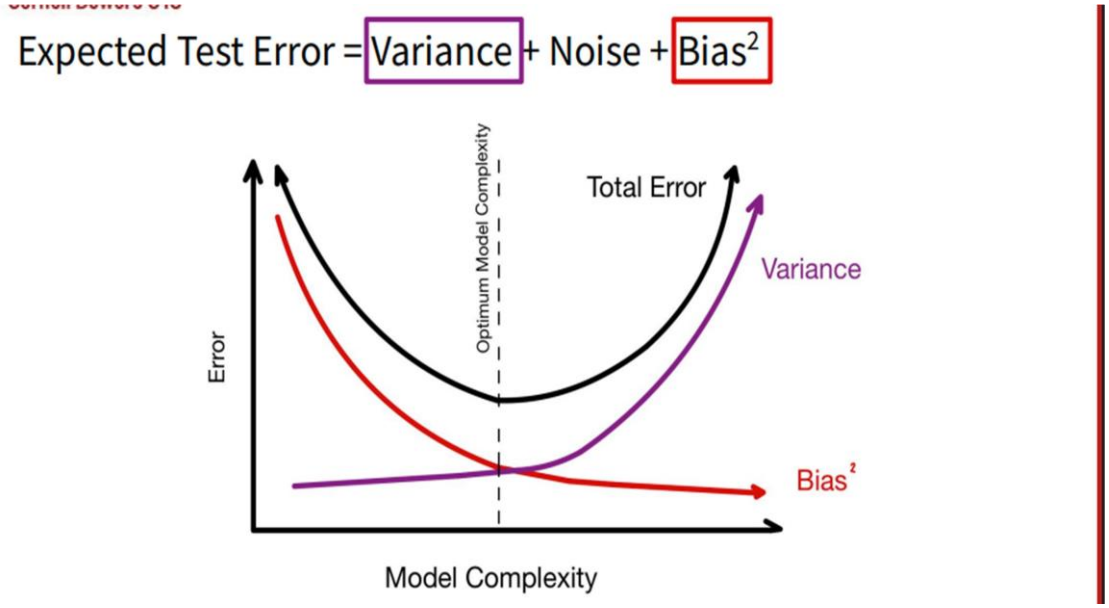
While decision trees can be used for regression, they are not ideal for continuous target variables.

❖ **BIAS–VARIANCE TRADE-OFF**

The bias-variance trade-off in machine learning refers to the inherent tension between a model's ability to fit the training data (low bias) and its ability to generalize to unseen data (low variance). The goal is to find a model that strikes a balance between these two, minimizing overall prediction error.

Here's a more detailed explanation

:



Understanding Bias and Variance:

Bias:

Refers to the error introduced by making simplifying assumptions about the data or the model's structure. High bias means the model is too simple and cannot capture the underlying patterns in the data, leading to underfitting.

Variance:

Refers to the model's sensitivity to variations in the training data. High variance means the model is too complex and has learned the noise in the training data, leading to overfitting.

The Trade-off:

High Bias, Low Variance:

A model with high bias is simple and consistent, but it may not accurately represent the data.

Low Bias, High Variance:

A model with low bias is complex and can fit the training data well, but it may not generalize well to unseen data.

The Goal:

The goal is to find the sweet spot where the model is complex enough to capture the underlying patterns without overfitting to the noise in the training data.

Generalization:

The ability of a model to perform well on unseen data is crucial for real-world applications.

Overfitting and Underfitting:

Understanding the bias-variance trade-off helps avoid these common problems in machine learning.

Model Selection:

It guides the choice of model complexity and regularization techniques to achieve optimal performance.

Examples:

Underfitting:

A linear regression model might have high bias and low variance if the relationship between the variables is not linear.

Overfitting:

A complex polynomial regression model might have low bias and high variance if it perfectly fits the training data but performs poorly on new data.

Finding the Balance:

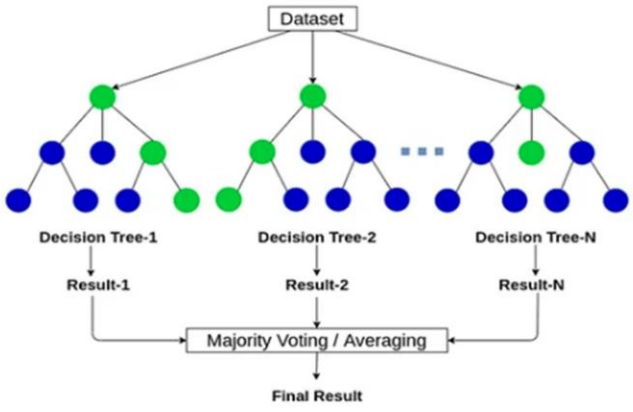
Using techniques like regularization or cross-validation can help find the right balance between bias and variance.

In summary, the bias-variance trade-off is a fundamental concept in machine learning that helps us build models that generalize well to unseen data by finding the optimal balance between model complexity and its ability to fit the training data.

RANDOM FORESTS FOR CLASSIFICATION AND REGRESSION:

Random forests are an ensemble learning method used for both classification and regression tasks, employing multiple decision trees to make predictions, with the final prediction being the mode of the classes (classification) or the average of the predictions (regression).

Random Forest



Random forest — workflow

What are Random Forests?

Ensemble Method:
Random forests are an ensemble learning method, meaning they combine multiple individual models (decision trees) to make predictions.

Decision Trees:
Each individual model in a random forest is a decision tree.

Randomness:
The "random" in "random forest" refers to the way the decision trees are constructed, with each tree being trained on a random subset of the data and a random subset of the features.

Classification vs. Regression:

Random forests can be used for both classification (predicting categorical outcomes) and regression (predicting continuous outcomes).

How Random Forests Work:

Training:

The algorithm constructs multiple decision trees during training.

Each tree is trained on a random subset of the training data (using bootstrapping) and a random subset of the features.

Prediction:

Classification: For classification tasks, the final prediction is the class that is the mode (most frequent) of the predictions from all the individual trees.

Regression: For regression tasks, the final prediction is the average of the predictions from all the individual trees.

Advantages of Random Forests:

1. **High Accuracy:** Random forests are known for their high accuracy and predictive power.
2. **Robustness:** They are relatively robust to overfitting and can handle noisy data well.

Feature Importance: Random forests can provide insights into the importance of different features in the data. **Handles Missing Values:** Random forests can handle missing values in the data without requiring preprocessing. Disadvantages of Random Forests:

Computational Cost:

Training random forests can be computationally expensive, especially for large datasets.

Interpretability:

Random forests are often considered "black box" models, meaning it can be difficult to understand why they make certain predictions.

Parameter Tuning:

The performance of random forests can depend on the choice of hyperparameters, which may require tuning.

Applications:

Classification:

Spam detection.

Medical diagnosis.

Customer churn prediction.

Regression:

Predicting house prices. Forecasting stock prices. Estimating customer lifetime value.

Introduction to the Bayes Classifier

A Bayesian classifier, or more specifically a Naive Bayes classifier, is a type of probabilistic classifier based on Bayes' theorem. It's a supervised machine learning algorithm used to classify data by assigning it to the most likely class based on its features. The Naive Bayes classifier makes the simplifying assumption that features are conditionally independent, meaning the presence of one feature doesn't affect the presence of another.

Here's a more detailed breakdown:

Bayes' Theorem:

The foundation of this classifier is Bayes' theorem, which describes the probability of an event based on prior knowledge of related conditions.

Probabilistic Approach:

Bayesian classifiers work by estimating the probability of a data point belonging to each class.

Naïve Assumption:

The term "naive" refers to the assumption that all features are independent of each other, given the class. This simplification makes the calculations easier and faster, while still achieving good performance in many cases.

Classification Process:

The classifier calculates the probability of each class given the input data and then assigns the data point to the class with the highest probability.

Applications:

Naive Bayes classifiers are widely used in various applications, including:

Text classification: Identifying the topic or sentiment of a piece of text.

Spam detection: Classifying emails as spam or not spam.

Medical diagnosis: Predicting the likelihood of a disease based on symptoms.

Weather prediction: Predicting the weather conditions based on various factors.

Advantages:

Simple to implement and understand.

Requires less training data compared to other classifiers. Fast to train and predict.

Works well with both continuous and categorical data.

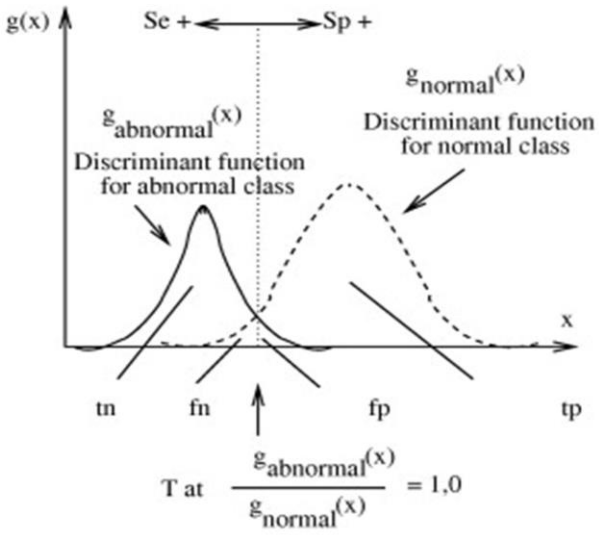
Limitations:

The independence assumption can be unrealistic in some cases. Sensitive to irrelevant features.

THE BAYES CLASSIFIER:

INTRODUCTION TO THE BAYES CLASSIFIER

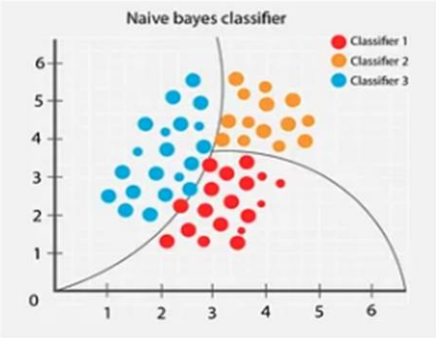
Bayesian classifiers are statistical classifiers, based on Bayes' theorem. They can predict class membership probabilities such as the probability that a given tuple belongs to a particular class. Naive Bayesian classifiers assume that all features in are mutually independent.



$$P(A|B) = \frac{P(B|A)\ P(A)}{P(B)}$$

using Bayesian probability terminology, the above equation can be written as

$$\text{Posterior} = \frac{\text{prior x likelihood}}{\text{evidence}}$$



Bayes' Rule and Bayesian inference are fundamental concepts in probability and statistics, used to update beliefs about an event based on new evidence. Bayes' Rule provides the mathematical framework for this updating, while Bayesian inference is the process of applying that framework to draw conclusions about a model or hypothesis given observed data.

Here's a more detailed explanation:

Bayes' Rule:

Bayes' Rule is a theorem that describes the probability of an event, given that another event has already occurred. It essentially provides a way to revise your prior beliefs about an event in light of new information.

The formula for Bayes' Rule is: P(A|B) = [P(B|A) * P(A)] / P(B).

P(A|B): The posterior probability of event A, given that event B has occurred (your updated belief).

P(B|A): The likelihood of event B, given that event A is true (how likely the evidence is, assuming A is true).

P(A): The prior probability of event A (your initial belief about A).

P(B): The probability of event B (evidence).

In simpler terms, Bayes' Rule tells you how to combine your prior knowledge with the new evidence to arrive at a more informed belief.

Bayesian Inference:

Bayesian inference is a statistical method that uses Bayes' Rule to update beliefs about parameters of a model given observed data.

It involves assigning prior probabilities to possible models or hypotheses, then using the observed data (likelihood) to update these probabilities to posterior probabilities.

The posterior probability distribution represents the updated belief about the model or hypothesis, given the data. Bayesian inference is widely used in various fields, including machine learning, medical diagnosis, and scientific modeling.

Key Differences:

Bayes' Rule is a mathematical formula. It's a specific equation that relates probabilities.

Bayesian inference is a statistical methodology. It's a process of using Bayes' Rule to draw conclusions from data.

In essence:

Bayes' Rule provides the mathematical framework for updating beliefs.

Bayesian inference is the practical application of that framework to make statistical inferences.

The Bayes Classifier and its Optimality

The Bayes Optimal Classifier (BOC) in machine learning is the theoretically best possible classifier for a given classification problem. It makes the most probable prediction for a new input, minimizing the probability of classification error. While practically unachievable due to unknown class-membership probabilities, it serves as a benchmark for evaluating the performance of other learning algorithms.

Here's a more detailed explanation:

Key Concepts:

- **Bayes Theorem:**

The BOC is based on Bayes' Theorem, which relates the probabilities of events.

- **Posterior Probability:**

The BOC makes predictions based on the posterior probability of a class given the input features.

- **Prior Probability:**

The BOC also considers the prior probability of each class.

- **Maximum A Posteriori (MAP):**

The BOC is closely related to the MAP principle, where the class with the highest posterior probability is selected.

- **Bayes Error:**

The BOC achieves the minimum possible error rate, known as the Bayes error.

How it works:

1. **Determine Posterior Probabilities:** For a new input, the BOC calculates the posterior probability of each class (e.g., P(class A | input data)).
2. **Select the Most Probable Class:** The BOC predicts the class with the highest posterior probability.
3. **Theoretical Optimality:** By making predictions based on these probabilities, the BOC minimizes the overall classification error rate.

Why it's important:

- **Theoretical Benchmark:**

The BOC provides a theoretical upper bound on the performance of any classifier.

- **Evaluation Tool:**

The BOC is used to evaluate the performance of other learning algorithms,

comparing their error rates to the Bayes error.

- **Understanding Learning Algorithms:**

  The BOC helps understand the limitations and potential of different learning algorithms.

  Limitations:

- **Unknown Class Probabilities:**

The BOC requires knowledge of the class-membership probabilities, which are often unknown in real-world scenarios.

- **Practical Unachievability:**

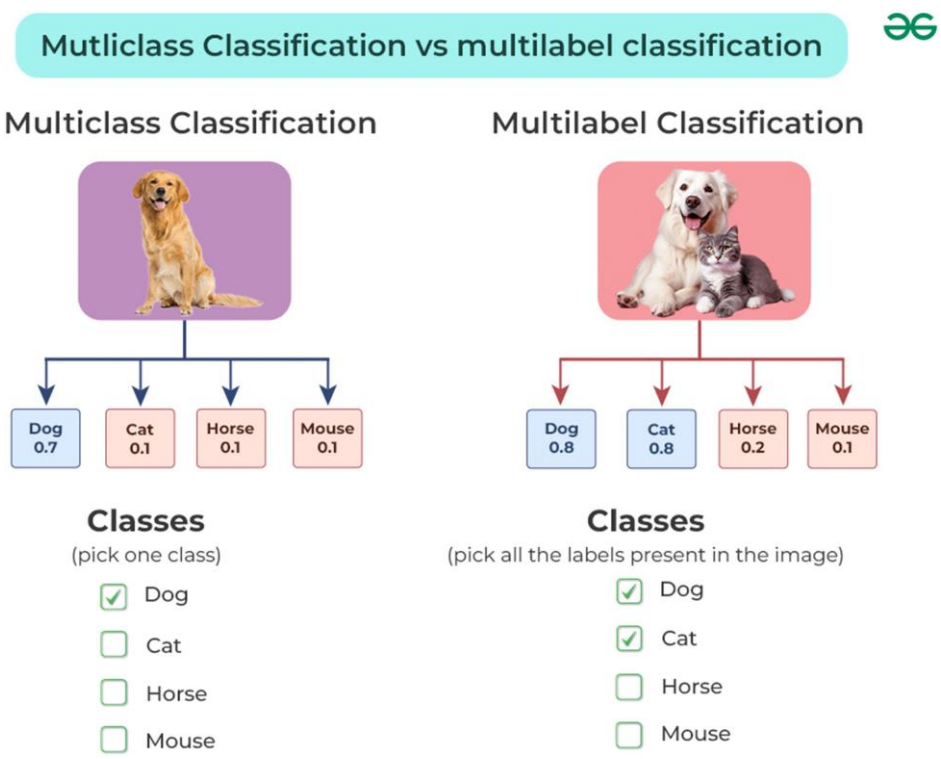
  Due to the need for perfect knowledge of class probabilities, the BOC is practically unachievable.

  In essence, the Bayes Optimal Classifier is a theoretical ideal that helps us understand the limits of what can be achieved in classification and serves as a benchmark for comparing the performance of different machine learning algorithms.

Multi-Class Classification

Multiclass Classification vs Multi-label Classification

Multiclass classification is a machine learning task where the goal is to assign instances to one of multiple predefined classes or categories, where each instance belongs to exactly one class. Whereas multilabel classification is a machine learning task where each instance can be associated with multiple labels simultaneously, allowing for the assignment of multiple binary labels to the instance. In this article we are going to understand the multi-class classification and multi-label classification, how they are different, how they are evaluated, how to choose the best method for your problem, and much more.



What is Multiclass Classification?

Multiclass classification is a machine learning challenge focused on categorizing data into more than two classes. While binary classification involves distinguishing between only two classes, multiclass classification expands this scope to involve distinguishing between multiple classes. In essence, the goal is to train a model that can effectively sort instances into various predefined categories, providing a nuanced solution for scenarios where items can belong to more than two exclusive groups. This approach is commonly employed in tasks such as handwriting recognition, email categorization, and image classification involving more than two distinct categories.

Multiclass classification is a type of machine learning task where the goal is to categorize instances into one of several predefined classes. Unlike binary classification, where there are only two possible outcomes, multiclass classification involves distinguishing between multiple classes or categories. The fundamental idea is to teach a model to assign the most appropriate class label to each instance based on its features.

Multiclass classification finds application in a wide range of real-world scenarios. Consider email categorization, where emails need to be sorted into categories like

"spam," "ham" (non-spam), or "important." Another classic example is handwritten digit recognition, where the task is to identify which digit (0 through 9) is written in a given image. Other applications include speech recognition, sentiment analysis, and image classification into multiple categories.

Model Training Techniques:

Training a multiclass classification model involves employing specific techniques to ensure accurate class assignment. One common approach is to use softmax activation in the output layer of the neural network. Softmax converts the raw model outputs into probabilities, assigning higher probabilities to the correct classes. Additionally, categorical cross-entropy loss is often used as the objective function during training. This loss function measures the dissimilarity between the predicted probabilities and the actual class labels, guiding the model to minimize errors and improve accuracy.

Evaluation Metrics:

To assess the performance of a multiclass classification model, various evaluation metrics like accuracy, precision, recall (sensitivity) and F1 score.

Understanding these concepts is crucial for practitioners working on multiclass classification problems, as they form the foundation for designing effective models and assessing their accuracy in real-world applications.

What is Multi-label Classification?

Multi-label classification is a machine learning paradigm where instances can be associated with multiple labels simultaneously. Unlike traditional classification tasks, where an instance is assigned a single exclusive label,

multi-label classification recognizes the possibility for instances to exhibit characteristics that span across various categories. The goal is to develop models capable of accurately predicting and assigning a set of relevant labels to each instance, reflecting the complex relationships and diversity inherent in real-world datasets.

This approach acknowledges the overlapping nature of labels, providing a more realistic representation of the multifaceted attributes present in the data.

Multi-label classification is a machine learning task where instances can be

associated with multiple labels simultaneously. This differs from multiclass classification, where each instance is assigned to one and only one class. In multi-label scenarios, an instance may exhibit characteristics that correspond to several different categories, making the task more intricate and reflecting the complexity often found in real-world data.

Multi-label classification is highly applicable in diverse scenarios where instances can possess multiple attributes or labels. Examples include:

- Document Tagging: Assigning multiple tags or topics to a document, such as labeling an article as both "technology" and "business."
- Image Classification with Multiple Labels: Identifying and labeling multiple objects or features within an image, like recognizing both "cat" and "outdoor" in a photograph.

Model Training Techniques:

Training models for multi-label classification involves specific techniques to accommodate the simultaneous assignment of multiple labels to instances:

- Sigmoid Activation: In the output layer of the neural network, sigmoid activation is often used. Unlike softmax in multiclass scenarios, sigmoid independently activates each output node, producing a value between 0 and 1, representing the likelihood of the corresponding label being present.
- Binary Cross-Entropy Loss: This loss function is employed during training to measure the dissimilarity between the predicted probabilities and the actual presence or absence of each label. It guides the model to minimize errors in its multi-label predictions.

Evaluation Metrics:

Assessing the performance of a multi-label classification model requires specific metrics tailored to handle the complexity of multiple labels per instance:

- Hamming Loss: This metric calculates the fraction of labels that are incorrectly predicted. It provides a comprehensive measure of overall model performance in terms of label accuracy.
- Precision at k: Precision at k evaluates the precision of the top-k predicted labels, recognizing that not all labels need to be considered. It accounts for scenarios where only the most relevant labels are of interest.

- Recall at k: Similar to precision at k, recall at k assesses the recall of the top-k predicted labels. It focuses on capturing the relevant labels among the top predictions.

Understanding these nuances of multi-label classification is essential for practitioners working on tasks where instances can belong to multiple categories simultaneously, ensuring effective model design and evaluation in complex real-world scenarios.

Differences between Multi class and Multi label Classification

Features	Multi class classification.	Multi label classification
Output Structure:	The output is a single class label assigned to each instance, indicating the most probable or correct class.	The output is a set of binary values indicating the presence or absence of each label for each instance. Instances can be associated with multiple labels simultaneously.
Model Output:	the model assigns a single class label to each instance based on the class with the highest probability or confidence.	The model outputs a binary vector for each instance, where each element corresponds to a label, indicating whether it is present or not.
Training Techniques:	Techniques like softmax activation and categorical cross-entropy loss are commonly used for training models to handle multiple classes.	Techniques like sigmoid activation and binary cross-entropy loss are employed, treating each label independently.

Features	Multi class classification.	Multi label classification
Class Assignment:	Each instance is assigned to one and only one class, making the classification mutually exclusive.	Instances can be associated with multiple labels, allowing for overlapping or shared characteristics.
Evaluation Metrics:	Metrics such as accuracy, precision, recall, and F1 score are commonly used to assess the overall performance of the model.	Metrics like Hamming loss, precision at k, and recall at k are more appropriate, as they account for the presence of multiple labels for each instance.
Model Complexity:	Generally considered simpler as it involves assigning instances to exclusive classes.	Can be more complex due to the need to capture dependencies and correlations between multiple labels.
Problem Complexity:	Typically used for simpler problems where instances belong to mutually exclusive categories.	Suited for more complex scenarios where instances can exhibit characteristics of multiple labels simultaneously.

Choosing Between Multi-Class and Multi-Label Classification:

When embarking on a classification task, one of the foundational decisions is whether to opt for multi-class or multi-label classification, and this choice significantly influences the model's performance and relevance to real- world scenarios.

- Assess whether the instances in your dataset belong to mutually exclusive classes (Multi-Class) or if they can have multiple labels simultaneously (Multi-Label). Understanding the nature of labels is fundamental in choosing the appropriate classification approach.
- Examine the relationships between labels. If the labels are independent or weakly correlated, multi-class classification may be suitable. For strong correlations or overlapping characteristics, multi-label classification is more appropriate.
- Gauge the complexity of your classification problem. Multi-class classification is generally simpler as it deals with exclusive categorization. If the problem is inherently complex and instances can have diverse characteristics, opt for multi-label classification.
- Consider domain-specific requirements and constraints. Some domains naturally

lend themselves to one approach over the other based on the inherent characteristics of the data and the specific objectives of the task.

In conclusion, the choice between multi-class and multi-label classification should be made considering the intricacies of the problem, the nature of the data, and the specific requirements of the application. Each approach has its merits, and selecting the most suitable classification method is pivotal for achieving optimal model performance in diverse real-world scenarios.

Class Conditional Independence and Naive Bayes Classifier (NBC)

Naive Bayes Classifiers

Naive Bayes is a classification algorithm that uses probability to predict which category a data point belongs to, assuming that all features are unrelated. This article will give you an overview as well as more advanced use and implementation of Naive Bayes in machine learning.

Illustration behind the Naive Bayes algorithm. We estimate $P(x\alpha|y)P(x\alpha|y)$ independently in each dimension (middle two images) and then obtain an estimate of the full data distribution by assuming conditional independence

$$P(x|y)=\prod\alpha P(x\alpha|y)P(x|y)=\prod\alpha P(x\alpha|y)$$(very right image).

Key Features of Naive Bayes Classifiers

The main idea behind the Naive Bayes classifier is to use **Bayes' Theorem** to classify data based on the probabilities of different classes given the features of the data. It is used mostly in high-dimensional text classification

- The Naive Bayes Classifier is a simple probabilistic classifier and it has very few number of parameters which are used to build the ML models that can predict at a faster speed than other classification algorithms.
- It is a probabilistic classifier because it assumes that one feature in the model is independent of existence of another feature. In other words, each feature contributes to the predictions with no relation between each other.
- Naïve Bayes Algorithm is used in spam filtration, Sentimental analysis, classifying articles and many more.

Why it is Called Naive Bayes?

It is named as "Naive" because it assumes the presence of one feature does not affect other features. The "Bayes" part of the name refers to its basis in Bayes' Theorem. Consider a fictional dataset that describes the weather conditions for playing a game of golf. Given the weather conditions, each tuple classifies the conditions as fit("Yes") or unfit("No") for playing golf. Here is a tabular representation of our dataset.

	Outlook	Temperature	Humidity	Windy	Play Golf
0	Rainy	Hot	High	False	No
1	Rainy	Hot	High	True	No
2	Overcast	Hot	High	False	Yes
3	Sunny	Mild	High	False	Yes
4	Sunny	Cool	Normal	False	Yes
5	Sunny	Cool	Normal	True	No
6	Overcast	Cool	Normal	True	Yes
7	Rainy	Mild	High	False	No
8	Rainy	Cool	Normal	False	Yes
9	Sunny	Mild	Normal	False	Yes

	Outlook	Temperature	Humidity	Windy	Play Golf
10	Rainy	Mild	Normal	True	Yes
11	Overcast	Mild	High	True	Yes
12	Overcast	Hot	Normal	False	Yes
13	Sunny	Mild	High	True	No

The dataset is divided into two parts, namely, **feature matrix** and the **response vector**.

- Feature matrix contains all the vectors(rows) of dataset in which each vector consists of the value

  of **dependent features**. In above dataset, features are 'Outlook', 'Temperature', 'Humidity' and 'Windy'.
- Response vector contains the value of **class variable**(prediction or output) for each row of feature matrix. In above dataset, the class variable name is 'Play golf'.

Assumption of Naive Bayes

The fundamental Naive Bayes assumption is that each feature makes an:

- **Feature independence:** This means that when we are trying to classify something, we assume that each feature (or piece of information) in the data does not affect any other feature.
- **Continuous features are normally distributed:** If a feature is continuous, then it is assumed to be normally distributed within each class.
- **Discrete features have multinomial distributions:** If a feature is discrete, then it is assumed to have a multinomial distribution within each class.
- **Features are equally important:** All features are assumed to contribute equally to the prediction of the class label.
- **No missing data:** The data should not contain any missing values.

Introduction to Bayes' Theorem

Bayes' Theorem provides a principled way to reverse conditional probabilities. It is defined as:

$P(y|X)=P(X|y)\cdot P(y)P(X)P(y|X)=P(X)P(X|y)\cdot P(y)$

Where:

- P(y|X)$P(y|X)$: Posterior probability, probability of class yy given features XX
- P(X|y)$P(X|y)$: Likelihood, probability of features XX given class yy
- P(y)$P(y)$: Prior probability of class yy
- P(X)$P(X)$: Marginal likelihood or evidence

Naive Bayes Working

1. Terminology

Consider a classification problem (like predicting if someone plays golf based on weather). Then:

- yy is the class label (e.g. "Yes" or "No" for playing golf)
- X=(x1,x2,...,xn)$X=(x1,x2,...,xn)$ is the feature vector (e.g. Outlook, Temperature, Humidity, Wind) A sample row from the dataset:

$X=(Rainy, Hot, High, False),y=NoX=(Rainy, Hot, High, False),y=No$

This represents:

What is the probability that someone will not play golf given that the weather is Rainy, Hot, High humidity, and No wind?

2. The Naive Assumption

The "naive" in Naive Bayes comes from the assumption that all features are independent given the class. That is:

$P(x1,x2,...,xn|y)=P(x1|y)\cdot P(x2|y)\cdots P(xn|y)P(x1,x2,...,xn|y)=P(x1|y)\cdot P(x2|y)\cdots P(xn|y)$

Thus, Bayes' theorem becomes:

$P(y|x1,...,xn)=P(y)\cdot \prod i=1nP(xi|y)P(x1)P(x2)...P(xn)P(y|x1,...,xn)=P(x1)P(x2)...P(xn)P(y)\cdot \prod i=1nP(xi|y)$

Since the denominator is constant for a given input, we can write:

$P(y|x1,...,xn)\propto P(y)\cdot \prod i=1nP(xi|y)P(y|x1,...,xn)\propto P(y)\cdot \prod i=1nP(xi|y)$

3. Constructing the Naive Bayes Classifier

We compute the posterior for each class yy and choose the class with the highest

probability:

$y^{=arg \quad max \quad yP(y)\cdot \prod i=1nP(xi|y)y^{=argmaxyP(y)\cdot \prod i=1nP(xi|y)}}$

This becomes our Naive Bayes classifier.

4. **Example: Weather Dataset**

Let's take a dataset used for predicting if golf is played based on:

- **Outlook**: Sunny, Rainy, Overcast
- **Temperature**: Hot, Mild, Cool
- **Humidity**: High, Normal
- **Wind**: True, False

Outlook

	Yes	No	P(yes)	P(no)
Sunny	3	2	3/10	2/4
Overcast	4	0	4/10	0/4
Rainy	3	2	3/10	2/4
Total	10	4	100%	100%

Temperature

	Yes	No	P(yes)	P(no)
Hot	2	2	2/9	2/5
Mild	4	2	4/9	2/5
Cool	3	1	3/9	1/5
Total	9	5	100%	100%

Humidity

	Yes	No	P(yes)	P(no)
High	3	4	3/9	4/5
Normal	6	1	6/9	1/5
Total	9	5	100%	100%

Wind

	Yes	No	P(yes)	P(no)
False	6	2	6/9	2/5
True	3	3	3/9	3/5
Total	9	5	100%	100%

Play		**P(Yes)/P(No)**
Yes	9	9/14
No	5	5/14
Total	14	100%

Example Tables for Naive Bayes

Feature	Value	P (Value | Yes)	P (Value | No)
Outlook	Sunny	2/9	3/5
Temperature	Hot	2/9	2/5
Humidity	Normal	6/9	1/5
Wind	False	6/9	2/5

Example Input: X=(Sunny,Hot,Normal,False)X=($Sunny$,Hot,$Normal$,$False$)
Goal: Predict if golf will be played (Yes or No).

5. **Pre-computation from Dataset Class Probabilities:**

From dataset of 14 rows:

- P(Yes)=914P(Yes)=149
- P(No)=514P(No)=145

Conditional Probabilities (Tables 1–4):

6. **Calculate Posterior Probabilities For Class = Yes:**

P(Yes | today)∝29·29·69·69·914P(Yes | today)∝92·92·96·96·149
P(Yes | today)≈0.02116P(Yes | today)≈0.02116

For Class = No:

P(No | today)∝35·25·15·25·514P(No | today)∝53·52·51·52·145 P(No | today)≈0.0068P(No | today)≈0.0068

7. **Normalize Probabilities**

To compare:

P(Yes | today)=0.021160.02116+0.0068≈0.756P(Yes | today)=0.02116+0.00680.02116≈0.756 P(No | today)=0.00680.02116+0.0068≈0.244P(No | today)=0.02116+0.00680.0068≈0.244

8. **Final Prediction**

Since:

P(Yes | today)>P(No | today)P(Yes | today)>P(No | today) The model predicts: ***Yes (Play Golf)***

Naive Bayes for Continuous Features

For continuous features, we assume a Gaussian distribution:

$P(x_i|y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i-\mu_y)^2}{2\sigma_y^2}\right) P(x_i|y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i-\mu_y)^2}{2\sigma_y^2}\right)$

Where:

- μ_y is the mean of feature x_i for class y
- σ_y^2 is the variance of feature x_i for class y This leads to what is called

Gaussian Naive Bayes. **Types of Naive Bayes Model**

There are three types of Naive Bayes Model :

1. **Gaussian Naive Bayes**

In **Gaussian Naive Bayes**, continuous values associated with each feature are assumed to be distributed according to a Gaussian distribution. A Gaussian distribution is also called Normal distribution When plotted, it gives a bell shaped curve which is symmetric about the mean of the feature values as shown below:

2. **Multinomial Naive Bayes**

Multinomial Naive Bayesis used when features represent the frequency of terms (such as word counts) in a document. It is commonly applied in text classification, where term frequencies are important.

3. **Bernoulli Naive Bayes**

Bernoulli Naive Bayes deals with binary features, where each feature indicates whether a word appears or not in a document. It is suited for scenarios where the presence or absence of terms is more relevant than their frequency. Both models are widely used in document classification tasks

Advantages of Naive Bayes Classifier

- Easy to implement and computationally efficient.
- Effective in cases with a large number of features.
- Performs well even with limited training data.
- It performs well in the presence of categorical features.
- For numerical features data is assumed to come from normal distributions

Disadvantages of Naive Bayes Classifier

- Assumes that features are independent, which may not always hold in real-world data.
- Can be influenced by irrelevant attributes.

- May assign zero probability to unseen events, leading to poor generalization.

Applications of Naive 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 in predicting 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.