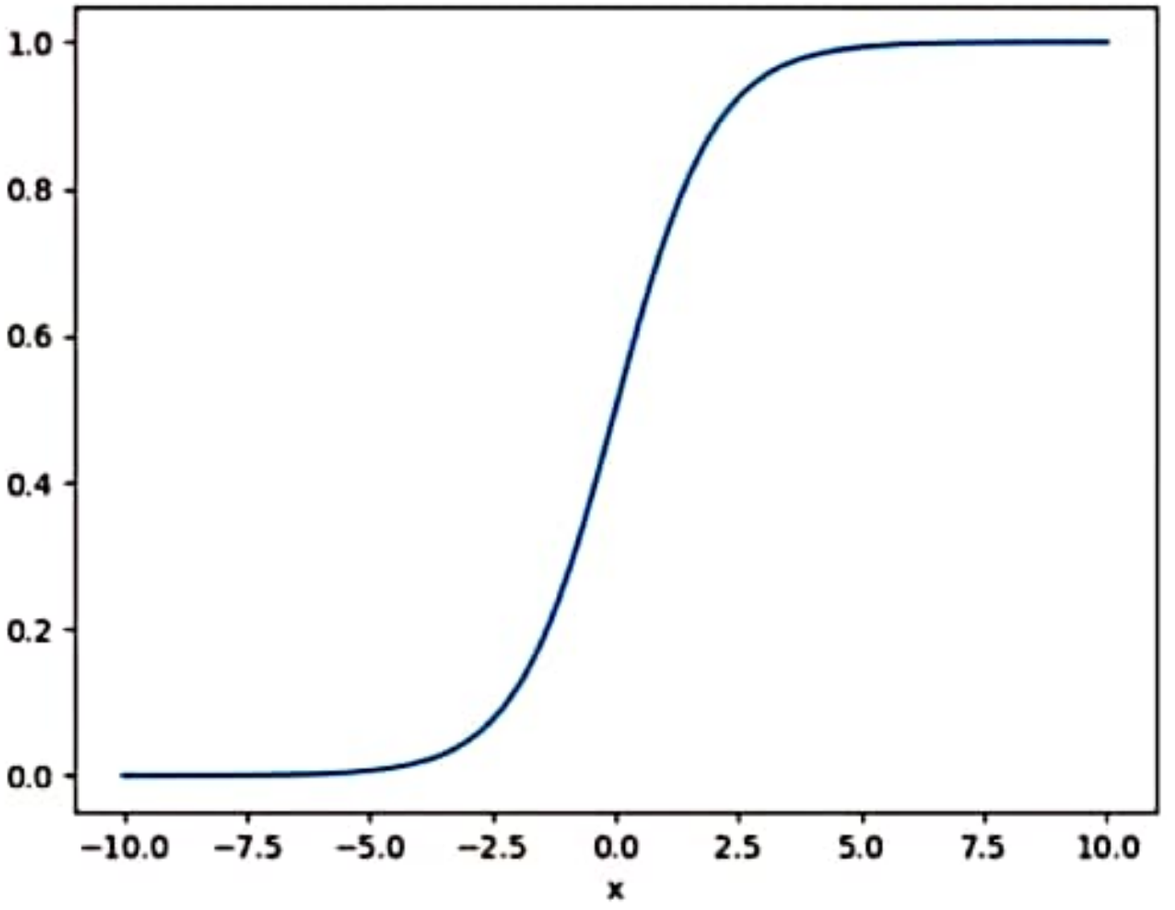
What is logistic regression?

Logistic regression is a common machine learning algorithm for binary classification and predicts a binary categorical variable through a logistic function.

The logistic regression model passes the outcome of a linear function of features through a logistic function to calculate the probability of an occurrence. The model then maps the probability to binary outcomes.

What is a sigmoid function?



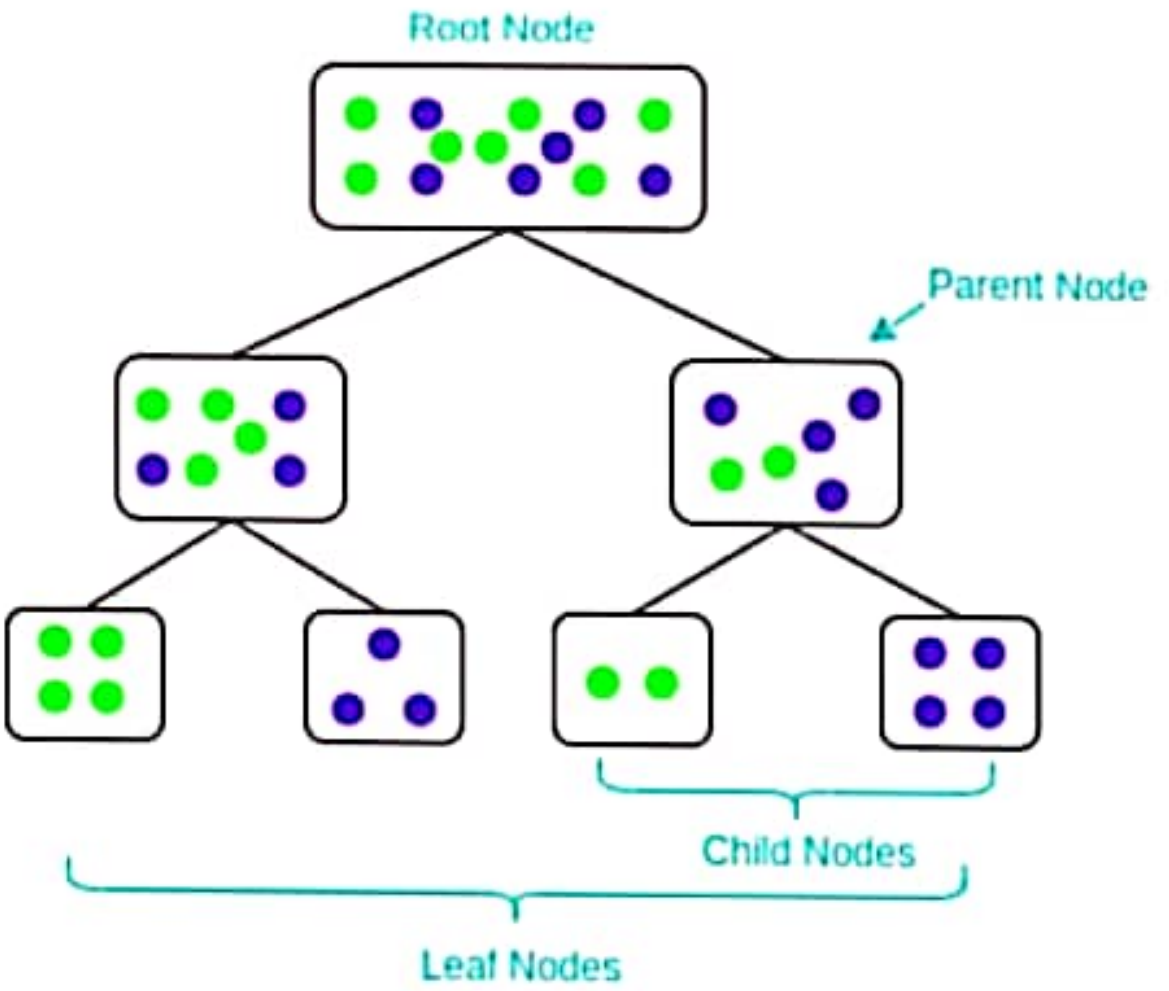
The sigmoid function forms an S shaped graph, which means as x approaches infinity, the probability becomes 1, and as x approaches negative infinity, the probability becomes 0. The model sets a threshold that decides what range of probability is mapped to which binary variable.

Suppose we have two possible outcomes, *true* and *false*, and have set the threshold as *0.5*. A probabilit[illegible]s than *0.5* would be mapped to the

4 Simple Ways to Split a Decision Tree in Machine Learning (Updated 2024)

Basic Decision Tree Terminologies

Let's quickly go through some of the key terminologies related to Split a decision tree which we'll be using throughout this article.



- **Parent and Child Node:** A node that gets divided into sub-nodes is known as Parent Node, and these sub-nodes are known as Child Nodes. Since a node can be divided into multiple sub-nodes, it can act as a parent node of numerous child nodes

What Is Node Splitting in a Decision Tree and Why Is It Done?

Modern-day programming libraries have made using any machine learning algorithm easy, but this comes at the cost of hidden implementation, which is a must-know for fully understanding an algorithm. Another reason for this infinite struggle is the availability of multiple ways to split decision tree nodes adding to further confusion.

Before learning any topic, I believe it is essential to understand why you're learning it. That helps in understanding the goal of learning a concept. So let's understand why to learn about node splitting in decision trees.

Since you all know how extensively decision trees are used, there is no

What Is Node Splitting in a Decision Tree and Why Is It Done?

Modern-day programming libraries have made using any machine learning algorithm easy, but this comes at the cost of hidden implementation, which is a must-know for fully understanding an algorithm. Another reason for this infinite struggle is the availability of multiple ways to split decision tree nodes adding to further confusion.

Before learning any topic, I believe it is essential to understand why you're learning it. That helps in understanding the goal of learning a concept. So let's understand why to learn about node splitting in decision trees.

Since you all know how extensively decision trees are used, there is no

We'll look at each splitting method in detail in the upcoming sections. Let's start with the first method of splitting – reduction in variance.

Reduction in Variance in Decision Tree

Reduction in Variance is a method for splitting the node used when the target variable is continuous, i.e., regression problems. It is called so because it uses variance as a measure for deciding the feature on which a node is split into child nodes.

$$Variance = \frac{\sum (X - \mu)^2}{N}$$

Variance is used for calculating the homogeneity of a node. If a node is entirely homogeneous, then the variance is zero.

CART — Classification and Regression Trees

Tree analogy is generally represented by CART known as Classification And Regression Tree. CART is simple to understand, interpret, visualize and requires little effort for data preparation. Moreover, it performs feature selection. Regression trees are mainly used when the target variable is numerical. Here value obtained by a terminal node is always the mean or average of the responses falling in that region. As a result, if any unseen data or observation will predict with the mean value. Classification is used when the target variable is categorical. Here value obtained by a terminal node is the mode of response falling in that region and any unseen data or observation in this region will make a prediction based on the mode value.

Even though CART is simple and has great advantages, but it can lead to

Entropy is a measure of disorder or impurity in the given dataset.

In the decision tree, messy data are split based on values of the feature vector associated with each data point. With each split, the data becomes more homogenous which will decrease the entropy. However, some data in some nodes will not be homogenous, where the entropy value will not be small. The higher the entropy, the harder it is to draw any conclusion. When the tree finally reaches the terminal or leaf node maximum purity is added.

For a dataset that has C classes and the probability of randomly choosing data from class, i is Pi. Then entropy E(S) can be mathematically represented as

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

If we have a dataset of 10 observations

is approximately equal to 0.971

$$E(S_{Red}) = -\left(\frac{3}{6} * \log_2 \frac{3}{6} + \frac{3}{6} * \log_2 \frac{3}{6}\right) = 1$$

$$E(S_{Yellow}) = -\left(\frac{3}{4} * \log_2 \frac{3}{4} + \frac{1}{4} * \log_2 \frac{1}{4}\right) \approx 0.811$$

$$\text{Weighted average} = \frac{6}{10} * E(S_{Red}) + \frac{4}{10} * E(S_{Yellow})$$

$$= \frac{6}{10} * 1 + \frac{4}{10} * 0.811$$

$$= 0.924$$

$$\text{Information Gain (S, Color)} = E(S) - \text{Weighted Average}$$

$$= 0.971 - 0.924 \approx -0.047$$

For a dataset having many features, the information gain of each feature is calculated. The feature having maximum information gain will be the

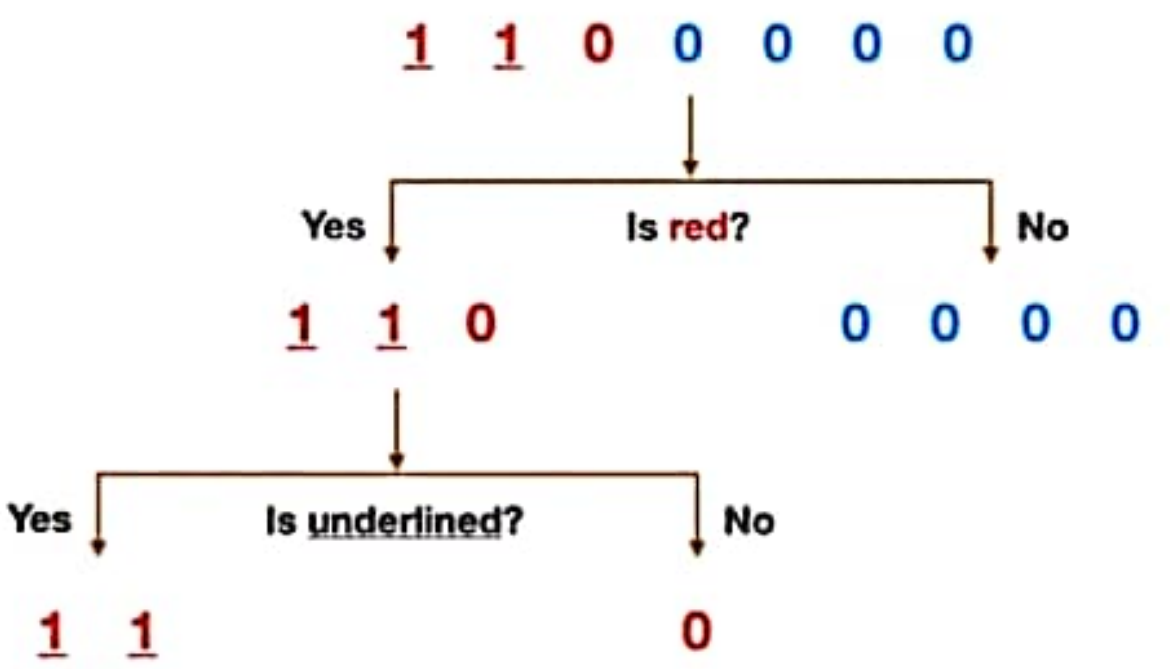
Pruning

When the tree is fully grown up, it is liking to overfit data due to noise or outliers which can lead to anomalies in decision trees. Which in turn leads to poor accuracy. This can be handled by using pruning.

Pruning is the process of removing redundant comparisons or removing subtrees. Pruning reduces unnecessary comparisons and achieves better performance. Pruned trees are less complex, smaller, and easy to understand. There are two approaches for pruning, the pre-pruning approach in which splitting or partition of the tree is halted at a particular node whereas in post-pruning approach

Decision Trees

Let's quickly go over decision trees as they are the building blocks of the random forest model. Fortunately, they are pretty intuitive. I'd be willing to bet that most people have used a decision tree, knowingly or not, at some point in their lives.



Simple Decision Tree Example

It's probably much easier to understand how a decision tree works through an example.

Imagine that our dataset consists of the numbers at the top of the figure to the left. We have two 1s and five 0s (1s and 0s are our classes) and desire to separate the classes using their

Data science provides a plethora of classification algorithms such as logistic regression, support vector machine, naive Bayes classifier, and decision trees. But near the top of the classifier hierarchy is the random forest classifier (there is also the random forest regressor but that is a topic for another day).

In this post, we will examine how basic decision trees work, how individual decisions trees are combined to make a random forest, and ultimately discover why random forests are so good at what they do.

Decision Trees

Let's quickly go over decision trees as they are the building blocks of the random forest model. Fortunately, they are pretty intuitive. I'd be willing to bet that most people have used a decision tree, knowingly or not, at some point in their lives.

It's probably much easier to understand how a decision tree works through an example.

Imagine that our dataset consists of the numbers at the top of the figure to the left. We have two 1s and five 0s (1s and 0s are our classes) and desire to separate the classes using their features. The features are color (red vs. blue) and whether the observation is underlined or not. So how can we do this?

Color seems like a pretty obvious feature to split by as all but one of the 0s are blue. So we can use the question, "Is it red?" to split our first node. You can think of a node in a tree as the point where the path splits into two — observations that meet the criteria go down the Yes branch and ones that don't go down the No branch.

The No branch (the blues) is all 0s now so we are done there, but our Yes

Understanding Random Forest

How the Algorithm Works and Why it Is So Effective

Tony Yiu · Follow

Published in Towards Data Science

9 min read · Jun 12, 2019

Listen Share

A big part of machine learning is classification — we want to know what class (a.k.a. group) an observation belongs to. The ability to precisely classify observations is extremely valuable for various business applications like predicting whether a particular user will buy a product or forecasting whether a given loan will default or not.

Data science provides a plethora of classification algorithms such as logistic regression, support vector machine, naive Bayes classifier, and decision trees. But near the top of the

Pruning

When the tree is fully grown up, it is liking to overfit data due to noise or outliers which can lead to anomalies in decision trees. Which in turn leads to poor accuracy. This can be handled by using pruning.

Pruning is the process of removing redundant comparisons or removing subtrees. Pruning reduces unnecessary comparisons and achieves better performance. Pruned trees are less complex, smaller, and easy to understand. There are two approaches for pruning, the pre-pruning approach in which splitting or partition of the tree is halted at a particular node whereas in post-pruning approach removes subtree from the full tree. A

and replacing it with a leaf node.

Implementation of Decision Tree Classifier using Python

For the Decision Tree classification algorithm, the feature values need to be categorical. The most important feature forms the root node and for each feature in the dataset, the algorithm forms a node. The leaf node contains the decision or outcome of the decision tree. If the entropy of a node is zero it is called a pure node.

Here we are using a dataset that is used to analyze if the mushroom is poisonous or edible. For data analysis, libraries and datasets are to be imported.

```
#Importing libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
%matplotlib inline
```

Gini index

The Gini index can also be used for feature selection. The tree chooses the feature that minimizes the Gini impurity index. The higher value of the Gini Index indicates the impurity is higher. Both Gini Index and Gini Impurity are used interchangeably. The Gini Index or Gini Impurity favors large partitions and is very simple to implement. It performs only binary split. For categorical variables, it gives the results in terms of "success" or "failure".

Gini Index can be calculated from the below mathematical formula where c is the number of classes and pi is the probability associated with the ith class.

$$Gini = 1 - \sum_{i=1}^{C} (p_i)^2$$

Information Gain is the variance reduction and can calculate by how much the variance decreases after each split.

Information gain of a parent node can be calculated as the entropy of the parent node subtracted entropy of the weighted average of the child node.

As per the above example, the dataset has 10 observations belonging to two classes YES and NO. Where 6 observations belong to the class, YES, and 4 observations belong to class NO.

Observations	COLOR	Outcome
1	Red	Yes
2	Red	No
3	Yellow	Yes
4	Yellow	Yes
5	Red	Yes
6	Yellow	Yes
7	Red	No
8	Red	No
9	Red	Yes
10	Yellow	No

Red color has 3 Yes outcome and 3 No outcome whereas yellow has 3 Yes

If we have a dataset of 10 observations belonging to two classes YES and NO. If 6 observations belong to the class, YES, and 4 observations belong to class NO, then entropy can be written as below.

$$E(S) = -(P_{yes}\log_2 P_{yes} + P_{no}\log_2 P_{no})$$

Pyes is the probability of choosing Yes and *Pno* is the probability of choosing a No. Here *Pyes* is 6/10 and *Pno* is 4/10.

$$E(S) = -(6/10 * \log_2 * 6/10 + 4/10 * \log_2 * 4/10) \approx 0.971$$

If all the 10 observations belong to 1 class then entropy will be equal to zero. Which implies the node is a pure node.

$$E(S) = -(1 \log_2 1) = 0$$

If both classes YES and NO have an

$$E(S) = -(1 \log_2 1) = 0$$

If both classes YES and NO have an equal number of observations, then entropy will be equal to 1.

$$E = -\left(\frac{5}{10} * \log_2 \frac{5}{10} + \frac{5}{10} * \log_2 \frac{5}{10}\right) = -2(0.5 \log_2 0.5) = 1$$

Information Gain

The Information Gain measures the expected reduction in entropy. Entropy measures impurity in the data and information gain measures reduction in impurity in the data. The feature which has minimum impurity will be considered as the root node.

Information gain is used to decide which feature to split on at each step in building the tree. The creation of

Gini Index can be used for choosing the best split. Entropy and Information gain go hand in hand.

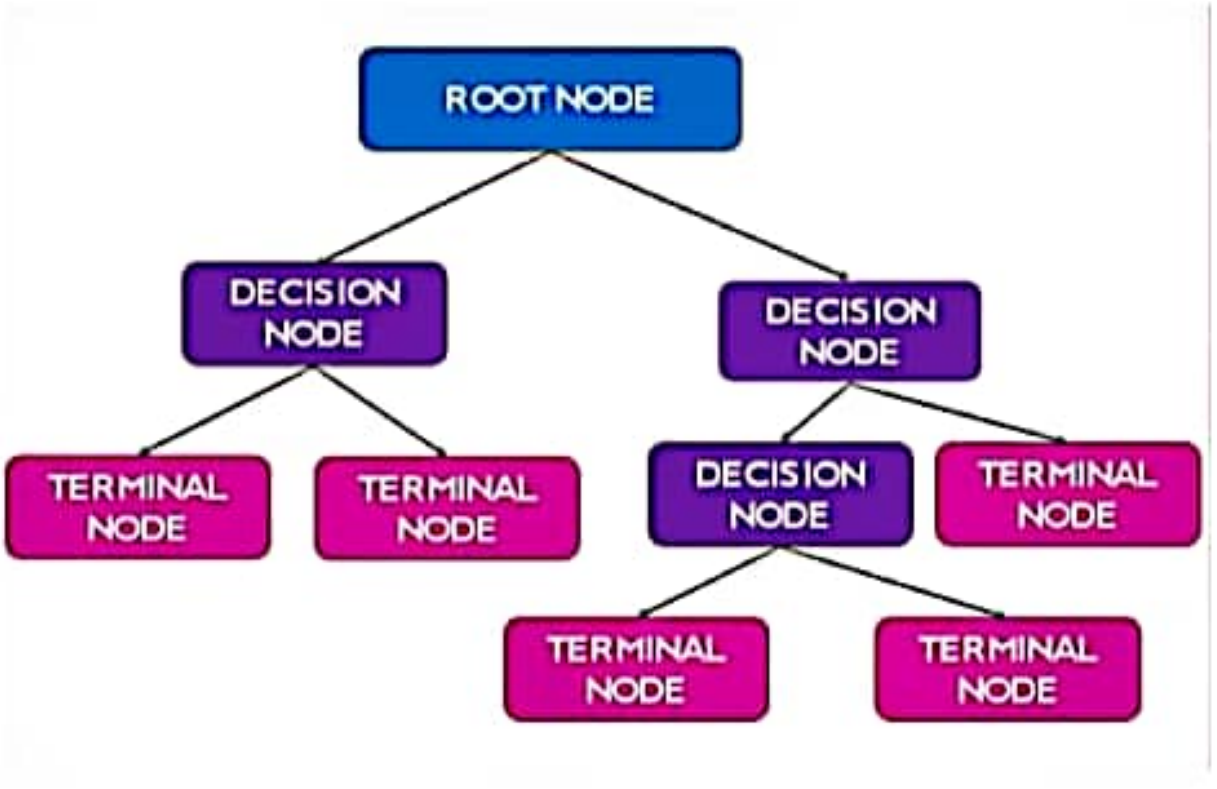
For a given dataset with different features, to decide which feature to be considered as the root node and which feature should be the next decision node and so on, information gain of each feature should be known. The feature which has maximum information gain will be considered as the root node. To calculate information gain first we should calculate the entropy.

Entropy :

Entropy is a measure of disorder or impurity in the given dataset.

In the decision tree, messy data are split based on values of the feature vector associated with each data point. With each split, the data becomes more homogenous which will decrease the entropy. However, some data in some nodes will not be homogenous,

represents an outcome of a test and each terminal node or leaf holds a class. It has one incoming edge and has two or more outgoing edges. Terminal node or Leaf node represents a class node and has exactly one incoming node and no outgoing node.



Each node in the decision tree is a condition on the feature. Which is designed to split the dataset into similar response values ends up in the same dataset.

CART — Classification and Regression Trees

Tree analogy is generally represented by CART known as Classification And Regression Tree. CART is simple to

prediction based on the mode value.

Even though CART is simple and has great advantages, but it can lead to overfitting if data is not properly handled. Moreover, it can lead to instability, if there is a small variation in data.

While growing a tree below points are to be considered :

1. Features to choose

2. Conditions for splitting

3. To know where to stop

4. Pruning

The decision to make a strategic split heavily affects the accuracy of the tree and the decision criteria for regression and classification trees will be different. Entropy/Information gain or Gini Index can be used for choosing the best split. Entropy and Information gain go hand in hand.

Here are the steps to split a decision tree using the reduction in variance method:

1. For each split, individually calculate the variance of each child node
2. Calculate the variance of each split as the weighted average variance of child nodes
3. Select the split with the lowest variance
4. Perform steps 1-3 until completely homogeneous nodes are achieved

The below video excellently explains the reduction in variance using an example:

training process until only homogenous nodes are left. This is why a decision tree performs so well. The process of recursive node splitting into subsets created by each sub-tree can cause overfitting. Therefore, node splitting is a key concept that everyone should know.

Node splitting, or simply splitting, divides a node into multiple sub-nodes to create relatively pure nodes. This is done by finding the best split for a node and can be done in multiple ways. The ways of splitting a node can be broadly divided into two categories based on the type of target variable:

1. **Continuous Target Variable:** Reduction in Variance
2. **Categorical Target Variable:** Gini Impurity, Information Gain, and

- **Root Node:** The topmost node of a decision tree. It does not have any parent node. It represents the entire population or sample.
- **Leaf / Terminal Nodes:** Nodes of the tree that do not have any child node are known as Terminal/Leaf Nodes.

There are multiple tree models to choose from based on their learning technique when building a decision tree, e.g., ID3, CART, Classification and Regression Tree, C4.5, etc. Selecting which decision tree to use is based on the problem statement. For example, for classification problems, we mainly use a classification tree with a gini index to identify class labels for datasets with relatively more number of classes.

In this article, we will mainly discuss the CART tree.

What is a sigmoid function?

The logistic function in linear regression is a type of sigmoid, a class of functions with the same specific properties.

Sigmoid is a mathematical function that takes any real number and maps it to a probability between 1 and 0.

The formula of the sigmoid function is:

$$f(x) = \frac{1}{1 + e^{-x}}$$

outcome *false*, and a probability greater than or equal to *0.5* would be mapped to the outcome *true*.

Example

Suppose, a regression model is fit using some training data to obtain β and x represents the input features:

$$z = \beta^t x$$
$$\beta^t = [15\ 1]$$
$$x^t = [0.1\ -0.5]$$

The probability of z being mapped to 1 is given by the equation:

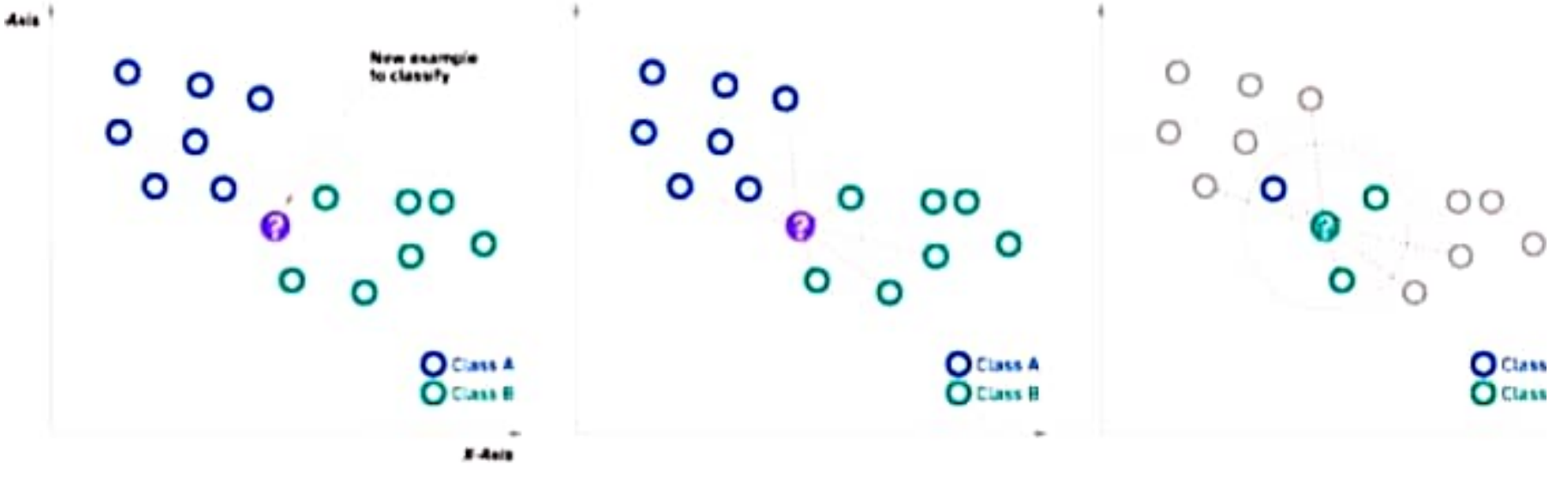
$$\sigma(z) = \sigma(0.1 * 15 + (-0.5*1)) = \sigma(1)$$
$$= 1/(1 + e^{-1}) \approx 0.7311$$

What is the KNN algorithm?

The k-nearest neighbors (KNN) algorithm is a non-parametric, supervised learning classifier, which uses proximity to make classifications or predictions about the grouping of an individual data point. It is one of the popular and simplest classification and regression classifiers used in machine learning today.

While the KNN algorithm can be used for either regression or classification problems, it is typically used as a classification algorithm, working off the assumption that similar points can be found near one another.

For classification problems, a class label is assigned on the basis of a majority vote—i.e. the label that is most frequently represented around a given data point is used. While this is technically considered "plurality voting", the term, "majority vote" is more commonly used in literature. The distinction between these terminologies is that 'majority voting" technically requires a majority of greater than 50%, which primarily works when there are only two categories. When you have multiple classes—e.g. four categories, you don't necessarily need 50% of the vote to make a conclusion about a class; you could assign a class label with a vote of greater than 25%. The University of Wisconsin-Madison summarizes this well with an example here (link resides outside of ibm.com).



Compute KNN: distance metrics

To recap, the goal of the k-nearest neighbor algorithm is to identify the nearest neighbors of a given query point, so that we can assign a class label to that point. In order to do this, KNN has a few requirements:

Determine your distance metrics

In order to determine which data points are closest to a given query point, the distance between the query point and the other data point will need to be calculated. These distance metrics help to form decision boundaries, which partition query points into different regions. You commonly will see decision boundaries visualized with Voronoi diagrams.

While there are several distance measures that you can choose from, this article will only cover the following:

Euclidean distance (p=2): This is the most commonly used distance measure, and it is limited to real-valued vectors. Using the below formula, it measures a straight line between

As an example, if you had the following strings, the hamming distance would be 2 since only two of the values differ.

Vector 1	1	0	1	0	0	0	1	1
Vector 2	1	0	0	0	0	0	0	1

Hamming distance example

Compute KNN: defining k

The k value in the k-NN algorithm defines how many neighbors will be checked to determine the classification of a specific query point. For example, if k=1, the instance will be assigned to the same class as its single nearest neighbor. Defining k can be a balancing act as different

Hamming distance: This technique is used typically used with Boolean or string vectors, identifying the points where the vectors do not match. As a result, it has also been referred to as the overlap metric. This can be represented with the following formula:

$$\text{Hamming Distance} = D_H = \left(\sum_{i=1}^{k} |x_i - y_i| \right)$$

$$x = y \qquad D = 0$$

$$x \neq y \qquad D \neq 1$$

$$\text{Manhattan Distance} = d(x,y) = \left(\sum_{i=1}^{m} \left|x_i - y_i\right|\right)$$

Manhattan distance formula

Minkowski distance: This distance measure is the generalized form of Euclidean and Manhattan distance metrics. The parameter, p, in the formula below, allows for the creation of other distance metrics. Euclidean distance is represented by this formula when p is equal to two, and Manhattan distance is denoted with p equal to one.

Such as Natural Language Processing.

To understand the naive Bayes classifier we need to understand the Bayes theorem. So let's first discuss the Bayes Theorem.

> How Naive Bayes classifier algorithm works in machine learning
>
> CLICK TO TWEET 

What is Bayes Theorem?

Bayes theorem named after Rev. Thomas Bayes. It works on conditional **probability**. Conditional probability is the probability that something will happen, ***given that something else*** **has already occurred**. Using the conditional probability, we can calculate the probability of an event using its prior knowledge.

Below is the formula for calculating the conditional probability.

$latex \textrm{P(H \textbar E) = }
\frac{\textrm{ P(E \textbar H) * P(H)}}
{\textrm{P(E)}}$

where

- P(H) is the probability of hypothesis H being true. This is known as the prior probability.
- P(E) is the probability of the evidence(regardless of the hypothesis).
- P(E|H) is the probability of the evidence given that hypothesis is true.
- P(H|E) is the probability of the hypothesis given that the evidence is there.

Let's consider an example to understand how the above formula of Bayes theorem works.

Probability of people not suffering from Disease D, P(~D) = 0.97 = 97%

Probability that test gives "positive" result and patient does have the disease, P(Pos | ~D) = 0.01 =1%

For calculating the probability that the patient actually have the disease i.e, **P(D | Pos)** we will use Bayes theorem:

$latex \textrm{P(D \textbar Pos) = }
\frac{\textrm{ P(Pos \textbar D) * P(D)}}
{\textrm{P(Pos)}}$

We have all the values of numerator but we need to calculate P(Pos):

P(Pos) = P(D, pos) + P(~D, pos)

= P(pos | D)*P(D) + P(pos | ~D)*P(~D)

= 0.99 * 0.03 + 0.01 * 0.97

= 0.0297 + 0.0097

= 0.0394

Let's calculate, **P(D | Pos) = (P(Pos | D) ***

Problem:

A Path Lab is performing a Test of disease say "D" with two results "Positive" & "Negative." They guarantee that their test result is 99% accurate: if you have the disease, they will give test positive 99% of the time. If you don't have the disease, they will test negative 99% of the time. If 3% of all the people have this disease and test gives "positive" result, **what is the probability that you actually have the disease?**

For solving the above problem, we will have to use conditional probability.

Probability of people suffering from Disease D, $P(D) = 0.03 = 3\%$

Probability that test gives "positive" result and patient have the disease, $P(Pos \mid D) = 0.99 = 99\%$

Probability of people not suffering from Disease D, $P(\sim D) = 0.97 = 97\%$

Probability that test gives "positive" result and patient does have the disease, P(Pos |

data point belongs to a particular class. The class with the highest probability is considered as the most likely class. This is also known as **Maximum A Posteriori (MAP).**

The MAP for a hypothesis is:

MAP(H)

$= \max(P(H|E))$

$= \max((P(E|H)*P(H))/P(E))$

$= \max(P(E|H)*P(H))$

P(E) is evidence probability, and it is used to normalize the result. It remains same so, removing it won't affect.

Naive Bayes classifier assumes that all the features are **unrelated** to each other. Presence or absence of a feature does not influence the presence or absence of any other feature. We can use Wikipedia example for explaining the logic i.e.,

> A fruit may be considered to be an apple if it is red, round, and about 4" in diameter. Even if

We have 3 classes associated with Animal Types:

- Parrot,
- Dog,
- Fish.

The Predictor features set consists of 4 features as

- Swim
- Wings
- Green Color
- Dangerous Teeth.

Green Color, Dangerous Teeth. All the features are categorical variables with either of the 2 values: **T(True) or F(False)**.

Swim	Wings	Green Color	Dangero Teeth
50	500/500	400/500	0
450/500	0	0	500/500
500/500	0	100/500	50/500

of Green color and 500 out of 500(100%) dogs have Dangerous Teeth.

- Classes with Animal type Fishes shows that 500 out of 500(100%) can swim, 0(0%) fishes have wings, 100(20%) fishes are of Green color and 50 out of 500(10%) dogs have Dangerous Teeth.

Now, it's time to work on predict classes using the Naive Bayes model. We have taken 2 records that have values in their feature set, but the target variable needs to predicted.

	Swim	Wings
1.	True	False
2.	True	False

We have to predict animal type using the feature values. We have to predict whether the animal is a Dog, a Parrot or a Fish

We will use the Naive Bayes approach

P(Dog | Swim, Green, Teeth) =

P(Swim|Dog) * P(Green|Dog) *

P(Teeth|Dog) * P(Dog) / P(Swim, Green, Teeth)

= 0.9 * 0 * 1 * 0.333 / P(Swim, Green, Teeth)

= 0

For Hypothesis testing for the animal to be a Parrot:

P(Parrot| Swim, Green, Teeth) =

P(Swim|Parrot) * P(Green|Parrot)*

P(Teeth|Parrot) * P(Parrot) / P(Swim, Green, Teeth)

= 0.1 * 0.80 * 0 *0.333 / P(Swim, Green, Teeth)

= 0

For Hypothesis testing for the animal to be a Fish:

P(Fish|Swim, Green, Teeth) = P(Swim|Fish) * P(Green|Fish) * P(Teeth|Fish) *P(Fish) / P(Swim, Green, Teeth)

= 1 * 0.2 * 0.1 * 0.333 / P(Swim, Green,

Disadvantages

- It considers all the features to be unrelated, so it cannot learn the relationship between features. E.g., Let's say Remo is going to a part. While cloth selection for the party, Remo is looking at his cupboard. Remo likes to wear a white color shirt. In Jeans, he likes to wear a brown Jeans, But Remo doesn't like wearing a white shirt with Brown Jeans. Naive Bayes can learn individual features importance but can't determine the relationship among features.

Gaussian Naive Bayes

When attribute values are continuous, an assumption is made that the values associated with each class are distributed according to Gaussian i.e., Normal Distribution.

If in our data, an attribute say “x” contains continuous data. We first segment the data by the class and then compute mean $latex \mu_{y} $ & Variance $latex {\sigma_{y}}^{2} $ of each class.

MultiNomial Naive Bayes

MultiNomial Naive Bayes is preferred to use on data that is multinomially distributed. It is one of the standard classic algorithms. Which is used in text

Advantages and Disadvantage of Naive Bayes classifier

Advantages

- Naive Bayes Algorithm is a fast, highly scalable algorithm.
- Naive Bayes can be use for Binary and Multiclass classification. It provides different types of Naive Bayes Algorithms like GaussianNB, MultinomialNB, BernoulliNB.
- It is a simple algorithm that depends on doing a bunch of counts.
- Great choice for Text Classification problems. It's a popular choice for spam email classification.
- It can be easily train on small dataset

Disadvantages

- It considers all the features to be unrelated, so it cannot learn the relationship between features. E.g., Let's say Remo is going to a part.

MultiNomial Naive Bayes

MultiNomial Naive Bayes is preferred to use on data that is multinomially distributed. It is one of the standard classic algorithms. Which is used in text categorization (classification). Each event in text classification represents the occurrence of a word in a document.

Bernoulli Naive Bayes

Bernoulli Naive Bayes is used on the data that is distributed according to multivariate Bernoulli distributions.i.e. multiple features can be there, but each one is assumed to be a binary-valued (Bernoulli, boolean) variable. So, it requires features to be binary valued.

Advantages and Disadvantage of Naive Bayes classifier

* P(Green | Fish) * P(Teeth | Fish) *P(Fish) / P(Swim, Green, Teeth)

= 1 * 0.2 * 0.1 * 0.333 / P(Swim, Green, Teeth)

= 0.00666 / P(Swim, Green, Teeth)

The denominator of all the above calculations is same i.e, P(Swim, Green, Teeth). The value of P(Fish | Swim, Green, Teeth) is the only positive value greater than 0. Using Naive Bayes, we can predict that the class of this record is Fish.

As the calculated value of probabilities is very less. To normalize these values, we need to use denominators.

Let's proceed to learn the various type of **Naive Bayes Methods**.

Types of Naive Bayes Algorithm

Gaussian Naive Bayes

When attribute values are continuous, an

the animal is a Dog, a Parrot or a Fish

We will use the Naive Bayes approach

P(H|Multiple Evidences) = P(E1 | H)* P(E2|H) .....*P(En|H) * P(H) / P(Multiple Evidences)

Let's consider the first record.
The Evidence here is **Swim & Green**. The Hypothesis can be an animal type to be Dog, Parrot, Fish.

For Hypothesis testing for the animal to be a Dog:

P(Dog | Swim, Green) = P(Swim|Dog) * P(Green|Dog) * P(Dog) / P(Swim, Green)
= 0.9 * 0 * 0.333 / P(Swim, Green)
= 0

For Hypothesis testing for the animal to be a Parrot:

P(Parrot| Swim, Green) = P(Swim|Parrot) * P(Green|Parrot) * P(Parrot) / P(Swim, Green)
= 0.1 * 0.80 * 0.333 / P(Swim, Green)

Green Color, Dangerous Teeth. All the features are categorical variables with either of the 2 values: **T(True) or F(False)**.

Swim	Wings	Green Color	Dangerou Teeth
50	500/500	400/500	0
450/500	0	0	500/500
500/500	0	100/500	50/500

The above table shows a frequency table of our data. In our training data:

- Parrots have 50(10%) value for Swim, i.e., 10% parrot can swim according to our data, 500 out of 500(100%) parrots have wings, 400 out of 500(80%) parrots are Green and 0(0%) parrots have Dangerous Teeth.
- Classes with Animal type Dogs shows that 450 out of 500(90%) can swim, 0(0%) dogs have wings, 0(0%) dogs are of Green color and 500 out of 500(100%) dogs have Dangerous

these features depend on each other or upon the existence of the other features, a naive Bayes classifier considers all of these properties to independently contribute to the probability that this fruit is an apple.

In real datasets, we test a hypothesis given multiple evidence(feature). So, calculations become complicated. To simplify the work, the feature independence approach is used to 'uncouple' multiple evidence and treat each as an independent one.

P(H|Multiple Evidences) = P(E1| H)* P(E2|H) .....*P(En|H) * P(H) / P(Multiple Evidences)

Example of Naive Bayes Classifier

For understanding a theoretical concept, the best procedure is to try it on an example. Since I am a pet lover so selected

Let's calculate, **P(D | Pos) = (P(Pos | D) * P(D)) / P(Pos)**

= (0.99 * 0.03) / 0.0394

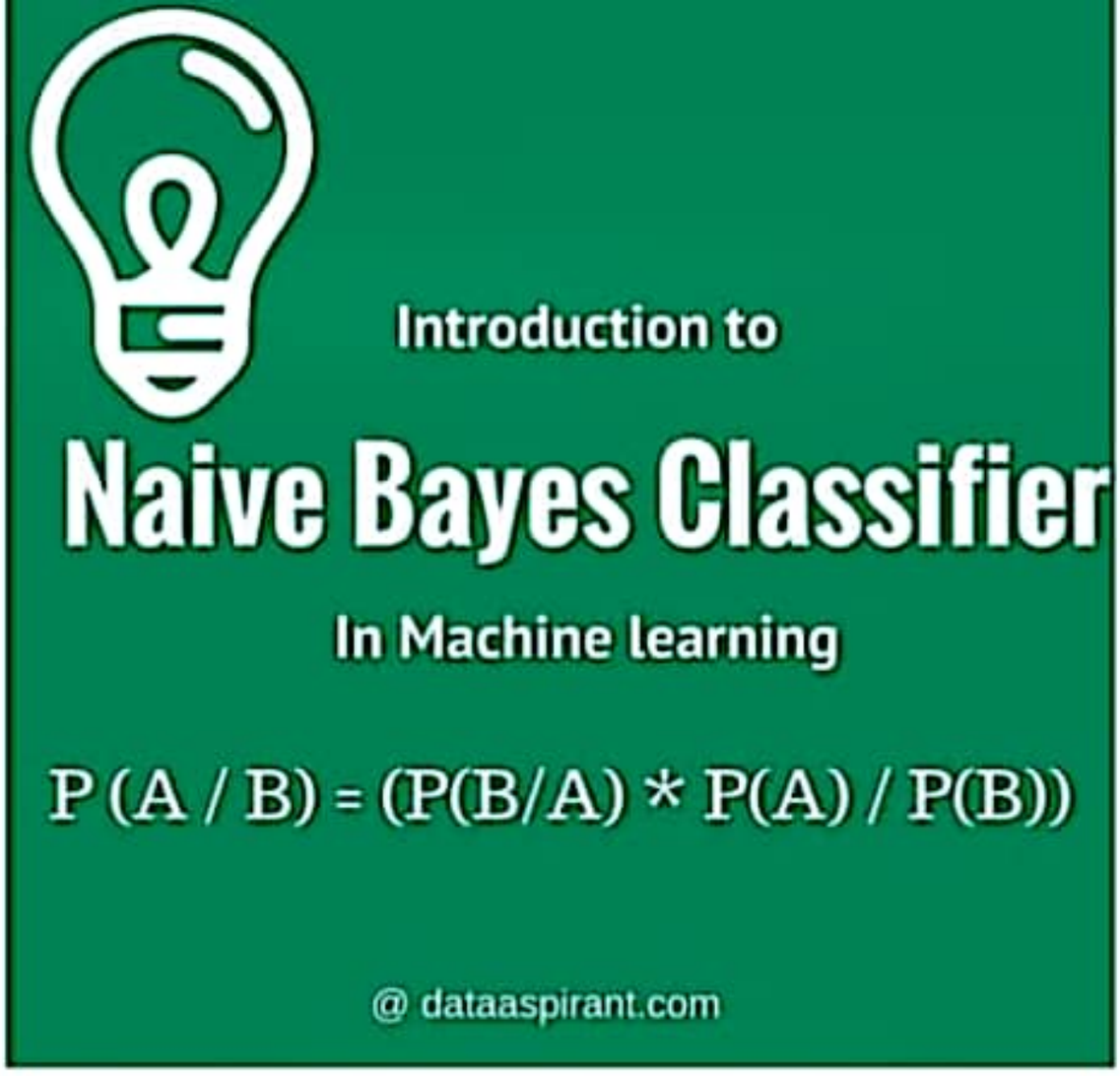
= 0.753807107

So, Approximately **75%** chances are there that the patient is actually suffering from disease.

I hope we understand the Bayes theorem. Now let's use this understanding to find out more about the naive Bayes classifier.

Naive Bayes Classifier

Naive Bayes is a kind of classifier which uses the Bayes Theorem. It predicts membership probabilities for each class such as the probability that given record or data point belongs to a particular class. The class with the highest probability is considered as the most likely class. This is also known as **Maximum A Posteriori (MAP)**.



Naive Bayes Classifier Algorithm

Naive Bayes classifier is a straightforward and powerful algorithm for the **classification** task. Even if we are working on a data set with millions of records with some attributes, it is suggested to try Naive Bayes approach.

Naive Bayes classifier gives great results when we use it for textual data analysis. Such as Natural Language Processing.

To understand the naive Bayes classifier we need to understand the Bayes

The k value in the k-NN algorithm defines how many neighbors will be checked to determine t classification of a specific query point. For example, if k=1, the instance will be assigned t the same class as its single nearest neighbor. Defining k can be a balancing act as different values can lead to overfitting or underfitting. Lower values of k can have high variance, but l bias, and larger values of k may lead to high bi and lower variance. The choice of k will largely depend on the input data as data with more outliers or noise will likely perform better with higher values of k. Overall, it is recommended have an odd number for k to avoid ties in classification, and cross-validation tactics can help you choose the optimal k for your dataset

k-nearest neighbors and python

To delve deeper, you can learn more about the NN algorithm by using Python and scikit-learn also known as sklearn). Our tutorial in Watson Studio helps you learn the basic syntax from th ibrary, which also contains other popular ibraries, like NumPy, pandas, and Matplotlib. T ollowing code is an example of how to create predict with a KNN model:

While there are several distance measures that you can choose from, this article will only cover the following:

Euclidean distance (p=2): This is the most commonly used distance measure, and it is limited to real-valued vectors. Using the below formula, it measures a straight line between the query point and the other point being measured.

$$d(x,y) = \sqrt{\sum_{i=1}^{n}(y_i - x_i)^2}$$

Euclidean distance formula

Manhattan distance (p=1): This is also another popular distance metric, which measures the absolute value between two points. It is also referred to as taxicab distance or city block

Regression problems use a similar concept as classification problem, but in this case, the average the k nearest neighbors is taken to make a prediction about a classification. The main distinction here is that classification is used for discrete values, whereas regression is used with continuous ones. However, before a classification can be made, the distance must be defined. Euclidean distance is most commonly used, which we'll delve into more below.

It's also worth noting that the KNN algorithm is also part of a family of "lazy learning" models, meaning that it only stores a training dataset versus undergoing a training stage. This also means that all the computation occurs when a classification or prediction is being made. Since it heavily relies on memory to store all its training data, it is also referred to as an instance-based or memory-based learning method.

Evelyn Fix and Joseph Hodges are credited with the initial ideas around the KNN model in this 1951 paper (link resides outside of ibm.com) while Thomas Cover expands on their concept in his research (link resides outside of ibm.com), "Nearest Neighbor Pattern Classification." While it's not as popular as it once was, it is still one of the first algorithms one learns in data science due to its simplicity and accuracy. However, as a dataset grows, KNN becomes increasingly inefficient, compromising overall

Solution
In programming, Loops are used to repeat a block of code until a specific condition is met. The While loop loops through a block of code as long as a specified condition is true.

To Learn more about working of While Loops read:

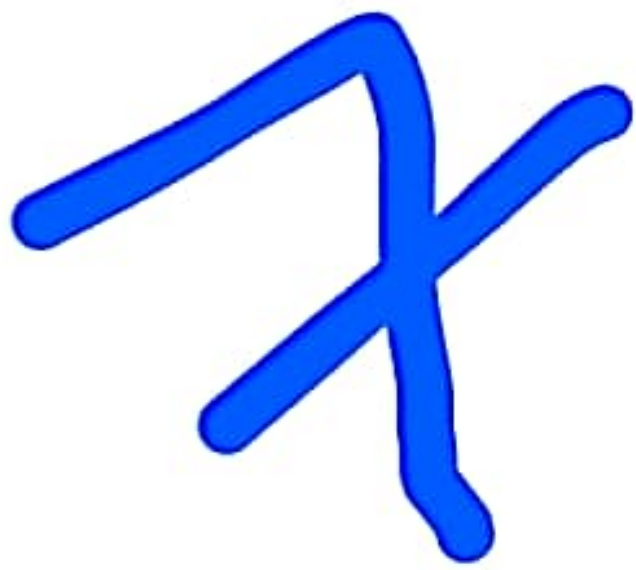
How To Construct While Loops In Python
Creating A Guessing Game In Python Program

```
i = 1
while(i<=10):
    print(i)    i += 1
```

Copy

Output

```
1
2
3
4
5
6
7
8
9
10
```



6:39 pm ✓

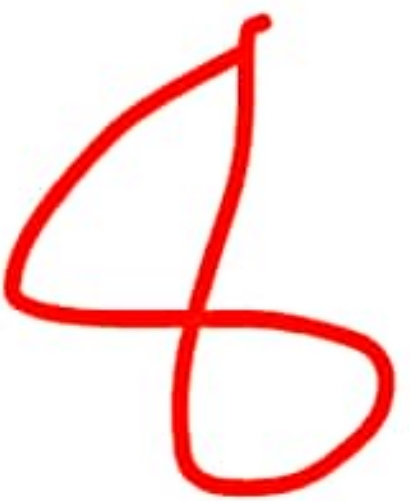
Factorial is a product of all positive numbers from `1 to n`, here `n` is a number to find factorial.

Ex: 5! = 5*4*3*2*1

You can also check factorial of a program using for loop, factorial of a program using Recursion, Flowchart to Find Factorial of a Number and Factorial of a number using Functions in C.

```c
#include <stdio.h>
int main()
{
    int n,i,f;
    f=i=1;
    printf("Enter a Number to Find
Factorial: ");
    scanf("%d",&n);
    while(i<=n)
    {
        f*=i;
        i++;
    }
    printf("The Factorial of %d is :
%d",n,f);
    return 0;
}
```

```c
#include <stdio.h>
int main()
{
    int n,i,f;
    f=i=1;
    printf("Enter a Number to Find
Factorial: ");
    scanf("%d",&n);
    while(i<=n)
    {
        f*=i;
        i++;
    }
    printf("The Factorial of %d is :
%d",n,f);
    return 0;
}
```



5:34 pm ✓

```
Enter a Number to Find Factorial: 5
The Factorial of 5 is : 120
```

5:35 pm ✓

Python Basic: Exercise-109 with Solution

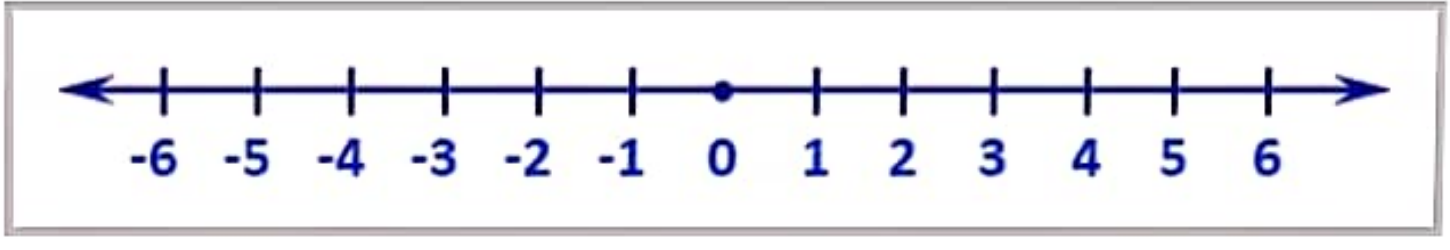
Write a Python program to check if a number is positive, negative or zero.

Positive Numbers: Any number above zero is known as a positive number. Positive numbers are written without any sign or a '+' sign in front of them and they are counted up from zero, i.e. 1, + 2, 3, +4 etc.

Negative Numbers: Any number below zero is known as a negative number. Negative numbers are always written with a '–' sign in front of them and they are counted down from zero, i.e. -1, -2, -3, -4 etc.

Always look at the sign in front of a number to check if it is positive or negative. Zero, 0, is neither positive nor

Pictorial Presentation:



resource
Python: Check if a number is positive, negative or zero
Last update on October 30 2023 13:02:04 (UTC/GMT +8 hours)

Python Basic: Exercise-109 with Solution
Write a Python program to check if a number is positive, negative or zero.

Positive Numbers: Any number above zero is known as a positive number. Positive numbers are written without any sign or a '+' sign in front of them and they are counted up from zero, i.e. 1, + 2, 3, +4 etc.
Negative Numbers: Any number below zero is known as a negative number. Negative numbers are always written with a '−' sign in front of them and they are counted down from zero, i.e. -1, -2, -3, -4 etc.
Always look at the sign in front of a number to check if it is positive or negative. Zero, 0, is neither positive nor negative.

Pictorial Presentation:

Python: Check if a number is positive, negative or zero.

resource
Python: Check if a number is positive, negative or zero
Last update on October 30 2023 13:02:04 (UTC/GMT +8 hours)

Python Basic: Exercise-109 with Solution
Write a Python program to check if a number is positive, negative or zero.

Positive Numbers: Any number above zero is known as a positive number. Positive numbers are written without any sign or a '+' sign in front of them and they are counted up from zero, i.e. 1, + 2, 3, +4 etc.
Negative Numbers: Any number below zero is known as a negative number. Negative numbers are always written with a '−' sign in front of them and they are counted down from zero, i.e. -1, -2, -3, -4 etc.
Always look at the sign in front of a number to check if it is positive or negative. Zero, 0, is neither positive nor negative.

Pictorial Presentation:

Python: Check if a number is positive, negative or zero.
Sample Solution-1:

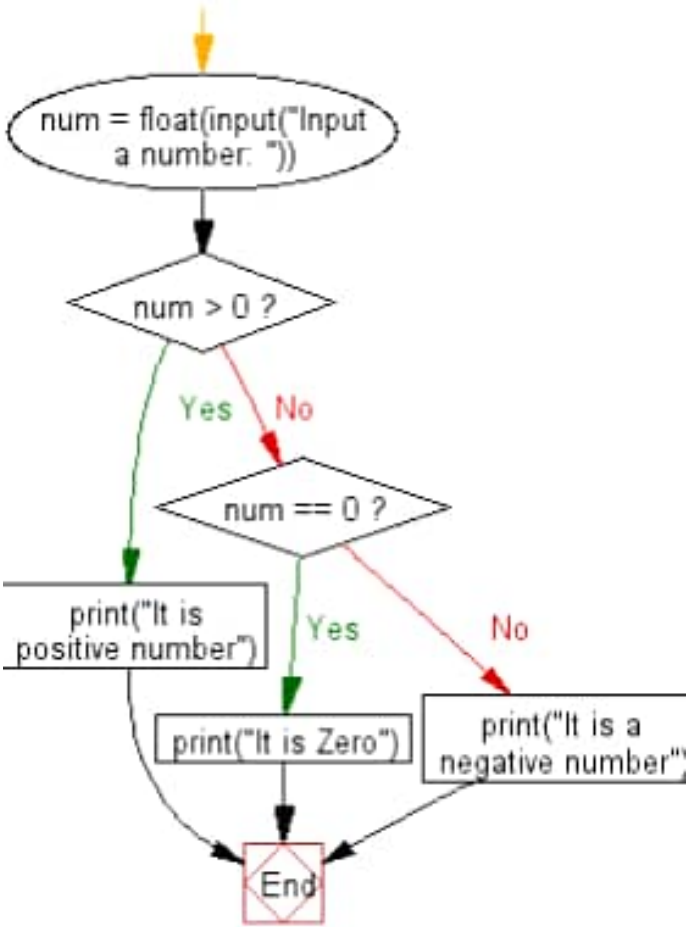
Python Code:

```
# Prompt the user to input a number, and convert the input to a floating-point number.
num = float(input("Input a number: "))

# Check if the number is greater than zero.
if num > 0:
   # If true, print that it is a positive number.
   print("It is a positive number")
# Check if the number is equal to zero.
elif num == 0:
   # If true, print that it is zero.
   print("It is zero")
else:
   # If the above conditions are not met, print that it is a negative number.
   print("It is a negative number")
```

Sample Output:

```
Input a number: 150
It is positive number
```



Find the greatest number in given three numbers

10

```
#include<stdio.h>
int main(){
    int a,b,c,big;
    printf("\nEnter 3 numbers:");
    scanf("%d %d %d",&a,&b,&c);
    big=(a>b&&a>c?a:b>c?b:c);
    printf("\nThe biggest number is:%d",big);
    return 0;
}
```

C program for largest of 3 numbers

Write a c program to find largest of three numbers

```
#include<stdio.h>
int main(){
   int a,b,c;
   int big;

      printf("Enter any there
numbers: ");
   scanf("%d%d%d",&a,&b,&c);


   if(a>b && a>c)
     big = a;
    else if(b>c)
     big = b;
    else
     big = c;
```

10

```
    printf("Largest number is:
%d",big);


   return 0;
}
```

Sample output:

```
Enter any there numbers: 13 25
6
Largest number is: 25
```

NumPy | Create array filled with all ones

To create an **array filled with all ones**, given the shape and type of array we can use **numpy.ones() method** of NumPy library in Python.

Example:

Python3

```
import numpy as np

array = np.ones(5)
print(array)
```

Output:

```
[1. 1. 1. 1. 1.]
```

Syntax: np.ones(shape, dtype=None, order='C', *, like=None)

Parameters:

- **shape :** integer or sequence of integers
- **order :**
  - C_contiguous or F_contiguous
  - C-contiguous order in memory(last index varies the fastest)
  - C order means that operating row-rise on the array will be slightly quicker
  - FORTRAN-contiguous order in memory (first index varies the fastest).
  - F order means that column-wise operations will be faster.
- **dtype :** [optional, float(byDefault)] Data type of returned array.
- **like:** [optional] allows you to create an array with the same shape and data type as another array-like object

Example 1:

Python3

```
# Python Program to create arra
import numpy as geek

a = geek.ones(3, dtype = int)
print("Matrix a : \n", a)

b = geek.ones([3, 3], dtype = i
print("\nMatrix b : \n", b)
```

Output:

```
Matrix a :
 [1 1 1]
Matrix b :
 [[1 1 1]
 [1 1 1]
 [1 1 1]]
```

Write a NumPy program to create a random set of rows from a 2D array.

Sample Solution:

Python Code:

```
# Importing the NumPy library and aliasing it as 'np'
import numpy as np

# Creating a new 2D NumPy array 'new_array' with random integers between 0 and 4 inclusive
new_array = np.random.randint(5, size=(5, 3))

# Printing a message indicating a random set of rows from the 2D array will be displayed
print("Random set of rows from 2D array array:")

# Printing the generated 2D NumPy array 'new_array'
print(new_array)
```

Sample Output:

Sample Output:

Random set of rows from 2D array

array:

```
[[4 0 2]
 [4 2 4]
 [1 0 4]
 [4 4 3]
 [3 4 3]]
```

5:57 pm ✓

Parameters of np.linspace
np.linspace takes three main parameters:

start – This is the starting value of the sequence.
stop – This is the end value of the sequence.
num – This is the number of evenly spaced samples to be generated. The default is 50.
Now, let's look at a basic example of how to use np.linspace:

```
import numpy as np
sequence = np.linspace(0, 10, 5)
print(sequence)

# Output:
# array([ 0. ,  2.5,  5. ,  7.5, 10. ])
```

Python
In this example, we're asking Python to generate a sequence of 5 numbers, starting at 0 and ending at 10. The output is an array of 5 numbers: 0, 2.5, 5, 7.5, and 10. These numbers are evenly spaced over the range of 0 to 10. This is the basic usage of np.linspace in Python.

Let's look at different examples on how to create equally spaced arrays with linspace() method of NumPy library in Python.

Example 1:

```
# Python Programming illustrating
# numpy.linspace method


import numpy as geek


# restep set to True

print("B\n", geek.linspace(2.0, 3.0,
num=5, retstep=True), "\n")


# To evaluate sin() in long range

x = geek.linspace(0, 2, 10)

print("A\n", geek.sin(x))
```

Output :

```
B
 (array([ 2.  ,  2.25,  2.5 ,  2.75,  3.  ]),
0.25)
A
 [ 0.         0.22039774  0.42995636
0.6183698   0.77637192  0.8961922
```

Example:

15

Input: start = 4, end = 15
Output: 4, 6, 8, 10, 12, 14

Input: start = 8, end = 11
Output: 8, 10

Example #1: Print all even numbers from the given list using for loop
Define start and end limit of range. Iterate from start till the range in the list using for loop and check if num % 2 == 0. If the condition satisfies, then only print the number.

```
# Python program to print all even numbers  in range

for even_numbers in range(4,15,2):

  #here inside range function first no denotes starting,

  #second denotes end and

  #third denotes the interval

    print(even_numbers,end=' ')
```

Output
4 6 8 10 12 14
Time Complexity: O(n)

Method 1 – Use A For Loop

When you want to create an array from 1 to 10 while increasing it in parts of 0.5, you can create an empty array, then loop over the range(0, k+1) or over (k+1), and then add a single element in the end by using the append function.

The first approach to counting from 1 to 10 in increment of 0.5, you can use a for loop


```
for n in range(1,11):

if n == 10:

print(n)

else:

print(n,n+0.5,end=" ")
```

Method 2 – Using The While Loop
In case you cannot use the for loop, you can also try using the while loop to count from 1 to 10 in increments of 0.5 on Python.

```
x = 1

while x <= 10:

print(x)

x += 0.5
```

You can also rewrite the code in the following way,

```
num = 1.0

while num < 10:

print(num, end=' ')

num +=0.5
```

6:23 pm ✓