**Module 2**

**Data Warehousing and Mining**

**Data Mining** -

Data mining is the process of discovering patterns, correlations, and trends by sifting through large sets of data stored in databases, data warehouses, or other repositories. In the context of data warehousing, data mining plays a critical role by extracting valuable insights from vast amounts of data collected and stored over time. These insights can be used for decision-making, predictive analysis, and gaining a competitive advantage in various industries.

**Association rule mining**

Association rule mining is a data mining technique that aims to discover interesting relationships, patterns, and correlations within large datasets. It focuses on identifying strong associations between different items or variables in the data. It presents these associations in the form of if-then rules, commonly known as association rules.

An association rule consists of an antecedent (if part) and a consequent (then part). The dataset contains an antecedent, and we derive a consequent by using the antecedent.

**Rule Evaluation Metrics –**

* **Support(s) –** The number of transactions that include items in the {X} and {Y} parts of the rule as a percentage of the total number of transaction. It is a measure of how frequently the collection of items occur together as a percentage of all transactions.
* **Support =(X+Y)total –** It is interpreted as fraction of transactions that contain both X and Y.
* **Confidence(c) –** It is the ratio of the no of transactions that includes all items in {B} as well as the no of transactions that includes all items in {A} to the no of transactions that includes all items in {A}.
* **Conf(X=>Y) = Supp(XY)Supp(X) –** It measures how often each item in Y appears in transactions that contains items in X also.
* **Lift(l) –** The lift of the rule X=>Y is the confidence of the rule divided by the expected confidence, assuming that the itemsets X and Y are independent of each other. The expected confidence is the confidence divided by the frequency of {Y}.
* **Lift(X=>Y) = Conf(X=>Y)Supp(Y) –** Lift value near 1 indicates X and Y almost often appear together as expected, greater than 1 means they appear together more than expected and less than 1 means they appear less than expected. Greater lift values indicate stronger association.

**Key Concepts in Association Rule Mining**

1. **Itemset:**
   * A collection of one or more items. For example, in a market basket context, {bread, milk} is an itemset.
2. **Association Rule:**
   * An implication expression of the form X -> Y, where X and Y are itemsets. The rule means that if X occurs, then Y is likely to occur as well. For example, {bread} -> {milk} suggests that customers who buy bread are also likely to buy milk.
3. **Support:**
   * The support of an itemset is the proportion of transactions in the dataset in which the itemset appears. It is a measure of how frequently the itemset occurs in the dataset.
   * Formula: Support(X)=Number of transactions containing XTotal number of transactions\text{Support}(X) = \frac{\text{Number of transactions containing } X}{\text{Total number of transactions}}Support(X)=Total number of transactionsNumber of transactions containing X​
4. **Confidence:**
   * Confidence is a measure of how often items in Y appear in transactions that contain X. It indicates the reliability of the rule.
   * Formula: Confidence(X→Y)=Support(X∪Y)Support(X)\text{Confidence}(X \rightarrow Y) = \frac{\text{Support}(X \cup Y)}{\text{Support}(X)}Confidence(X→Y)=Support(X)Support(X∪Y)​
5. **Lift:**
   * Lift is a measure of how much more likely Y is to be bought when X is bought, compared to the baseline likelihood of Y.
   * Formula: Lift(X→Y)=Support(X∪Y)Support(X)×Support(Y)\text{Lift}(X \rightarrow Y) = \frac{\text{Support}(X \cup Y)}{\text{Support}(X) \times \text{Support}(Y)}Lift(X→Y)=Support(X)×Support(Y)Support(X∪Y)​
   * A lift value greater than 1 indicates a positive correlation between X and Y.

**Steps in Association Rule Mining**

1. **Frequent Itemset Generation:**
   * The first step in association rule mining is to find all itemsets that have a support greater than or equal to a user-specified minimum support threshold. These itemsets are known as frequent itemsets.
2. **Rule Generation:**
   * From the frequent itemsets, generate association rules that satisfy the minimum confidence threshold. This involves calculating the confidence of all possible rules derived from the frequent itemsets and selecting those that meet the minimum confidence criterion.

**Example of Association Rule Mining**

Consider a retail store transaction dataset with the following transactions:

| **Transaction ID** | **Items Purchased** |
| --- | --- |
| 1 | Milk, Bread, Butter |
| 2 | Milk, Bread |
| 3 | Bread, Butter |
| 4 | Milk, Butter |
| 5 | Milk, Bread, Butter, Cheese |

* **Frequent Itemsets:** Let's say the minimum support threshold is 60%. The frequent itemsets could be:
  + {Milk, Bread}: Support = 60% (3 out of 5 transactions)
  + {Milk, Butter}: Support = 60%
  + {Bread, Butter}: Support = 60%
* **Association Rule Generation:**
  + From the itemset {Milk, Bread}, a possible rule is Milk -> Bread with a confidence of 100% (since every transaction with milk also has bread).

**Applications of Association Rule Mining**

1. **Market Basket Analysis:**
   * Helps retailers understand the purchasing behavior of customers by identifying products that are frequently bought together. This can lead to better product placements, cross-selling strategies, and promotions.
2. **Recommender Systems:**
   * Association rules can be used in recommendation systems to suggest products to users based on the items they have previously purchased or interacted with.
3. **Fraud Detection:**
   * By identifying unusual patterns or associations in financial transactions, association rule mining can help detect fraudulent activities.
4. **Healthcare:**
   * Used to discover associations between symptoms and diseases, or between drugs and side effects, to improve diagnosis and treatment plans.
5. **Web Usage Mining:**
   * Helps analyze user behavior by finding patterns in web access logs, such as frequently visited pages or sequences of pages.

**Challenges in Association Rule Mining**

1. **Handling Large Datasets:**
   * The process of finding frequent itemsets and generating rules can be computationally expensive, especially with large datasets containing millions of transactions.
2. **Setting the Right Thresholds:**
   * Choosing appropriate support and confidence thresholds is critical, as too high values may result in few rules, while too low values can lead to an overwhelming number of rules, including many insignificant ones.
3. **Rule Redundancy:**
   * Many association rules may be redundant, providing no new information. Filtering and pruning unnecessary rules are important to focus on the most valuable ones.
4. **Scalability:**
   * As datasets grow, the algorithms must scale efficiently to maintain performance.

**Naïve algorithm**

**Naive Algorithm in Data Mining**

In data mining, a **naive algorithm** typically refers to a simple and straightforward approach to problem-solving. While "naive" often implies a lack of sophistication or optimization, naive algorithms are fundamental, easy to understand, and sometimes surprisingly effective for certain tasks. In data mining, the term "naive" is most commonly associated with the **Naive Bayes classifier**. However, it can also refer to other basic algorithms used in data mining tasks like clustering, classification, or pattern recognition.

**Naive Bayes Algorithm**

The **Naive Bayes algorithm** is a classification technique based on Bayes' Theorem, with a strong (naive) independence assumption between features. Despite its simplicity, it is widely used due to its efficiency and performance in various domains like spam filtering, sentiment analysis, and document classification.

**Bayes' Theorem**

Bayes' Theorem provides a way to calculate the probability of a hypothesis given prior knowledge. The theorem is stated as:

P(H∣E)=P(E∣H)×P(H)P(E)P(H | E) = \frac{P(E | H) \times P(H)}{P(E)}P(H∣E)=P(E)P(E∣H)×P(H)​

Where:

* P(H∣E)P(H | E)P(H∣E) is the posterior probability of hypothesis HHH given evidence EEE.
* P(E∣H)P(E | H)P(E∣H) is the likelihood of observing EEE given that HHH is true.
* P(H)P(H)P(H) is the prior probability of the hypothesis HHH.
* P(E)P(E)P(E) is the probability of the evidence EEE.

**Naive Assumption**

The "naive" aspect of the Naive Bayes classifier comes from the assumption that the features (or attributes) in the dataset are independent of each other given the class label. This means that the presence or absence of one feature does not affect the presence or absence of any other feature, which is often not the case in real-world data. Despite this strong assumption, the Naive Bayes classifier performs surprisingly well in many applications.

**Naive Bayes Classification Process**

1. **Training Phase:**
   * The algorithm is trained on a dataset where each instance is associated with a class label.
   * It calculates the prior probability of each class, which is simply the proportion of instances belonging to that class.
   * For each feature, the algorithm calculates the likelihood of that feature given each class.
2. **Prediction Phase:**
   * For a new instance, the algorithm calculates the posterior probability of each class by combining the prior probabilities with the likelihoods of the features using Bayes' Theorem.
   * The class with the highest posterior probability is assigned to the new instance.

**Example**

Consider a simple example of classifying whether an email is spam or not spam based on the presence of certain words.

* **Training Data:**
  + Email 1: "Buy cheap watches" (Spam)
  + Email 2: "Meeting tomorrow" (Not Spam)
  + Email 3: "Cheap watches available" (Spam)
  + Email 4: "Project meeting" (Not Spam)
* **Features:** The words in the emails (e.g., "cheap," "watches," "meeting").
* **Process:**
  + Calculate the prior probability of Spam and Not Spam.
  + Calculate the likelihood of each word given Spam and Not Spam.
  + For a new email, "Cheap watches meeting," calculate the posterior probabilities for Spam and Not Spam using the Naive Bayes formula, and classify it accordingly.

**Advantages of Naive Bayes Algorithm**

1. **Simplicity:** The algorithm is easy to understand and implement.
2. **Efficiency:** Naive Bayes is computationally efficient and works well with large datasets.
3. **Fast Training and Prediction:** The algorithm requires a small amount of training data to estimate the parameters and can make predictions quickly.
4. **Scalability:** It scales well with the number of features and instances in the dataset.
5. **Works Well with Categorical Data:** Naive Bayes performs particularly well with categorical data.

**Disadvantages of Naive Bayes Algorithm**

1. **Strong Independence Assumption:** The naive assumption that features are independent of each other is often unrealistic, leading to less accurate predictions when feature dependencies exist.
2. **Zero Probability Issue:** If a certain feature value does not occur in the training set for a particular class, the probability estimate becomes zero, leading to incorrect predictions. This can be mitigated by techniques like Laplace smoothing.
3. **Limited to Linearly Separable Data:** Naive Bayes may struggle with data that is not linearly separable or when the decision boundary is complex.

**Applications of Naive Bayes**

1. **Spam Filtering:** Naive Bayes is commonly used in email spam detection, where it classifies emails based on the presence of spam-related words.
2. **Sentiment Analysis:** It is used to classify text into positive, negative, or neutral sentiment in social media, reviews, and other textual data.
3. **Document Classification:** Naive Bayes is effective in categorizing documents into predefined categories, such as news articles into different topics.
4. **Medical Diagnosis:** The algorithm can be used to predict diseases based on patient symptoms and medical history.

**Other Naive Algorithms in Data Mining**

1. **Naive Clustering (K-Means):**
   * In the context of clustering, a naive algorithm might refer to the K-Means algorithm, which assigns data points to clusters based on their proximity to cluster centroids. Although simple, K-Means is a widely used and effective clustering technique.
2. **Naive Sequential Pattern Mining:**
   * This involves identifying sequences of events or actions that frequently occur together in a dataset, without considering temporal or contextual relationships between the events.
3. **Naive Association Rule Mining:**
   * A basic approach to finding association rules in a dataset, without optimization for rule pruning or handling large datasets.

**Apriori algorithm**

The Apriori algorithm in data mining is a widely used algorithm to find frequent itemsets in a dataset. The Apriori algorithm is used to implement frequent pattern mining (FPM). Frequent pattern mining is a data mining technique to discover frequent patterns or relationships between items in a dataset. Frequent pattern mining involves finding sets of items or itemsets that occur together frequently in a dataset. These sets of items or itemsets are called frequent patterns, and their frequency is measured by the number of transactions in which they occur.

In data mining, an itemset is a collection of one or more items that appear together in a transaction or dataset. An itemset can be either a single item, also known as a 1-itemset, or a set of k items, also known as a k-itemset. For example, in sales transactions of a retail store, an itemset can be referred to as products purchased together, such as bread and milk, which would be a 2-item set. The Apriori algorithm can be used to discover frequent itemsets in the sales transactions of a retail store. For instance, the algorithm might discover that customers who purchase bread and milk together often also purchase eggs. This information can be used to recommend eggs to customers who purchase bread and milk in the future.

The Apriori algorithm is called "apriori" because it uses prior knowledge about the frequent itemsets. The algorithm uses the concept of "apriori property," which states that if an itemset is frequent, then all of its subsets must also be frequent.

**Apriori Property**

The Apriori property is a fundamental property of frequent itemsets used in the Apriori algorithm. In other words, if an itemset appears frequently enough in the dataset to be considered significant, then all of its subsets must also appear frequently enough to be significant. For example, if the itemset {A, B, C} frequently appears in a dataset, then the subsets {A, B}, {A, C}, {B, C}, {A}, {B}, and {C} must also appear frequently in the dataset.

The Apriori property allows the Apriori algorithm in data mining to efficiently search for frequent itemsets by eliminating candidate itemsets containing infrequent subsets, as they cannot be frequent. This search space pruning reduces the time and memory required to find frequent itemsets in large datasets.

**Apriori Algorithm Components**

Before getting into the steps involved in the Apriori algorithm, let’s understand the various terminologies used in the Apriori algorithm-

**Support**

In the Apriori algorithm, support refers to the frequency or occurrence of an item set in a dataset. It is defined as the proportion of transactions in the dataset that contain the itemset. For example, let's consider a dataset of sales transactions in a retail store that contains the following items - milk, bread, cheese, eggs, butter, and yogurt. To calculate the support of an itemset, we count the number of transactions in which the itemset appears and divide it by the total number of transactions in the dataset. For instance, if the itemset {milk, bread} appears in 5 transactions out of 10 transactions in the dataset, then its support is 5/10=0.55/10=0.5, or 50%.

In the Apriori algorithm, itemsets with a support value above the minimum defined support threshold are considered frequent and are used to generate candidate itemsets for the next iteration of the algorithm.

Support(A)=Number of Transactions in which A occursNumber of all Transactions*Support*(*A*)=*Number* *of* *all* *TransactionsNumber* *of* *Transactions* *in* *which* *A* *occurs*​

**Lift**

Lift measures the strength of the association between two items. It is defined as the ratio of the support of the two items occurring together to the support of the individual items multiplied together. Lift for any two items can be calculated using the below formula -

Lift(A→B)=Support(A and B)Support(A)∗Support(B)*Lift*(*A*→*B*)=*Support*(*A*)∗*Support*(*B*)*Support*(*A* *and* *B*)​

If the lift value is greater than 1, then it indicates a positive association between the two items, which means that the two items are more likely to be bought together. A lift value of exactly 1 indicates that the two items are independent and there is no association between the two items, while a value less than 1 indicates a negative association, meaning that two items are more likely to be bought separately.

**Confidence**

In the Apriori algorithm, confidence is also a measure of the strength of the association between two items in an itemset. It is defined as the conditional probability that item B appears in a transaction, given that another item A appears in the same transaction. Support for two items can be calculated using the below formula.

confidence(A⇒B)=P(B/A)=sup(A∪B)sup(A)*confidence*(*A*⇒*B*)=*P*(*B*/*A*)=*sup*(*A*)*sup*(*A*∪*B*)​

If the confidence value exceeds a specified threshold, it indicates that item B is likely to be purchased with item A. For instance, if the confidence of the association between "bread" and "butter" is 0.8, it means that when a customer buys "bread", there is an 80% chance that they will also buy "butter". This can be useful in recommending to customers or optimizing product placement in a store.

**Steps in Apriori Algorithm**

Here are the steps involved in implementing the Apriori algorithm in data mining -

1. **Define minimum support threshold** - This is the minimum number of times an item set must appear in the dataset to be considered as frequent. The support threshold is usually set by the user based on the size of the dataset and the domain knowledge.
2. **Generate a list of frequent 1-item sets** - Scan the entire dataset to identify the items that meet the minimum support threshold. These item sets are known as frequent 1-item sets.
3. **Generate candidate item sets** - In this step, the algorithm generates a list of candidate item sets of length k+1 from the frequent k-item sets identified in the previous step.
4. **Count the support of each candidate item set** - Scan the dataset again to count the number of times each candidate item set appears in the dataset.
5. **Prune the candidate item sets** - Remove the item sets that do not meet the minimum support threshold.
6. Repeat steps 3-5 until no more frequent item sets can be generated.
7. **Generate association rules** - Once the frequent item sets have been identified, the algorithm generates association rules from them. Association rules are rules of form A -> B, where A and B are item sets. The rule indicates that if a transaction contains A, it is also likely to contain B.
8. **Evaluate the association rules** - Finally, the association rules are evaluated based on metrics such as confidence and lift.

**Apriori Algorithm Example**

Let’s try to understand the Apriori algorithm implementation using an example. In this example, we will use a minimum support threshold of 3. This means an item set must appear in at least three transactions to be considered frequent.

* Let’s consider the transaction dataset of a retail store as shown in the below table.

| **TID** | **Items** |
| --- | --- |
| T1 | {milk, bread} |
| T2 | {bread, sugar} |
| T3 | {bread, butter} |
| T4 | {milk, bread, sugar} |
| T5 | {milk, bread, butter} |
| T6 | {milk, bread, butter} |
| T7 | {milk, sugar} |
| T8 | {milk, sugar} |
| T9 | {sugar, butter} |
| T10 | {milk, sugar, butter} |
| T11 | {milk, bread, butter} |

* Let’s calculate support for each item present in the dataset. As shown in the below table, support for all items is greater than 3. It means that all items are considered as frequent 1-itemsets and will be used to generate candidates for 2-itemsets.

| **Item** | **Support (Frequency)** |
| --- | --- |
| milk | 8 |
| bread | 7 |
| sugar | 5 |
| butter | 7 |

* Below table represents all candidates generated from frequent 1-itemsets identified from the previous step and their support value.

| **Candidate Item Sets** | **Support (Frequency)** |
| --- | --- |
| {milk, bread} | 5 |
| {milk, sugar} | 3 |
| {milk, butter} | 5 |
| {bread, sugar} | 2 |
| {bread, butter} | 3 |
| {sugar, butter} | 2 |

* Now remove candidate item sets that do not meet the minimum support threshold of 3. After this step, frequent 2-itemsets would be - {milk, bread}, {milk, sugar}, {milk, butter}, and {bread, butter}. In the next step, let’s generate candidates for 3-itemsets and calculate their respective support values. It is shown in the below table.

| **Candidate Item Sets** | **Support (Frequency)** |
| --- | --- |
| {milk, bread, sugar} | 1 |
| {milk, bread, butter} | 3 |
| {milk, sugar, butter} | 1 |

* As we can see in the above table, only one candidate itemset exceeds the minimum defined support threshold - {milk, bread, butter}. As there is only one 3-itemset exceeding minimum support, we can’t generate candidates for 4-itemsets. So, in the next step, we can write the association rules and their respective metrics, as shown in the below table.

| **Candidate Item Sets** | **Support (Frequency)** |
| --- | --- |
| {milk, bread} | {butter} (Confidence - 60%)1 |
| {bread, butter} | {milk} (Confidence - 100%) |
| {milk, butter} | {bread} (Confidence - 60%) |

* Based on association rules mentioned in the above table, we can recommend products to the customer or optimize product placement in retail stores.

**Advantages and Limitations of Apriori Algorithm**

Here are some of the advantages of the Apriori algorithm in data mining -

* Apriori algorithm is simple and easy to implement, making it accessible even to those without a deep understanding of data mining or machine learning.
* Apriori algorithm can handle large datasets and run on distributed systems, making it scalable for large-scale applications.
* Apriori algorithm is one of the most widely used algorithms for association rule mining and is supported by many popular data mining tools.

Below are some of the limitations of the Apriori algorithm in data mining -

* Apriori algorithm can be computationally expensive, especially for large datasets with many itemsets. For example, if a dataset contains 104104 from frequent 1- itemsets, it will generate more than 107107 2-length candidates, which makes this algorithm computationally expensive.
* Apriori algorithm can generate a large number of rules, making it difficult to sift through and identify the most important ones.
* The algorithm requires multiple database scans to generate frequent itemsets, which can be a limitation in systems where data access is slow or expensive.
* Apriori algorithm is sensitive to data sparsity, meaning it may not perform well on datasets with a low frequency of itemsets.

**Direct hashing and pruning**

Direct Hashing and Pruning (DHP) is an advanced technique in data mining, specifically designed to enhance the efficiency of association rule mining, which is often associated with the Apriori algorithm. The DHP method helps in reducing the number of candidate itemsets generated during the mining process, thereby optimizing both time and space complexity.

**Background and Motivation**

Association rule mining is a fundamental task in data mining, used to discover interesting relationships or associations between items in large datasets. The Apriori algorithm is one of the most well-known methods for association rule mining, but it suffers from performance inefficiencies due to the large number of candidate itemsets it generates. The process of generating and testing these candidate itemsets can be computationally expensive, especially in large datasets with many items.

The Direct Hashing and Pruning (DHP) technique was introduced to address these challenges by reducing the number of candidate k-itemsets (sets of k items) during the frequent itemset generation process.

**Key Concepts of DHP**

1. **Hashing Technique**:
   * DHP employs a hashing technique to manage candidate k-itemsets. During the generation of k-itemsets, a hash table is used to record the occurrence of itemsets in the transaction database.
   * Each candidate k-itemset is mapped to a specific bucket in the hash table using a hash function. The idea is that if a hash bucket contains few or no itemsets, then the k-itemsets that hash to this bucket are unlikely to be frequent, and can be pruned early from further consideration.
2. **Pruning of Candidates**:
   * After the hash table is populated, DHP prunes the candidate k-itemsets that correspond to hash buckets with counts below a certain threshold.
   * This pruning significantly reduces the number of candidate k-itemsets that need to be checked for support in subsequent iterations.
3. **Support Counting**:
   * For the remaining candidate k-itemsets after pruning, DHP counts their support by scanning the transaction database. Only those itemsets that meet the minimum support threshold are kept as frequent itemsets.

**Steps in the DHP Process**

1. **Initialization**:
   * Start with the set of all items in the dataset (1-itemsets).
   * Use a hash table to record pairs of items (2-itemsets) by hashing them into appropriate buckets.
2. **Hash Table Construction**:
   * During the first pass over the transaction database, populate the hash table by mapping pairs of items (2-itemsets) into hash buckets. Count the occurrences of these item pairs.
3. **Candidate Generation and Pruning**:
   * For the next level (k-itemsets), generate candidate itemsets by extending the frequent (k-1)-itemsets.
   * Before counting their support, prune those k-itemsets whose corresponding hash bucket counts are below a specified threshold.
4. **Support Counting and Frequent Itemset Identification**:
   * For the non-pruned candidates, count their support by scanning the transaction database.
   * Keep only those itemsets that meet the minimum support threshold as frequent itemsets.
5. **Iterate**:
   * Repeat the process for k+1 itemsets until no further frequent itemsets can be generated.

**Advantages of DHP**

1. **Efficiency**:
   * DHP significantly reduces the number of candidate k-itemsets, thereby decreasing the computational cost associated with support counting.
2. **Memory Usage**:
   * By pruning unnecessary candidates early, DHP also reduces memory consumption, making it more scalable for large datasets.
3. **Improved Performance**:
   * DHP enhances the performance of the Apriori algorithm by focusing only on the most promising candidate itemsets, leading to faster identification of frequent itemsets.

**Disadvantages of DHP**

1. **Hash Collisions**:
   * The effectiveness of DHP depends on the quality of the hash function. Poorly designed hash functions can lead to collisions, where multiple k-itemsets are mapped to the same bucket, reducing the pruning effectiveness.
2. **Overhead**:
   * The process of constructing and maintaining the hash table adds some overhead to the algorithm, especially for large datasets with many item pairs.
3. **Dependent on Transaction Length**:
   * The effectiveness of DHP can vary depending on the average length of transactions in the database. Very short or very long transactions can affect the hashing and pruning efficiency.

**Dynamic Item Set Counting (DIC)**

Dynamic Item Set Counting (DIC) is an approach for mining frequent itemsets in large databases, specifically in the context of association rule mining. The idea behind DIC is to count itemsets dynamically during the scanning process of the database, reducing the number of passes required.

**Key Features:**

* **Partitioning:** The database is partitioned into equal-sized blocks.
* **Dynamic Counting:** During the scan of the database, the itemsets are counted at different intervals (dynamically). When an itemset meets a certain threshold of support in the current block, it starts being tracked in subsequent blocks.
* **Less Memory Usage:** Only itemsets that meet the support threshold are counted, reducing the memory needed compared to other methods like Apriori.

**Advantages:**

* Reduces the number of scans required over the database.
* Dynamically adjusts to the data being scanned, improving efficiency in certain scenarios.

**Disadvantages:**

* The performance may still degrade with very large datasets or when the distribution of itemsets varies significantly across blocks.

**2. Mining Frequent Patterns without Candidate Generation**

The FP-Growth Algorithm is an alternative way to find frequent item sets without using candidate generations, thus improving performance. For so much, it uses a divide-and-conquer strategy. The core of this method is the usage of a special data structure named frequent-pattern tree (FP-tree), which retains the item set association information.

**This algorithm works as follows:**

* First, it compresses the input database creating an FP-tree instance to represent frequent items.
* After this first step, it divides the compressed database into a set of conditional databases, each associated with one frequent pattern.
* Finally, each such database is mined separately.

Using this strategy, the FP-Growth reduces the search costs by recursively looking for short patterns and then concatenating them into the long frequent patterns.

FP-Tree

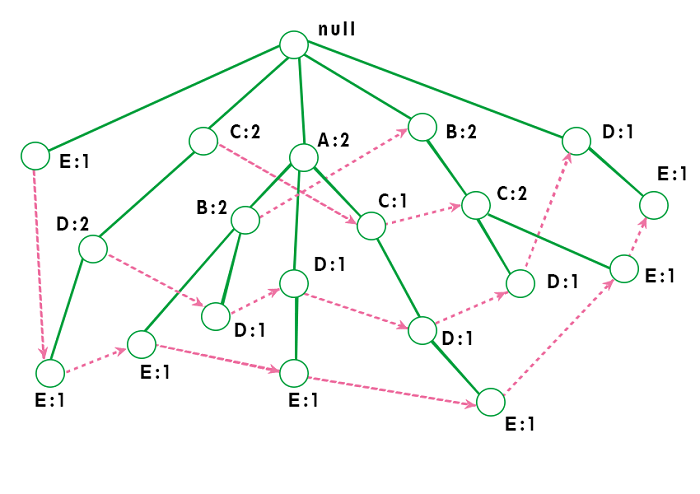
The frequent-pattern tree (FP-tree) is a compact data structure that stores quantitative information about frequent patterns in a database. Each transaction is read and then mapped onto a path in the FP-tree. This is done until all transactions have been read. Different transactions with common subsets allow the tree to remain compact because their paths overlap.

A frequent Pattern Tree is made with the initial item sets of the database. The purpose of the FP tree is to mine the most frequent pattern. Each node of the FP tree represents an item of the item set.

The root node represents null, while the lower nodes represent the item sets. The associations of the nodes with the lower nodes, that is, the item sets with the other item sets, are maintained while forming the tree.

Han defines the FP-tree as the tree structure given below:

1. One root is labelled as "null" with a set of item-prefix subtrees as children and a frequent-item-header table.
2. Each node in the item-prefix subtree consists of three fields:
   * Item-name: registers which item is represented by the node;
   * Count: the number of transactions represented by the portion of the path reaching the node;
   * Node-link: links to the next node in the FP-tree carrying the same item name or null if there is none.
3. Each entry in the frequent-item-header table consists of two fields:
   * Item-name: as the same to the node;
   * Head of node-link: a pointer to the first node in the FP-tree carrying the item name.



The worst-case scenario occurs when every transaction has a unique item set. So the space needed to store the tree is greater than the space used to store the original data set because the FP-tree requires additional space to store pointers between nodes and the counters for each item. The diagram below shows how a worst-case scenario FP-tree might appear. As you can see, the tree's complexity grows with each transaction's uniqueness.

Algorithm by Han

The original algorithm to construct the FP-Tree defined by Han is given below:

***Algorithm 1: FP-tree construction***

***Input*:** A transaction database DB and a minimum support threshold?

***Output*:** FP-tree, the frequent-pattern tree of DB.

***Method*:** The FP-tree is constructed as follows.

1. The first step is to scan the database to find the occurrences of the itemsets in the database. This step is the same as the first step of Apriori. The count of 1-itemsets in the database is called support count or frequency of 1-itemset.
2. The second step is to construct the FP tree. For this, create the root of the tree. The root is represented by null.
3. The next step is to scan the database again and examine the transactions. Examine the first transaction and find out the itemset in it. The itemset with the max count is taken at the top, and then the next itemset with the lower count. It means that the branch of the tree is constructed with transaction itemsets in descending order of count.
4. The next transaction in the database is examined. The itemsets are ordered in descending order of count. If any itemset of this transaction is already present in another branch, then this transaction branch would share a common prefix to the root.  
   This means that the common itemset is linked to the new node of another itemset in this transaction.
5. Also, the count of the itemset is incremented as it occurs in the transactions. The common node and new node count are increased by 1 as they are created and linked according to transactions.
6. The next step is to mine the created FP Tree. For this, the lowest node is examined first, along with the links of the lowest nodes. The lowest node represents the frequency pattern length 1. From this, traverse the path in the FP Tree. This path or paths is called a conditional pattern base.  
   A conditional pattern base is a sub-database consisting of prefix paths in the FP tree occurring with the lowest node (suffix).
7. Construct a Conditional FP Tree, formed by a count of itemsets in the path. The itemsets meeting the threshold support are considered in the Conditional FP Tree.
8. Frequent Patterns are generated from the Conditional FP Tree.

Using this algorithm, the FP-tree is constructed in two database scans. The first scan collects and sorts the set of frequent items, and the second constructs the FP-Tree.

**Example**

Support threshold=50%, Confidence= 60%

**Table 1:**

|  |  |
| --- | --- |
| **Transaction** | **List of items** |
| T1 | I1,I2,I3 |
| T2 | I2,I3,I4 |
| T3 | I4,I5 |
| T4 | I1,I2,I4 |
| T5 | I1,I2,I3,I5 |
| T6 | I1,I2,I3,I4 |

**Solution:** Support threshold=50% => 0.5\*6= 3 => min\_sup=3

**Table 2: Count of each item**

|  |  |
| --- | --- |
| **Item** | **Count** |
| I1 | 4 |
| I2 | 5 |
| I3 | 4 |
| I4 | 4 |
| I5 | 2 |

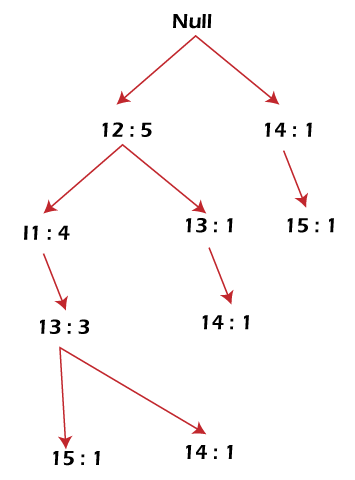
**Table 3: Sort the itemset in descending order.**

|  |  |
| --- | --- |
| **Item** | **Count** |
| I2 | 5 |
| I1 | 4 |
| I3 | 4 |
| I4 | 4 |

**Build FP Tree**

**Let's build the FP tree in the following steps, such as:**

1. Considering the root node null.
2. The first scan of Transaction T1: I1, I2, I3 contains three items {I1:1}, {I2:1}, {I3:1}, where I2 is linked as a child, I1 is linked to I2 and I3 is linked to I1.
3. T2: I2, I3, and I4 contain I2, I3, and I4, where I2 is linked to root, I3 is linked to I2 and I4 is linked to I3. But this branch would share the I2 node as common as it is already used in T1.
4. Increment the count of I2 by 1, and I3 is linked as a child to I2, and I4 is linked as a child to I3. The count is {I2:2}, {I3:1}, {I4:1}.
5. T3: I4, I5. Similarly, a new branch with I5 is linked to I4 as a child is created.
6. T4: I1, I2, I4. The sequence will be I2, I1, and I4. I2 is already linked to the root node. Hence it will be incremented by 1. Similarly I1 will be incremented by 1 as it is already linked with I2 in T1, thus {I2:3}, {I1:2}, {I4:1}.
7. T5:I1, I2, I3, I5. The sequence will be I2, I1, I3, and I5. Thus {I2:4}, {I1:3}, {I3:2}, {I5:1}.
8. T6: I1, I2, I3, I4. The sequence will be I2, I1, I3, and I4. Thus {I2:5}, {I1:4}, {I3:3}, {I4 1}.

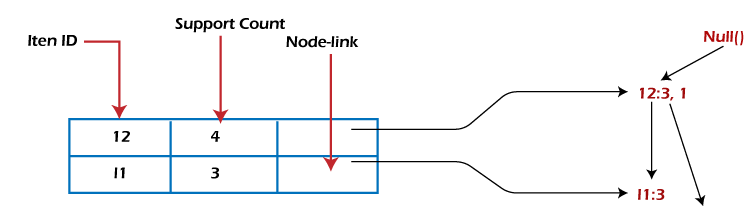


**Mining of FP-tree is summarized below:**

1. The lowest node item, I5, is not considered as it does not have a min support count. Hence it is deleted.
2. The next lower node is I4. I4 occurs in 2 branches , {I2,I1,I3:,I41},{I2,I3,I4:1}. Therefore considering I4 as suffix the prefix paths will be {I2, I1, I3:1}, {I2, I3: 1} this forms the conditional pattern base.
3. The conditional pattern base is considered a transaction database, and an FP tree is constructed. This will contain {I2:2, I3:2}, I1 is not considered as it does not meet the min support count.
4. This path will generate all combinations of frequent patterns : {I2,I4:2},{I3,I4:2},{I2,I3,I4:2}
5. For I3, the prefix path would be: {I2,I1:3},{I2:1}, this will generate a 2 node FP-tree : {I2:4, I1:3} and frequent patterns are generated: {I2,I3:4}, {I1:I3:3}, {I2,I1,I3:3}.
6. For I1, the prefix path would be: {I2:4} this will generate a single node FP-tree: {I2:4} and frequent patterns are generated: {I2, I1:4}.

|  |  |  |  |
| --- | --- | --- | --- |
| **Item** | **Conditional Pattern Base** | **Conditional FP-tree** | **Frequent Patterns Generated** |
| I4 | {I2,I1,I3:1},{I2,I3:1} | {I2:2, I3:2} | {I2,I4:2},{I3,I4:2},{I2,I3,I4:2} |
| I3 | {I2,I1:3},{I2:1} | {I2:4, I1:3} | {I2,I3:4}, {I1:I3:3}, {I2,I1,I3:3} |
| I1 | {I2:4} | {I2:4} | {I2,I1:4} |

The diagram given below depicts the conditional FP tree associated with the conditional node I3.



FP-Growth Algorithm

After constructing the FP-Tree, it's possible to mine it to find the complete set of frequent patterns. Han presents a group of lemmas and properties to do this job and then describes the following FP-Growth Algorithm.

**Algorithm 2: FP-Growth**

***Input*:** A database DB, represented by FP-tree constructed according to Algorithm 1, and a minimum support threshold?

***Output*:** The complete set of frequent patterns.

Advantages of FP Growth Algorithm

Here are the following advantages of the FP growth algorithm, such as:

* This algorithm needs to scan the database twice when compared to Apriori, which scans the transactions for each iteration.
* The pairing of items is not done in this algorithm, making it faster.
* The database is stored in a compact version in memory.
* It is efficient and scalable for mining both long and short frequent patterns.

Disadvantages of FP-Growth Algorithm

This algorithm also has some disadvantages, such as:

* FP Tree is more cumbersome and difficult to build than Apriori.
* It may be expensive.
* The algorithm may not fit in the shared memory when the database is large.

Difference between Apriori and FP Growth Algorithm

Apriori and FP-Growth algorithms are the most basic FIM algorithms. There are some basic differences between these algorithms, such as:

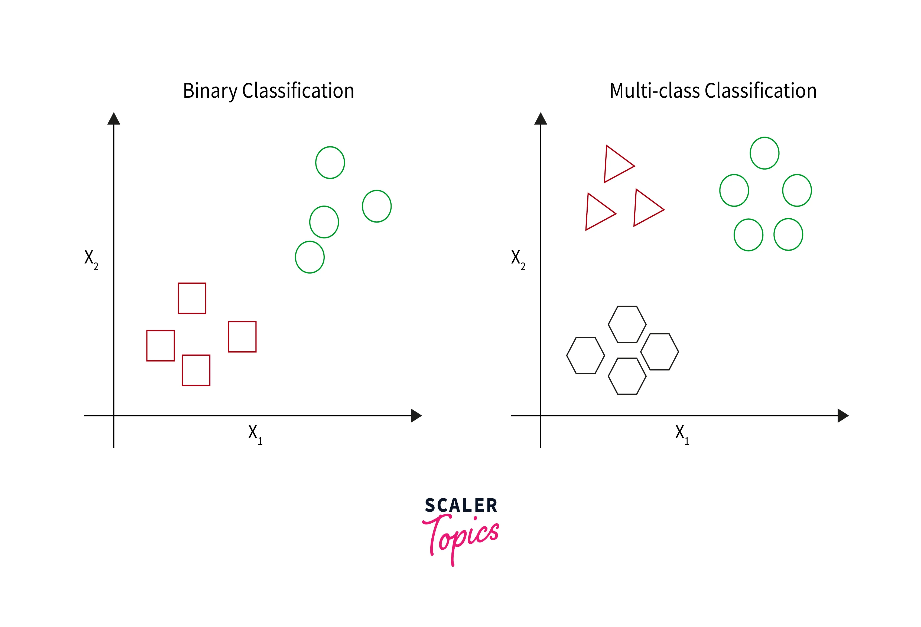
|  |  |
| --- | --- |
| **Apriori** | **FP Growth** |
| Apriori generates frequent patterns by making the itemsets using pairings such as single item set, double itemset, and triple itemset. | FP Growth generates an FP-Tree for making frequent patterns. |
| Apriori uses candidate generation where frequent subsets are extended one item at a time. | FP-growth generates a conditional FP-Tree for every item in the data. |
| Since apriori scans the database in each step, it becomes time-consuming for data where the number of items is larger. | FP-tree requires only one database scan in its beginning steps, so it consumes less time. |
| A converted version of the database is saved in the memory | A set of conditional FP-tree for every item is saved in the memory |
| It uses a breadth-first search | It uses a depth-first search. |

**3. Performance Evaluation of Algorithms**

Performance evaluation of algorithms in data mining and data warehouses involves choosing appropriate evaluation methods and metrics to measure and compare the results of applied techniques and models against established objectives and criteria. The goal is to ensure the quality and effectiveness of the results.

Here are some things to consider when evaluating the performance of algorithms in data mining and data warehouses:

* Data quality: Ensure the data is high quality before evaluating the model.
* Training and testing: Split the data into two subsets, a training set and a testing set.
* Evaluation metrics: Use different metrics depending on the type and purpose of the model. Some examples include accuracy, precision, recall, F-measure, ROC curve, AUC, lift, or profit.
* Confusion matrix: This is another useful tool for evaluating the model.
* Benchmark: Compare the results of the data mining process against a pre-existing benchmark, or "ground truth". This could be a dataset with a known outcome that allows you to measure how often the data mining correctly identifies patterns or classifications.



* Dataset characteristics: The performance of algorithms may be related to the characteristics of the dataset. For example, classification algorithms may perform better on large datasets with many attributes.

**Key Metrics:**

* **Execution Time:** Measures the total time taken by the algorithm to mine frequent patterns.
* **Memory Usage:** Evaluates how much memory the algorithm consumes during execution.
* **Scalability:** How well the algorithm handles increasing data sizes.
* **Accuracy:** Ensures that the algorithm correctly identifies all frequent patterns without missing any or including false positives.

**Performance Factors:**

* **Dataset Characteristics:** The size of the dataset, the distribution of items, and the minimum support threshold can significantly affect performance.
* **Algorithm Complexity:** Algorithms with lower computational complexity (e.g., FP-Growth) tend to perform better on large datasets compared to those with higher complexity (e.g., Apriori).

**Evaluation Process:**

* **Experimental Setup:** Use a variety of datasets to assess the algorithm's performance under different conditions.
* **Comparison:** Compare the algorithm with other frequent pattern mining algorithms like Apriori, Eclat, and others.

**Classification** - introduction

Classification in data mining is a technique used to assign labels or classify each instance, record, or data object in a dataset based on their features or attributes. The objective of the classification approach is to predict class labels of new, unseen data accurately. It is an important task in data mining because it enables organizations to make data-driven decisions. For example, businesses can assign or classify sentiments of customer feedback, reviews, or social media posts to understand how well their products or services are doing.

Classification techniques can be divided into categories - binary classification and multi-class classification. Binary classification assigns labels to instances into two classes, such as fraudulent or non-fraudulent. Multi-class classification assigns labels into more than two classes, such as happy, neutral, or sad.

**Steps to Build a Classification Model**

There are several steps involved in building a classification model, as shown below -

* **Data preparation** - The first step in building a classification model is to prepare the data. This involves collecting, cleaning, and transforming the data into a suitable format for further analysis.
* **Feature selection** - The next step is to select the most important and relevant features that will be used to build the classification model. This can be done using various techniques, such as correlation, feature importance analysis, or domain knowledge.
* **Prepare train and test data** - Once the data is prepared and relevant features are selected, the dataset is divided into two parts - training and test datasets. The training set is used to build the model, while the testing set is used to evaluate the model's performance.
* **Model selection** - Many algorithms can be used to build a classification model, such as decision trees, logistic regression, k-nearest neighbors, and neural networks. The choice of algorithm depends on the type of data, the number of features, and the desired accuracy.
* **Model training** - Once the algorithm is selected, the model is trained on the training dataset. This involves adjusting the model parameters to minimize the error between the predicted and actual class labels.
* **Model evaluation** - The model's performance is evaluated using the test dataset. The accuracy, precision, recall, and F1 score are commonly used metrics to evaluate the model performance.
* **Model tuning** - If the model's performance is not satisfactory, the model can be tuned by adjusting the parameters or selecting a different algorithm. This process is repeated until the desired performance is achieved.
* **Model deployment** - Once the model is built and evaluated, it can be deployed in production to classify new data. The model should be monitored regularly to ensure its accuracy and effectiveness over time.

**Syntaxes Used**

Here are some common notations and syntax used for classification in data mining -

* **X** - Input data matrix or feature matrix, where each row represents an observation or data point, and each column represents a feature or attribute.
* **y** - Output or target variable vector, where each element represents the class label or target variable for the corresponding data point in **X**.
* **p(y|x)** - Probability of class **y** given input **x**.
* **θ** - Model parameters or coefficients that are learned during the training process.
* **J(θ)** - Cost function that measures the overall error or loss of the model on the training data and is typically a function of the model parameters **θ**.

**Categorization of Classification in Data Mining**

There are different types of classification algorithms based on their approach, complexity, and performance. Here are some common categorizations of classification in data mining -

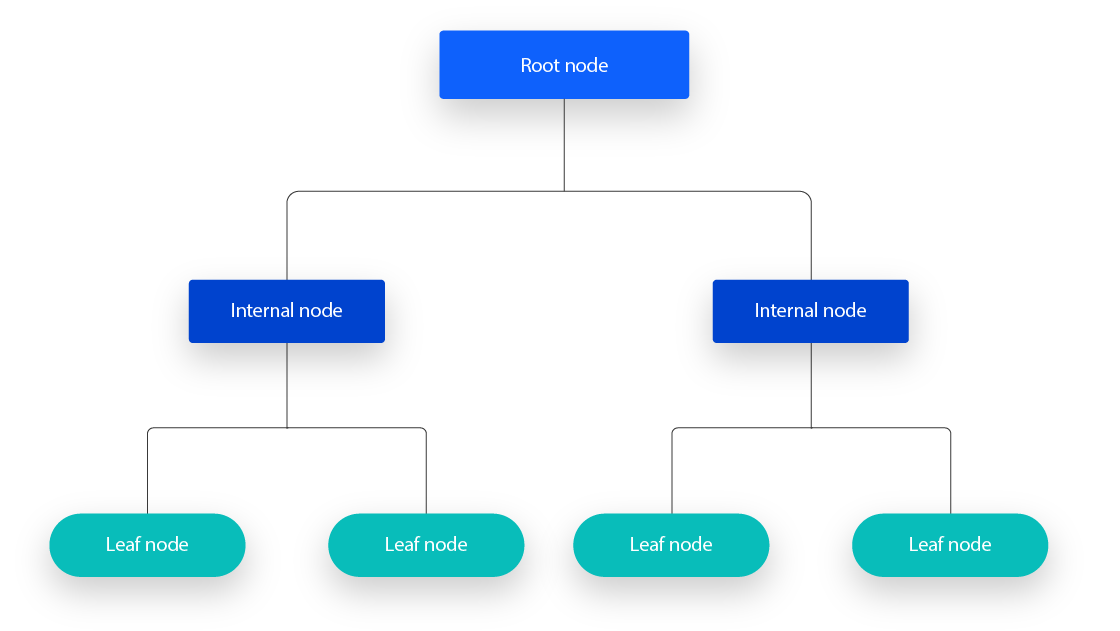
* **Decision tree-based classification** - This type of classification algorithm builds a tree-like model of decisions and their possible consequences. Decision trees are easy to understand and interpret, making them a popular choice for classification problems.
* **Rule-based classification** - This type of classification algorithm uses a set of rules to determine the class label of an observation. The rules are typically expressed in the form of IF-THEN statements, where each statement represents a condition and a corresponding action.
* **Instance-based classification** - This type of classification algorithm uses a set of training instances to classify new, unseen instances. The classification is based on the similarity between the training instances' features and the new instances' features.
* **Bayesian classification** - This classification algorithm uses Bayes' theorem to compute the probability of each class label given the observed features. Bayesian classification is particularly useful when dealing with incomplete or uncertain data.
* **Neural network-based classification** - This classification algorithm uses a network of interconnected nodes or neurons to learn a mapping between the input features and the output class labels. Neural networks can handle complex and nonlinear relationships between the features and the class labels.
* **Ensemble-based classification** - This classification algorithm combines the predictions of multiple classifiers to improve the overall accuracy and robustness of the classification model. Ensemble methods include bagging, boosting, and stacking.

Difference between classification ad regression-

| **Factors** | **Classification** | **Regression** |
| --- | --- | --- |
| Task/Objective | Identifying or assigning the class label of a new observation based on its features. | Estimating a continuous or discrete value for a new observation based on its features. |
| Outcome | Categorical variable, i.e., a class label or category. | Continuous or discrete variable, i.e., a numeric value. |
| Evaluation | Accuracy, precision, recall, F1 score, AUC. | Mean squared error, root mean squared error, correlation coefficient. |
| Algorithms | Decision trees, rule-based systems, neural networks, support vector machines, k-nearest neighbors. | Linear regression, logistic regression, polynomial regression, time series analysis, neural networks. |
| Examples | Spam email classification, sentiment analysis, fraud detection, etc. | Housing price prediction, stock price prediction, predicting a customer's purchase amount or sale, etc. |

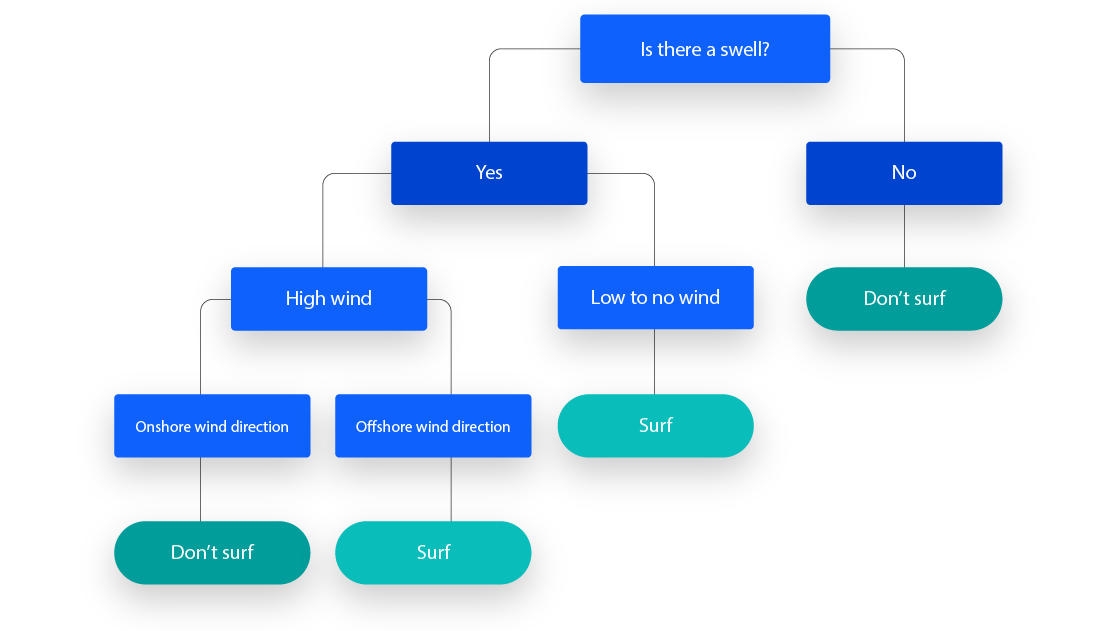
**Decision tree**

A decision tree is a non-parametric supervised learning algorithm, which is utilized for both classification and regression tasks. It has a hierarchical, tree structure, which consists of a root node, branches, internal nodes and leaf nodes.



As you can see from the diagram above, a decision tree starts with a root node, which does not have any incoming branches. The outgoing branches from the root node then feed into the internal nodes, also known as decision nodes. Based on the available features, both node types conduct evaluations to form homogenous subsets, which are denoted by leaf nodes, or terminal nodes. The leaf nodes represent all the possible outcomes within the dataset.

As an example, let’s imagine that you were trying to assess whether or not you should go surf, you may use the following decision rules to make a choice:



This type of flowchart structure also creates an easy to digest representation of decision-making, allowing different groups across an organization to better understand why a decision was made.

Decision tree learning employs a divide and conquer strategy by conducting a greedy search to identify the optimal split points within a tree. This process of splitting is then repeated in a top-down, recursive manner until all, or the majority of records have been classified under specific class labels. Whether or not all data points are classified as homogenous sets is largely dependent on the complexity of the decision tree. Smaller trees are more easily able to attain pure leaf nodes—i.e. data points in a single class. However, as a tree grows in size, it becomes increasingly difficult to maintain this purity, and it usually results in too little data falling within a given subtree. When this occurs, it is known as data fragmentation, and it can often lead to [overfitting](https://www.ibm.com/topics/overfitting). As a result, decision trees have preference for small trees, which is consistent with the principle of parsimony in Occam’s Razor; that is, “entities should not be multiplied beyond necessity.” Said differently, decision trees should add complexity only if necessary, as the simplest explanation is often the best. To reduce complexity and prevent overfitting, pruning is usually employed; this is a process, which removes branches that split on features with low importance. The model’s fit can then be evaluated through the process of cross-validation. Another way that decision trees can maintain their accuracy is by forming an ensemble via a [random forest](https://www.ibm.com/topics/random-forest) algorithm; this classifier predicts more accurate results, particularly when the individual trees are uncorrelated with each other.

Types of Decision Trees

Hunt’s algorithm, which was developed in the 1960s to model human learning in Psychology, forms the foundation of many popular decision tree algorithms, such as the following:

**- ID3:**Ross Quinlan is credited within the development of ID3, which is shorthand for “Iterative Dichotomiser 3.” This algorithm leverages entropy and information gain as metrics to evaluate candidate splits.

**- C4.5:**This algorithm is considered a later iteration of ID3, which was also developed by Quinlan. It can use information gain or gain ratios to evaluate split points within the decision trees.

**- CART:**The term, CART, is an abbreviation for “classification and regression trees” and was introduced by Leo Breiman. This algorithm typically utilizes Gini impurity to identify the ideal attribute to split on. Gini impurity measures how often a randomly chosen attribute is misclassified. When evaluating using Gini impurity, a lower value is more ideal.

**Advantages**

**- Easy to interpret:**The Boolean logic and visual representations of decision trees make them easier to understand and consume. The hierarchical nature of a decision tree also makes it easy to see which attributes are most important, which isn’t always clear with other algorithms, like [neural networks](https://www.ibm.com/topics/neural-networks).

**- Little to no data preparation required:**Decision trees have a number of characteristics, which make it more flexible than other classifiers. It can handle various data types—i.e. discrete or continuous values, and continuous values can be converted into categorical values through the use of thresholds. Additionally, it can also handle values with missing values, which can be problematic for other classifiers, like Naïve Bayes.

**- More flexible:** Decision trees can be leveraged for both classification and regression tasks, making it more flexible than some other algorithms. It’s also insensitive to underlying relationships between attributes; this means that if two variables are highly correlated, the algorithm will only choose one of the features to split on.

**Disadvantages**

**- Prone to overfitting:** Complex decision trees tend to overfit and do not generalize well to new data. This scenario can be avoided through the processes of pre-pruning or post-pruning. Pre-pruning halts tree growth when there is insufficient data while post-pruning removes subtrees with inadequate data after tree construction.

**- High variance estimators:**Small variations within data can produce a very different decision tree. [Bagging](https://www.ibm.com/topics/bagging), or the averaging of estimates, can be a method of reducing variance of decision trees. However, this approach is limited as it can lead to highly correlated predictors.

**- More costly:**Given that decision trees take a greedy search approach during construction, they can be more expensive to train compared to other algorithms.

**Tree induction algorithm – split algorithm based on information theory**

Tree induction is a method used in [machine learning](https://deepai.org/machine-learning-glossary-and-terms/machine-learning) to derive [decision trees](https://deepai.org/machine-learning-glossary-and-terms/decision-tree) from data. Decision trees are predictive models that use a set of binary rules to calculate a target value. They are widely used for classification and regression tasks because they are interpretable, easy to implement, and can handle both numerical and categorical data. Tree induction algorithms work by recursively partitioning the dataset into subsets based on the features that provide the best separation between classes or values.

**How Tree Induction Works**

The goal of tree induction is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. The process begins with the entire dataset and divides it into subsets based on a selected feature. This is done recursively on each derived subset in a recursive manner. The recursion is completed when the algorithm determines that further splits will not add value to the predictions.

**Steps in Tree Induction**

1. [**Feature Selection**](https://deepai.org/machine-learning-glossary-and-terms/feature-selection)**:**

At each node in the tree, the algorithm selects the best feature to split the data. This is typically done using a measure of impurity or information gain, such as Gini impurity, entropy, or [variance](https://deepai.org/machine-learning-glossary-and-terms/variance) reduction.

1. **Partitioning:** The dataset is split into subsets based on the feature values. For categorical features, this could mean creating a branch for each category. For continuous features, the data is split at a value that maximizes the separation between classes.
2. **Recursion:** Steps 1 and 2 are applied recursively to each derived subset until the stopping criteria are met. Common stopping criteria include a maximum tree depth, a minimum number of samples required to split a node, or a minimum gain in impurity reduction.
3. **Pruning:** To prevent overfitting, the tree may be pruned back by removing branches that have little predictive power. This can be done using methods like reduced error pruning or cost complexity pruning.

**Algorithms for Tree Induction**

Several algorithms have been developed for tree induction, each with its own approach to feature selection and tree construction. Some of the most well-known algorithms include:

* **ID3 (Iterative Dichotomiser 3):** This algorithm uses entropy and information gain to build a decision tree for classification tasks.
* **C4.5:** An extension of ID3, C4.5 uses the gain ratio to address some of the limitations of information gain and can handle both continuous and discrete features.
* **CART (Classification and Regression Trees):** CART is a versatile algorithm that can be used for both classification and regression. It uses Gini impurity for classification and variance reduction for regression.

**Advantages and Disadvantages of Tree Induction**

**Advantages:**

* **Interpretability:** Decision trees can be easily visualized and understood, even by those with little knowledge of machine learning.
* **Handling mixed data:** Trees can handle both numerical and categorical data without the need for preprocessing.
* **Non-linearity:** Trees can model non-linear relationships between features and the target variable.

**Disadvantages:**

* **Overfitting:** Trees can easily overfit the training data, especially if they are allowed to grow deep without pruning.
* **Instability:** Small changes in the data can lead to very different trees being generated.
* **Performance:** While trees are simple and interpretable, they often do not have the predictive accuracy of more complex models.

**Applications of Tree Induction**

Tree induction is used in various domains, including:

* **Medical Diagnosis:** Decision trees can help in diagnosing diseases by analyzing patient data and identifying key symptoms and test results.
* **Financial Analysis:** In finance, trees can be used for credit scoring, fraud detection, and risk assessment.
* **Customer Segmentation:** Marketing teams use decision trees to segment customers based on purchasing behavior and preferences.

| **Aspect** | **Decision Tree** | **Decision Induction Tree** |
| --- | --- | --- |

|  |  |  |
| --- | --- | --- |
| **Definition** | A tree structure representing decisions and their outcomes. | The process of constructing a decision tree through an induction algorithm. |

|  |  |  |
| --- | --- | --- |
| **Focus** | The final model used for prediction or classification. | The method and process used to create the decision tree. |

|  |  |  |
| --- | --- | --- |
| **Construction** | Constructed top-down with splits based on criteria like information gain or Gini index. | Built using inductive learning algorithms that partition data recursively. |

|  |  |  |
| --- | --- | --- |
| **Purpose** | Used for making decisions or predictions on new data. | Used to learn and generalize patterns from a dataset. |

|  |  |  |
| --- | --- | --- |
| **Algorithm Emphasis** | Emphasizes the final tree structure and its use in prediction. | Emphasizes the algorithm and process of building the tree. |

|  |  |  |  |
| --- | --- | --- | --- |
| **Practical Use** | | Commonly applied in classification and regression tasks. | More of a conceptual framework describing how the tree is built. |
| **Output** | A completed model that can be applied to new data for predictions. | | The process or method that generates the Decision Tree model. |

|  |  |  |
| --- | --- | --- |
| **Emphasis** | Focuses on the structure and nodes that lead to a decision. | Focuses on the learning process and the criteria for constructing the tree. |

|  |  |  |
| --- | --- | --- |
| **End Product vs. Process** | Represents the final outcome used for decision-making. | Represents the iterative process used to build the final tree. |

|  |  |  |
| --- | --- | --- |
| **Algorithm Examples** | The result of algorithms like CART, C4.5, and ID3. | The application of algorithms such as ID3, C4.5, and CART to induce the tree. |

|  |  |  |
| --- | --- | --- |
| **Complexity** | Can be simple to interpret once constructed. | Involves complexity in deciding how to split nodes and manage tree growth. |

|  |  |  |
| --- | --- | --- |
| **Pruning Application** | Pruning is applied to the tree after it is constructed to simplify it. | Pruning is part of the induction process to avoid overfitting during tree construction. |

|  |  |  |
| --- | --- | --- |
| **Visual Representation** | Can be visualized as a tree structure with nodes and branches. | The process is not directly visualizable but can be understood through algorithm steps. |

|  |  |  |
| --- | --- | --- |
| **Use Case** | Directly used for making predictions based on input data. | Used during training to learn patterns and relationships in the data. |

|  |  |  |
| --- | --- | --- |
| **Performance Focus** | Evaluated based on accuracy, interpretability, and prediction performance. | Evaluated based on efficiency, scalability, and the ability to generalize. |

|  |  |  |
| --- | --- | --- |
| **Data Interaction** | Operates on the tree model for decision-making. | Interacts with the data to determine the structure of the tree during training. |

|  |  |  |
| --- | --- | --- |
| **User Interaction** | Users interact with the tree to make decisions or predictions. | Data scientists interact with the induction process to tune and optimize the model. |

**Split algorithm based on gini index**

The Gini Index is a way of quantifying how messy or clean a dataset is, especially when we use decision trees to classify it. It goes from 0 (cleanest, all data points have the same label) to 1 (messiest, data points are split evenly among all labels).

Think of a dataset that shows how much money people make. A high Gini Index for this data means that there is a huge difference between the rich and the poor, while a low Gini Index means that the income is more balanced.

When we build decision trees, we want to use the Gini Index to find the best feature to split the data at each node. The best feature is the one that reduces the Gini Index the most, meaning that it creates the purest child nodes. This way, we can create a tree that can distinguish different labels based on the features.

**What Does a Decision Tree do?**

A decision tree is a [machine learning algorithm](https://www.upgrad.com/blog/types-of-machine-learning-algorithms/) used for both classification and regression tasks. It resembles a tree-like structure with branches and leaves. Each branch represents a decision based on a specific feature of the data, and the leaves represent the predicted outcome.

Data points navigate through the decision tree based on their respective feature values, traversing down branches determined by the split conditions that are chosen using the decision tree Gini index as a criterion for selection. Ultimately, they reach a leaf and receive the prediction assigned to that leaf. Decision trees are popular for their interpretability and simplicity, allowing easy visualization of the decision-making process. The Gini Index plays a crucial role in building an effective decision tree by guiding the selection of optimal splitting features. By minimizing the Gini index for decision tree at each node, the tree progressively separates data points belonging to different classes, leading to accurate predictions at the terminal leaves.

Here’s a breakdown of how to build decision tree using Gini index:

1. **Calculate the Gini Index of the entire dataset.** This represents the initial level of impurity before any splitting.
2. **Consider each feature and its threshold values.** For each combination, calculate the Gini Index of the two resulting child nodes after splitting the data based on that feature and threshold.
3. **Choose the feature and threshold combination that leads to the smallest Gini Index for the child nodes.** This indicates the most significant decrease in impurity, resulting in a more homogeneous separation of data points.
4. **Repeat the process recursively on each child node.** Use the same approach to select the next split feature and threshold, further minimizing the Gini Index and separating data points based on their class labels.
5. **Continue splitting until a stopping criterion is met.** This could be reaching a pre-defined tree depth, minimum data size per node, or a sufficiently low Gini Index at all terminal leaves.

 By iteratively using the Decision Tree Gini Index to guide feature selection and data partitioning, decision trees can effectively learn complex relationships within the data and make accurate predictions for unseen instances.

**Flow of a Decision Tree**

Here I have noted the flow of a decision tree Gini index:

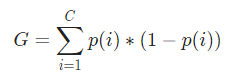
1. **Training:** The decision tree is built by applying a splitting algorithm to the training data. The algorithm chooses the feature and its threshold value that best minimizes the Gini Index within the resulting child nodes. This process is repeated recursively on each subgroup until reaching a stopping criterion, like minimum data size or maximum tree depth.
2. **Prediction:** A new data point traverses the tree based on its own feature values, navigating down branches determined by the splitting conditions. Finally, it reaches a leaf and receives the prediction assigned to that leaf.
3. **Ensembles:** Decision trees can be combined into ensembles like random forests or boosting to improve accuracy and reduce overfitting. This involves building multiple trees from different subsets of the data and aggregating their predictions, leading to a more robust model.

**Calculation**

The Gini Index or Gini Impurity is calculated by subtracting the sum of the squared probabilities of each class from one. It favours mostly the larger partitions and are very simple to implement. In simple terms, it calculates the probability of a certain randomly selected feature that was classified incorrectly.

The Gini Index varies between 0 and 1, where 0 represents purity of the classification and 1 denotes random distribution of elements among various classes. A Gini Index of 0.5 shows that there is equal distribution of elements across some classes.

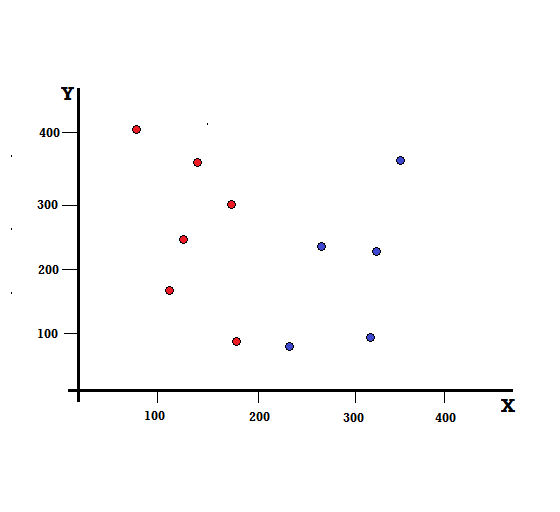
Mathematically, The Gini Index is represented by



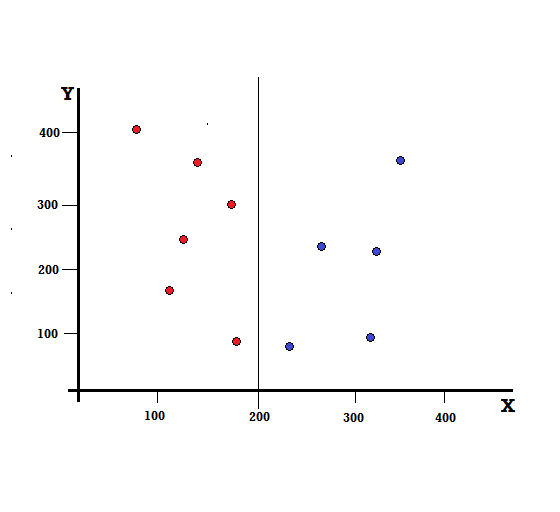
The Gini Index works on categorical variables and gives the results in terms of “success” or “failure” and hence performs only binary split. It isn’t computationally intensive as its counterpart – Information Gain. From the Gini Index, the value of another parameter named Gini Gain is calculated whose value is maximized with each iteration by the Decision Tree to get the perfect CART

**FYI:** [Free NLP course](https://www.upgrad.com/blog/deep-learning-free-online-course/)!

Let us understand the calculation of the Gini Index with a simple example. In this, we have a total of 10 data points with two variables, the reds and the blues. The X and Y axes are numbered with spaces of 100 between each term. From the given Gini index Decision tree example , we shall calculate the Gini Index and the Gini Gain.

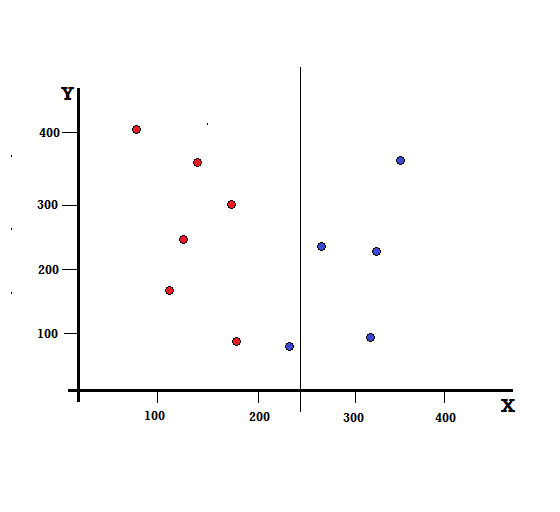


For a decision tree, we need to split the dataset into two branches. Consider the following data points with 5 Reds and 5 Blues marked on the X-Y plane. Suppose we make a binary split at X=200, then we will have a perfect split as shown below.



It is seen that the split is correctly performed and we are left with two branches each with 5 reds (left branch) and 5 blues (right branch).

But what will be the outcome if we make the split at X=250?



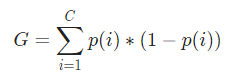
We are left with two branches, the left branch consisting of 5 reds and 1 blue, while the right branch consists of 4 blues. The following is referred to as an imperfect split. In training the Decision Tree model, to quantify the amount of imperfectness of the split, we can use the Gini Index.

**Basic Mechanism**

To calculate the Gini Impurity, let us first understand it’s basic mechanism.

* First, we shall randomly pick up any data point from the dataset
* Then, we will classify it randomly according to the class distribution in the given dataset. In our dataset, we shall give a data point chosen with a probability of 5/10 for red and 5/10 for blue as there are five data points of each colour and hence the probability.

Now, in order to calculate the Gini index decision tree formula:



Where, C is the total number of classes and p(*i*) is the probability of picking the data point with the class *i.*

In the above Gini index decision tree solved example, we have C=2 and p(1) = p(2) = 0.5, Hence the Gini Index can be calculated as,

*G =p(1) ∗ (1−p(1)) + p(2) ∗ (1−p(2))*

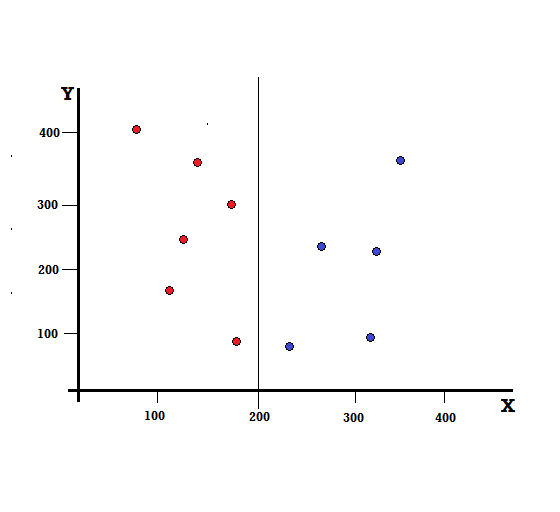
*=0.5 ∗ (1−0.5) + 0.5 ∗ (1−0.5)*

*=0.5*

Where 0.5 is the total probability of classifying a data point imperfectly and hence is exactly 50%.

Now, let us calculate the Gini Impurity for both the perfect and imperfect split that we performed earlier,

**Perfect Split**



The left branch has only reds and hence its Gini Impurity is,

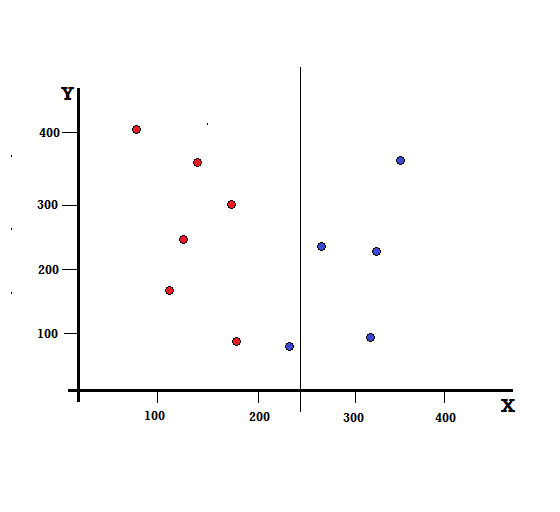
*G(left) =1 ∗ (1−1) + 0 ∗ (1−0) = 0*

The right branch also has only blues and hence its Gini Impurity is also given by,

*G(right) =1 ∗ (1−1) + 0 ∗ (1−0) = 0*

From the quick calculation, we see that both the left and right branches of our perfect split have probabilities of 0 and hence is indeed perfect. A Gini Impurity of 0 is the lowest and the best possible impurity for any data set.

**Imperfect Split**



In this case, the left branch has 5 reds and 1 blue. Its Gini Impurity can be given by,

*G(left) =1/6 ∗ (1−1/6) + 5/6 ∗ (1−5/6) = 0.278*

The right branch has all blues and hence as calculated above its Gini Impurity is given by,

*G(right) =1 ∗ (1−1) + 0 ∗ (1−0) = 0*

Now that we have the Gini Impurities of the imperfect split, in order to evaluate the quality or extent of the split, we will give a specific weight to the impurity of each branch with the number of elements it has.

*(0.6∗0.278) + (0.4∗0) = 0.167*

Now that we have calculated the Gini Index, we shall calculate the value of another parameter, Gini Gain and analyse its application in Decision Trees. The amount of impurity removed with this split is calculated by deducting the above value with the Gini Index for the entire dataset (0.5)

*0.5 – 0.167 = 0.333*

This value calculated is called as the “**Gini Gain**”. In simple terms, **Higher Gini Gain = Better Split**.

Hence, in a Decision Tree algorithm, the best split is obtained by maximizing the Gini Gain, which is calculated in the above manner with each iteration.

After calculating the Gini Gain for each attribute in the data set, the class, sklearn.tree.DecisionTreeClassifier will choose the largest Gini Gain as the Root Node. When a branch with Gini of 0 is encountered it becomes the leaf node and the other branches with Gini more than 0 need further splitting. These nodes are grown recursively till all of them are classified.

**Relevance of Entropy**

Entropy, a key concept in decision trees, measures the *uncertainty* or *randomness* within a dataset. It specifically quantifies the degree to which a subset of data contains examples belonging to different classes, playing a crucial role in the decision-making process of the tree. By choosing features that minimize entropy within splits, we lead to purer branches and, ultimately, construct a more accurate decision tree.

While both the Gini Index and entropy are utilized in decision trees to assess data purity, they calculate the difference in impurity slightly differently. The Gini Index, like entropy, serves as a metric to evaluate the likelihood of a specific feature being misclassified when selected randomly. However, entropy in the decision tree gives a more detailed measure of the disorder or variability of the system, offering a slightly different perspective on data purity and impurity reduction strategies.

* **Gini Index:** Compares the proportion of each class within a data subset before and after the split, favoring features that maximize the difference.
* **Entropy:** Compares the overall uncertainty of the original data to the combined uncertainty of the resulting subsets, preferring features that lead to the largest decrease in overall entropy.

Both Gini Index and entropy have their advantages and disadvantages, and the choice depends on the specific data and task. Generally, Gini Index works well for binary classification, while entropy might be better suited for multiple classes.

**Difference between Gini Index and Entropy**

|  |  |  |
| --- | --- | --- |
| **Factor** | **Gini Index** | **Entropy** |
| **Definition** | Measures the probability of misclassification. | Measures the amount of information (or uncertainty) in a dataset. |
| **Formula** | Gini=1−∑i=1n​pi2​ | Entropy=−∑i=1n​pi​log2​(pi​) |
| **Range** | 0 to 0.5 for binary classification. | 0 to 1 for binary classification. |
| **Impurity** | Lower values indicate purer nodes. | Lower values indicate purer nodes. |
| **Calculation Complexity** | Generally simpler to compute. | Generally more complex to compute. |
| **Splitting Criterion** | Prefers to maximize the probability of a single class. | Prefers splits that create the most uniform class distribution. |
| **Use in Algorithms** | Commonly used in the CART (Classification and Regression Tree) algorithm. | Commonly used in the ID3 (Iterative Dichotomiser 3) and C4.5 algorithms. |
| **Sensitivity to Data Distribution** | Less sensitive to changes in class distribution. | More sensitive to changes in class distribution. |
| **Interpretation** | Measures how often a randomly chosen element would be incorrectly classified. | Measures the average amount of information required to identify the class of an element. |
| **Bias Towards Purity** | Slightly biased towards larger classes. | More balanced, less biased towards larger or smaller classes. |
| **Behavior at Pure Nodes** | At a pure node (one class), Gini = 0. | At a pure node (one class), Entropy = 0. |
| **Mathematical Nature** | Quadratic measure. | Logarithmic measure. |
| **Robustness to Outliers** | More robust to outliers due to its quadratic nature. | Less robust to outliers due to the logarithmic calculation. |
| **Preferred When** | Simplicity and speed are crucial. | A more nuanced measure of information gain is needed. |

**Gini Index vs Information Gain**

Both Gini Index and Information Gain are measures of impurity used in decision trees to choose the best feature for splitting the data at each node. However, they calculate this difference in slightly different ways and have their own strengths and weaknesses.

**Gini Index:**

* **Focuses on class proportions:** Compares the proportion of each class within a data subset before and after the split, favoring features that maximize the difference. This makes it sensitive to class imbalance, potentially favoring splits that isolate minority classes even if they don’t significantly improve overall clarity.
* **Simple and computationally efficient:** Easier to calculate compared to Information Gain, making it faster to build decision trees.
* **Works well for binary classification:** Emphasizes maximizing the gap between classes, making it effective when dealing with two distinct outcomes.

**Information Gain:**

* **Measures entropy change:** Compares the total entropy of the original data to the combined entropy of the resulting subsets after the split, preferring features that lead to the largest decrease in overall uncertainty. This is more nuanced and can handle multiple classes effectively.
* **Less sensitive to class imbalance:** Doesn’t solely focus on isolating minority classes but accounts for overall reduction in uncertainty even if the split proportions are uneven.
* **More computationally expensive:** Calculating entropy involves logarithms, making it slightly slower than Gini Index for tree construction.
* **Can be better for multi-class problems:** Provides a more comprehensive picture of class distribution changes, potentially leading to better results with multiple outcomes.

**Naïve bayes method**

The Naive Bayes method is a probabilistic classifier and machine learning algorithm that's used for classification tasks like text classification. It's based on Bayes' theorem, which calculates probabilities by counting the frequency of values and combinations of values in historical data.

Naive Bayes uses the Bayes’ Theorem and assumes that all predictors are independent. In other words, this classifier assumes that the presence of one particular feature in a class doesn’t affect the presence of another one.

Here’s an example: you’d consider fruit to be orange if it is round, orange, and is of around 3.5 inches in diameter. Now, even if these features require each other to exist, they all contribute independently to your assumption that this particular fruit is orange. That’s why this algorithm has ‘Naive’ in its name.

Building the Naive Bayes model is quite simple and helps you in working with vast datasets. Moreover, this equation is popular for beating many advanced classification techniques in terms of performance.

**Here’s the equation for Naive Bayes:**

P (c|x) = P(x|c) P(c) / P(x)

P(c|x) = P(x1 | c) x P(x2 | c) x … P(xn | c) x P(c)

Here, P (c|x) is the posterior probability according to the predictor (x) for the class(c). P(c) is the prior probability of the class, P(x) is the prior probability of the predictor, and P(x|c) is the probability of the predictor for the particular class(c).

Apart from considering the independence of every feature, Naive Bayes also assumes that they contribute equally. This is an important point to remember.

**How does Naive Bayes Work?**

To understand how Naive Bayes works, we should discuss an example.

Suppose we want to find stolen cars and have the following dataset:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Serial No.** | **Color** | **Type** | **Origin** | **Was it Stolen?** |
| 1 | Red | Sports | Domestic | Yes |
| 2 | Red | Sports | Domestic | No |
| 3 | Red | Sports | Domestic | Yes |
| 4 | Yellow | Sports | Domestic | No |
| 5 | Yellow | Sports | Imported | Yes |
| 6 | Yellow | SUV | Imported | No |
| 7 | Yellow | SUV | Imported | Yes |
| 8 | Yellow | SUV | Domestic | No |
| 9 | Red | SUV | Imported | No |
| 10 | Red | Sports | Imported | Yes |

**According to our dataset, we can understand that our algorithm makes the following assumptions:**

* It assumes that every feature is independent. For example, the colour ‘Yellow’ of a car has nothing to do with its Origin or Type.
* It gives every feature the same level of importance. For example, knowing only the Color and Origin would predict the outcome correctly. That’s why every feature is equally important and contributes equally to the result.

Now, with our dataset, we have to classify if thieves steal a car according to its features. Each row has individual entries, and the columns represent the features of every car. In the first row, we have a stolen Red Sports Car with Domestic Origin. We’ll find out if thieves would steal a Red Domestic SUV or not (our dataset doesn’t have an entry for a Red Domestic SUV).

**We can rewrite the Bayes Theorem for our example as:**

P(y | X) = [P(X | y) P(y)P(X)]/P(X)

Here, y stands for the class variable (Was it Stolen?) to show if the thieves stole the car not according to the conditions. X stands for the features.

X = x1, x2, x3, …., xn)

Here, x1, x2,…, xn stand for the features. We can map them to be Type, Origin, and Color. Now, we’ll replace X and expand the chain rule to get the following:

P(y | x1, …, xn) = [P(x1 | y) P(x2 | y) … P(xn | y) P(y)]/[P(x1) P (x2) … P(xn)]

You can get the values for each by using the dataset and putting their values in the equation. The denominator will remain static for every entry in the dataset to remove it and inject proportionality.

P(y | x1, …, xn) ∝ P(y) i = 1nP(xi | y)

In our example, y only has two outcomes, yes or no.

y = argmaxyP(y) i = 1nP(xi | y)

We can create a Frequency Table to calculate the posterior probability P(y|x) for every feature. Then, we’ll mould the frequency tables to Likelihood Tables and use the Naive Bayesian equation to find every class’s posterior probability. The result of our prediction would be the class that has the highest posterior probability. Here are the Likelihood and Frequency Tables:

**Frequency Table of Color:**

|  |  |  |
| --- | --- | --- |
| **Color** | **Was it Stolen (Yes)** | **Was it Stolen (No)** |
| Red | 3 | 2 |
| Yellow | 2 | 3 |

**Likelihood Table of Color:**

|  |  |  |
| --- | --- | --- |
| **Color** | **Was it Stolen [P(Yes)]** | **Was it Stolen [P(No)]** |
| Red | 3/5 | 2/5 |
| Yellow | 2/5 | 3/5 |

**Frequency Table of Type:**

|  |  |  |
| --- | --- | --- |
| **Type** | **Was it Stolen (Yes)** | **Was it Stolen (No)** |
| Sports | 4 | 2 |
| SUV | 1 | 3 |

**Likelihood Table of Type:**

|  |  |  |
| --- | --- | --- |
| **Type** | **Was it Stolen [P(Yes)]** | **Was it Stolen [P(No)]** |
| Sports | 4/5 | 2/5 |
| SUV | 1/5 | 3/5 |

**Frequency Table of Origin:**

|  |  |  |
| --- | --- | --- |
| **Origin** | **Was it Stolen (Yes)** | **Was it Stolen (No)** |
| Domestic | 2 | 3 |
| Imported | 3 | 2 |

**Likelihood Table of Origin:**

|  |  |  |
| --- | --- | --- |
| **Origin** | **Was it Stolen [P(Yes)]** | **Was it Stolen [P(No)]** |
| Domestic | 2/5 | 3/5 |
| Imported | 3/5 | 2/5 |

Our problem has 3 predictors for X, so according to the equations we saw previously, the posterior probability P(Yes | X) would be as following:

P(Yes | X) = P(Red | Yes) \* P(SUV | Yes) \* P(Domestic | Yes) \* P(Yes)

= ⅗ x ⅕ x ⅖ x 1

= 0.048

P(No | X) would be:

P(No | X) = P(Red | No) \* P(SUV | No) \* P(Domestic | No) \* P(No)

= ⅖ x ⅗ x ⅗ x 1

= 0.144

So, as the posterior probability P(No | X) is higher than the posterior probability P(Yes | X), our Red Domestic SUV will have ‘No’ in the ‘Was it stolen?’ section.

Types of Naïve Bayes classifiers

There isn’t just one type of Naïve Bayes classifier. The most popular types differ based on the distributions of the feature values. Some of these include:

* **Gaussian Naïve Bayes (GaussianNB)**: This is a variant of the Naïve Bayes classifier, which is used with Gaussian distributions—i.e. normal distributions—and continuous variables. This model is fitted by finding the mean and standard deviation of each class.
* **Multinomial Naïve Bayes (MultinomialNB)**: This type of Naïve Bayes classifier assumes that the features are from multinomial distributions. This variant is useful when using discrete data, such as frequency counts, and it is typically applied within natural language processing use cases, like spam classification.
* **Bernoulli Naïve Bayes (BernoulliNB)**: This is another variant of the Naïve Bayes classifier, which is used with Boolean variables—that is, variables with two values, such as True and False or 1 and 0.

**Advantages**

* **Less complex**: Compared to other classifiers, Naïve Bayes is considered a simpler classifier since the parameters are easier to estimate. As a result, it’s one of the first algorithms learned within data science and machine learning courses.
* **Scales well**: Compared to logistic regression, Naïve Bayes is considered a fast and efficient classifier that is fairly accurate when the conditional independence assumption holds. It also has low storage requirements.
* **Can handle high-dimensional data**: Use cases, such document classification, can have a high number of dimensions, which can be difficult for other classifiers to manage.

**Disadvantages:**

* **Subject to Zero frequency**: Zero frequency occurs when a categorical variable does not exist within the training set. For example, imagine that we’re trying to find the maximum likelihood estimator for the word, “sir” given class “spam”, but the word, “sir” doesn’t exist in the training data. The probability in this case would zero, and since this classifier multiplies all the conditional probabilities together, this also means that posterior probability will be zero. To avoid this issue, laplace smoothing can be leveraged.
* **Unrealistic core assumption**: While the conditional independence assumption overall performs well, the assumption does not always hold, leading to incorrect classifications.

Applications of the Naïve Bayes classifier

Along with a number of other algorithms, Naïve Bayes belongs to a family of data mining algorithms which turn large volumes of data into useful information. Some applications of Naïve Bayes include:

* **Spam filtering**: Spam classification is one of the most popular applications of Naïve Bayes cited in literature.
* **Document classification**: Document and text classification go hand in hand. Another popular use case of Naïve Bayes is content classification. Imagine the content categories of a News media website.
* **Sentiment analysis**: While this is another form of text classification, sentiment analysis is commonly leveraged within marketing to better understand and quantify opinions and attitudes around specific products and brands.
* **Mental state predictions**: Using fMRI data, naïve bayes has been leveraged to predict different cognitive states among humans.