Probability Calculation Using Logistic Regression

Logistic Regression is the statistical fitting of an s-curve logistic or logit function to a dataset in order to calculate the probability of the occurrence of a specific categorical event based on the values of a set of independent variables.

Logistic Regression is an easily interpretable classification technique that gives the probability of an event occurring, not just the predicted classification. It also provides a measure of the significance of the effect of each individual input variable, together with a measure of certainty of the variable's effect. An example use case is determining the probability of loan default for an individual based on personal financial data.

Team Studio supports the following two common forms of Logistic Regression:

- The most common and widely used form, Binomial Logistic Regression, is used to predict a single category or binary decision, such as "Yes" or "No." A classic use case is determining the probability of loan default for an individual based on personal financial data. Specifically, Binomial Logistic Regression is the statistical fitting of an s-curve logistic or logit function to a dataset in order to calculate the probability of the occurrence of a specific event, or Value to Predict, based on the values of a set of independent variables.
- The more general form is Multinomial Logistic Regression (MLOR)* which handles the case in which there are multiple categories to predict, not just two. It handles categorical data and predicts the probabilities of the different possible outcomes of a categorically distributed dependent variable. Example use cases might include weather predictions (sunny, cloudy, rain, snow), election predictions, or medical issue classification. Specifically, Multinomial Logistic Regression is the statistical fitting of a multinomial logit function to a dataset in order to calculate the probability of the occurrence of a multi-category dependent variable which

General Principles

Logistic regression analysis predicts the odds of an outcome of a categorical variable based on one or more predictor variables. A categorical variable is one that can take on a limited number of values, levels, or categories, such as "valid" or "invalid". A major advantage of Logistic Regression is its predictions are always between 0 and 1, unlike Linear Regression.

For example, a logistic model might predict the likelihood of a given person going to the beach as a function of temperature. A reasonable model might predict, for example, that a change in 10 degrees makes a person two times more or less likely to go to the beach. The term "twice as likely" for probability refers to the odds doubling (as opposed to the probability doubling). Rather, it is the odds that are doubling: from 2:1 odds, to 4:1 odds, to 8:1 odds, etc. Such a logistic model is called a log-odds model.

Hence, in statistics, Logistic Regression is sometimes called the logistic model or logit model. It is used for predicting the probability of the occurrence of a specific event by fitting data to a logit Logistic Function curve.

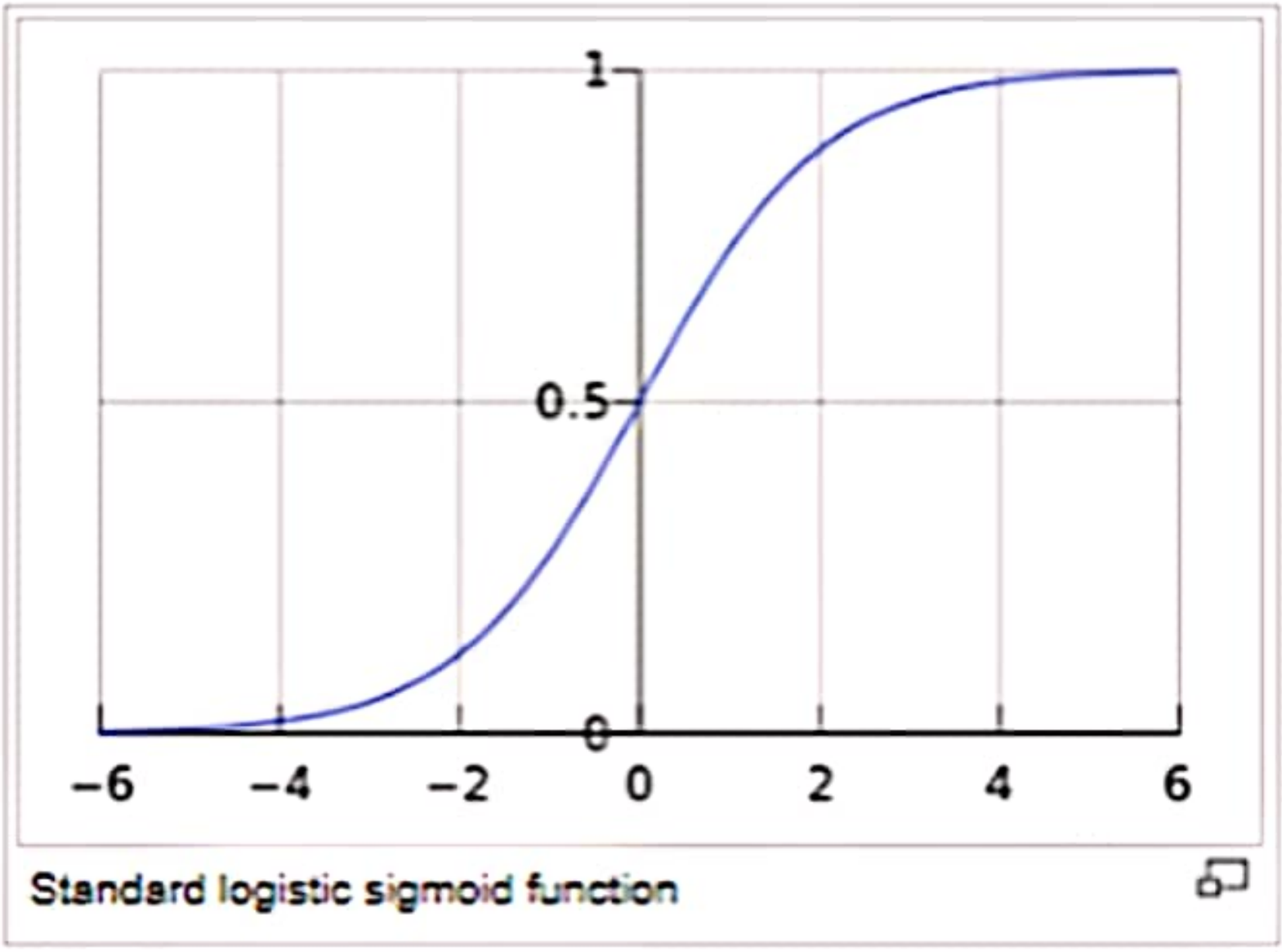
A Logistics Function is represented by an s-curve which was introduced by Pierre Verhulst in 1844, studied in relation to population growth. A generalized logistic curve can model the "S-shaped" behavior (abbreviated S-curve) of growth of some population P. The initial stage of growth is approximately exponential; then, as saturation begins, the growth slows, and at maturity, growth stops. This simple logistic function may be defined by the formula

$$P(t) = \frac{1}{1 + e^{-t}}$$

where the variable *P* denotes a population, *e* is Euler's number (2.72), and the variable *t* might be thought of as time.

For values of X in the range of real numbers from −∞ to +∞, the S-curve shown is obtained. In practice, due to the nature of the exponential function e^{-t}, it is sufficient to compute *t* over a small range of real numbers such as [−6, +6].

The following is an example of an S-Curve or Logistic Function.



Standard logistic sigmoid function

The Logistic Function can be applied to more generalized models that attempt to predict the probability of a specific event. In this case, there may be several factors or variables that contribute to whether the event happens. This Logistic Regression formula can be written generally in a linear equation form as:

$$\ln(P/(1-P)) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \dots$$

Where *P* = Probability of Event, and are the regression coefficients and *X1,X2,...* are the independent variable values. Solving for the Probability equation results in:

$$\textbf{Probability of event} = P = \frac{1}{1 + e^{-(\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots)}}$$

Logistic Regression Odds Ratio

The odds of an event occurring are defined as the probability of a case divided by the probability of a non-case given the value of the independent variable. The odds ratio is the primary measure of effect size in logistic regression and is computed to compare the odds that membership in one group leads to a case outcome with the odds that membership in some other group leads to a case outcome. The odds ratio (denoted OR) is simply calculated by the odds of being a case for one group divided by the odds of being a case for

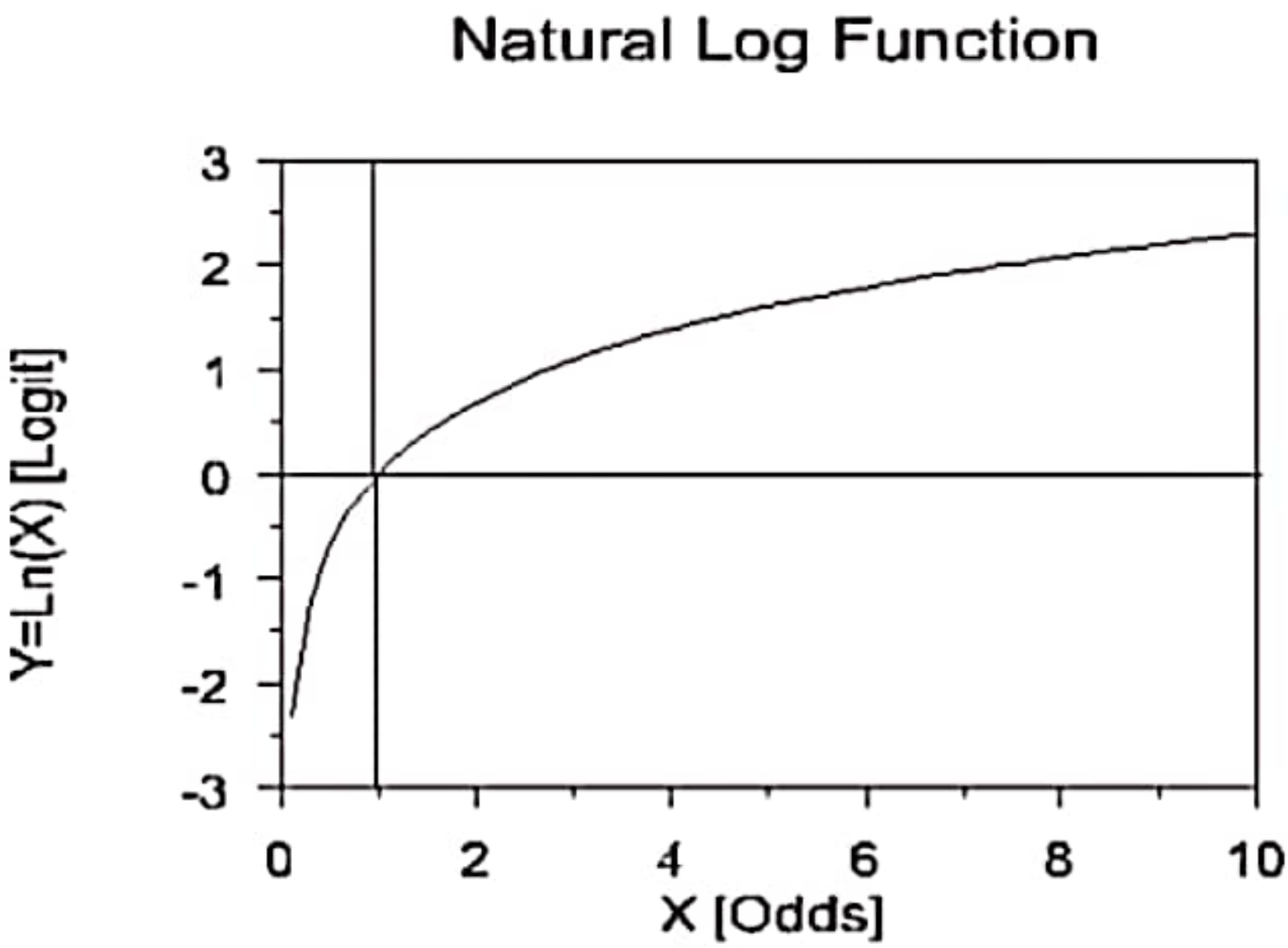
another group. This calculates how much a change in the independent variable affects the value of the dependent.

As an example, suppose we only know a person's height and we want to predict whether that person is male or female. We can talk about the probability of being male or female, or we can talk about the odds of being male or female. Let's say that the probability of being male at a given height is .90. Then the odds of being male would be:

$$odds = \frac{P}{1-P} = .9/.1 = 9 \text{ to } 1 \text{ odds}$$

Logistic Regression takes the natural logarithm of the odds (referred to as the logit or log-odds) to create a continuous criterion. The natural log function curve might look like the following.

Logistic Regression takes the natural logarithm of the odds (referred to as the logit or log-odds) to create a continuous criterion. The natural log function curve might look like the following.



The logit of success is then fit to the predictors using linear regression analysis. The results of the logit, however, are not intuitive, so the logit is converted back to the odds using the exponential function or the inverse of the natural logarithm. Therefore, although the observed variables in logistic regression are categorical, the predicted scores are actually modeled as a continuous variable (the logit).

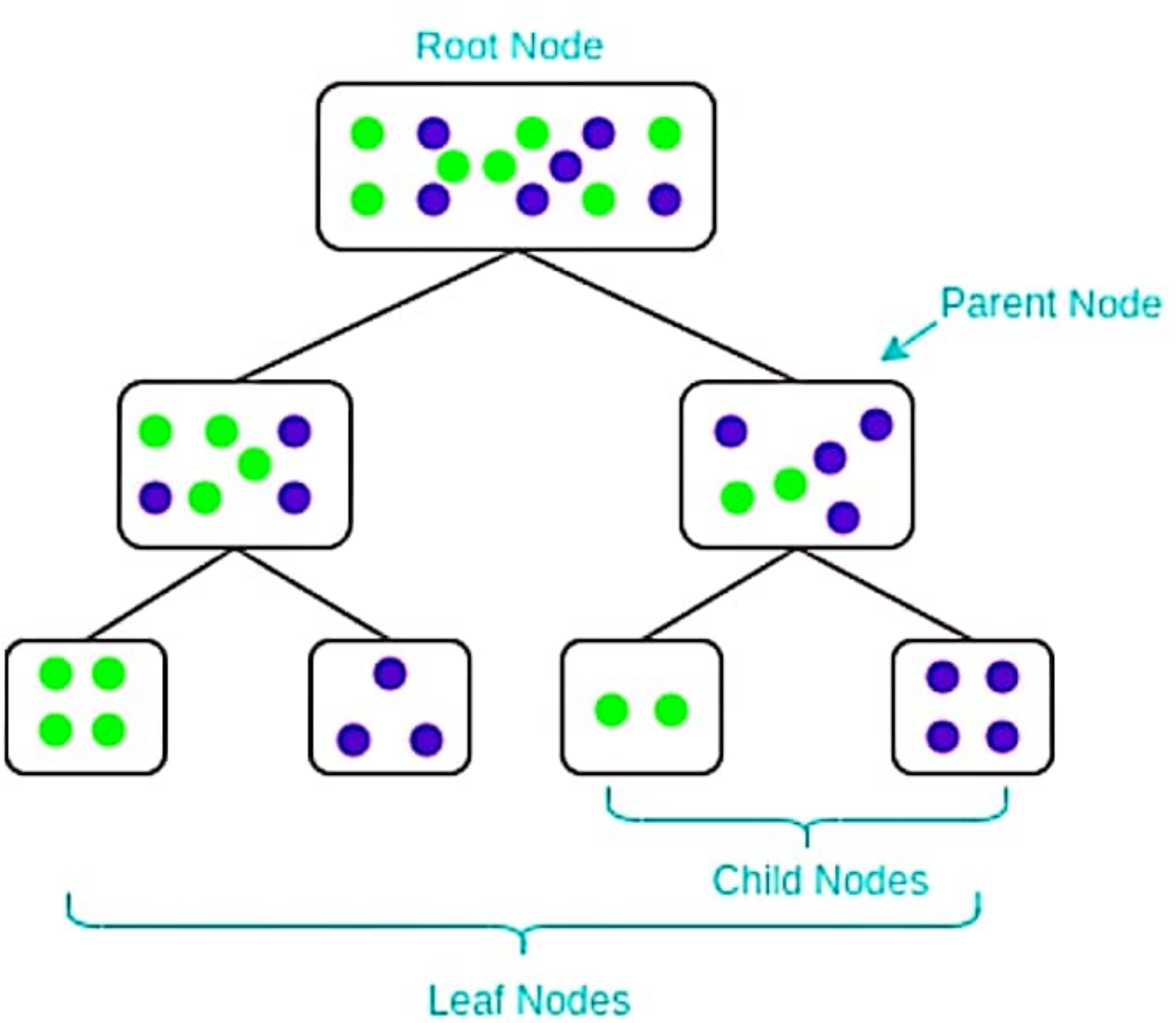
4 Simple Ways to Split a Decision Tree in Machine Learning (Updated 2024)

Introduction

A decision tree is a powerful machine learning algorithm extensively used in the field of data science. They are simple to implement and equally easy to interpret. It also serves as the building block for other widely used and complicated machine-learning algorithms like Random Forest, XGBoost, and LightGBM. I often lean on the decision tree algorithm as my go-to machine learning algorithm, whether I'm starting a new project or competing in a hackathon. In this article, I will explain 4 simple methods for split a decision tree.

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.

- **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.

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.

Decision Tree Classifier Building in Scikit-learn

The dataset that we have is a supermarket data which can be downloaded from here.

Load all the basic libraries.

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

Load the dataset. It consists of 5 features, `UserID`, `Gender`, `Age`, `EstimatedSalary` and `Purchased`.

```
data = pd.read_csv('/Users/ML/De
data.head()
```

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0

Dataset

We will take

only Age and `EstimatedSalary` as

our independent variables X because

of other features

like `Gender` and `User ID` are

irrelevant and have no effect on the

purchasing capacity of a person.

Purchased is our dependent

variable y.

```
feature_cols = ['Age','Estimated
y = data.iloc[:,4].values
```

The next step is to split the dataset

into training and test.

```
from sklearn.model_selection imp
X_train, X_test, y_train, y_test
```

Perform feature scaling

```
#feature scaling
from sklearn.preprocessing impor
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_t
X_test = sc_X.transform(X_test)
```

Fit the model in the Decision Tree

classifier.

```
from sklearn.tree import Decisio
classifier = DecisionTreeClassif
classifier = classifier.fit(X_tr
```

Make predictions and check

accuracy.

```
#prediction
y_pred = classifier.predict(X_te
from sklearn import metricsprint
```

The decision tree classifier gave an accuracy of 91%.

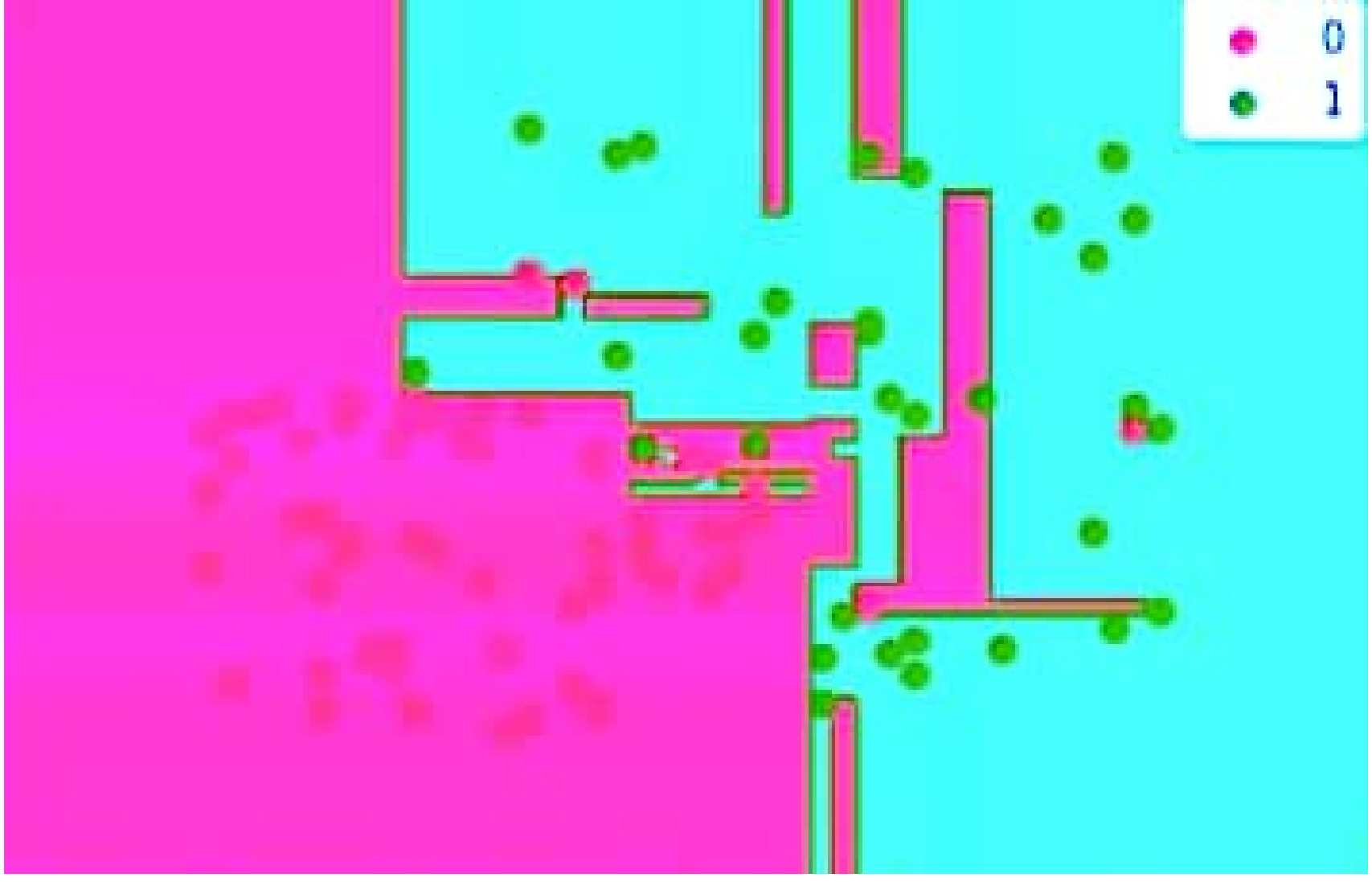
Confusion Matrix

```
from sklearn.metrics import conf
cm = confusion_matrix(y_test, y_
array([[64,  4],
       [ 2, 30]])
```

It means 6 observations have been classified as false.

Let us first visualize the model prediction results.

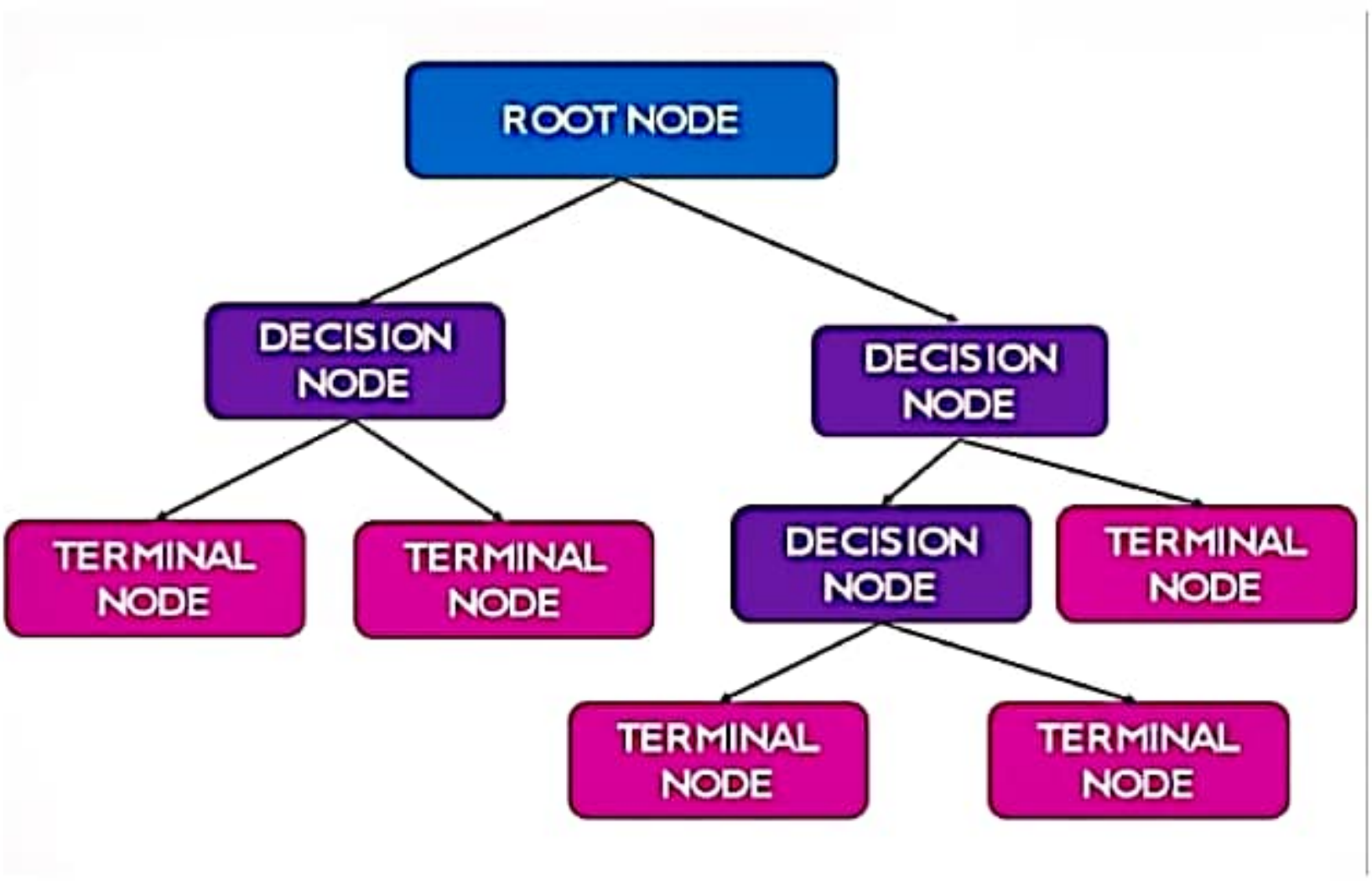
```
from matplotlib.colors import Li
X_set, y_set = X_test, y_test
X1, X2 = np.meshgrid(np.arange(s
plt.contourf(X1,X2, classifier.p
plt.ylim(X2.min(), X2.max())for
    plt.scatter(X_set[y_set==j,0
plt.title("Decision Tree(Test se
plt.xlabel("Age")
plt.ylabel("Estimated Salary")
plt.legend()
```



A Decision Tree learning is a predictive modeling approach. It is used to address classification problems in statistics, data mining, and machine learning. It is having a tree-like structure upside down and represents decisions or for decision-making. It can handle high dimension data and have good accuracy. Decision tree algorithm is used in many applications such as medical production, manufacturing, financial analysis, etc., For example, decision trees can be used in predicting the price of a house based on size and number of rooms or predicting if a person is male or female based on height and weight, and so on...

A decision tree can be represented as below. The topmost node is called the root node which has no incoming edges. An internal node represents a test or an attribute and each branch 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.

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, 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 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 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 sub-nodes increases the homogeneity, that is decreases the entropy of these nodes. The more the child node is homogeneous, the more the variance will be decreased after each split. Thus 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 N
outcome whereas yellow has 3 Yes

Red color has 3 Yes outcome and 3 No outcome whereas yellow has 3 Yes outcome and 1 No outcome.

E(S), we have already calculated and it is approximately equal to 0.971

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

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

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

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

$$= 0.924$$

Information Gain (S, Color) = E(S) – Weighted Average

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

Introduction

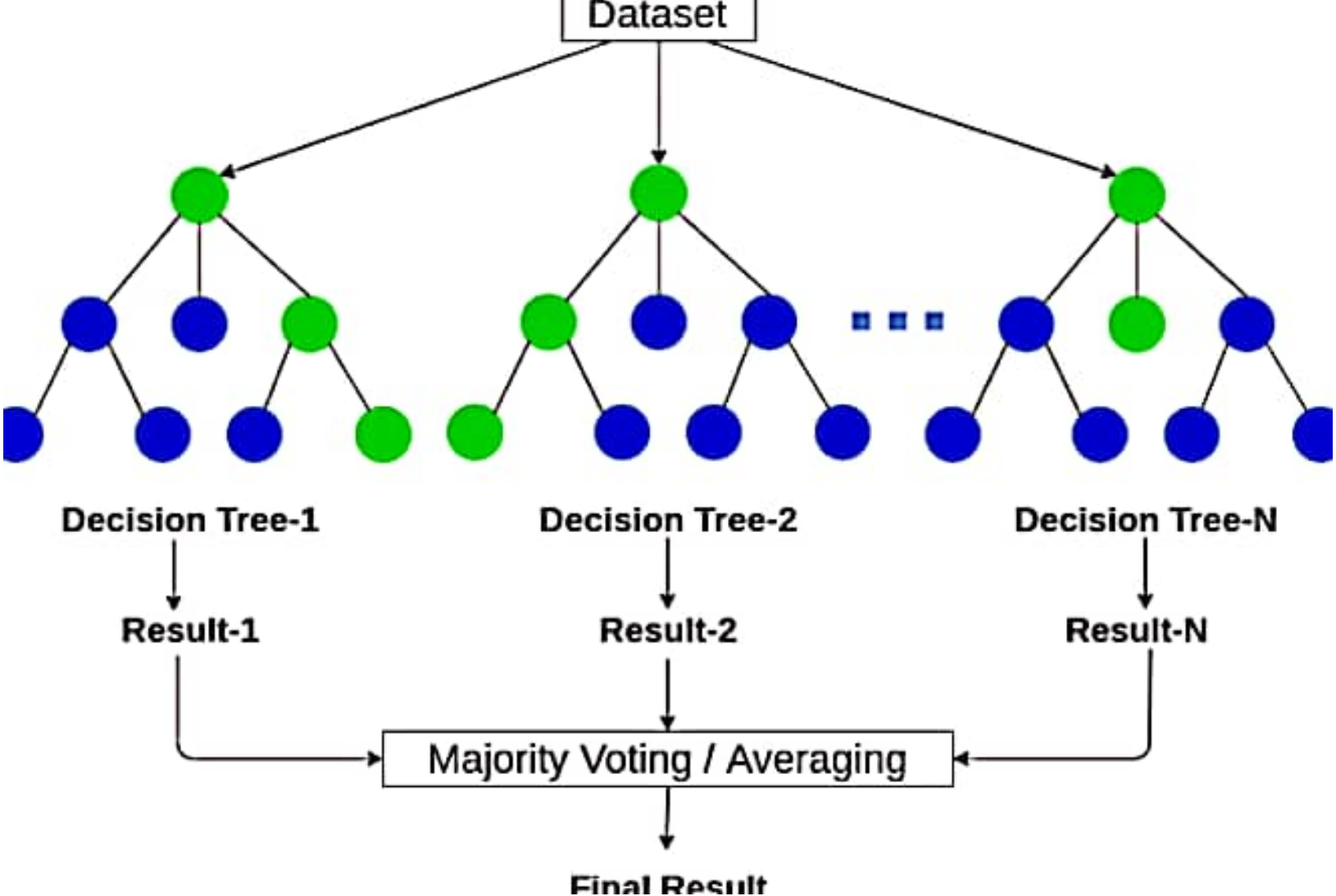
Random Forest is a widely-used machine learning algorithm developed by Leo Breiman and Adele Cutler, which combines the output of multiple decision trees to reach a single result. Its ease of use and flexibility have fueled its adoption, as it handles both classification and regression problems. In this article, we will understand how random forest algorithm works, how it differs from other algorithms and how to use it.

Learning Objectives

- Learn the working of random forest with an example.
- Understand the impact of different hyperparameters in random forest.
- Implement Random Forest on a classification problem using scikit-learn.

What is Random Forest Algorithm?

Random Forest Algorithm widespread popularity stems from its user-friendly nature and adaptability, enabling it to tackle both classification and regression problems effectively. The algorithm's strength lies in its ability to handle complex datasets and mitigate overfitting, making it a valuable tool for various predictive tasks in machine learning.



A Decision Tree learning is a predictive modeling approach. It is used to address classification problems in statistics, data mining, and machine learning. It is having a tree-like structure upside down and represents decisions or for decision-making. It can handle high dimension data and have good accuracy. Decision tree algorithm is used in many applications such as medical production, manufacturing, financial analysis, etc., For example, decision trees can be used in predicting the price of a house based on size and number of rooms or predicting if a person is male or female based on height and weight, and so on...

The k-nearest neighbour (k-NN) classifier is a conventional non-parametric classifier (Cover and Hart 1967). To classify an unknown instance represented by some feature vectors as a point in the feature space, the k-NN classifier calculates the distances between the point and points in the training data set. Usually, the Euclidean distance is used as the distance metric. Then, it assigns the point to the class among its k nearest neighbours (where k is an integer). Figure 1 illustrates this concept where * represents the point. If $k = 1$, the point belongs to the dark square class; if $k = 5$, the small circle class which are the majority class of the five nearest points.

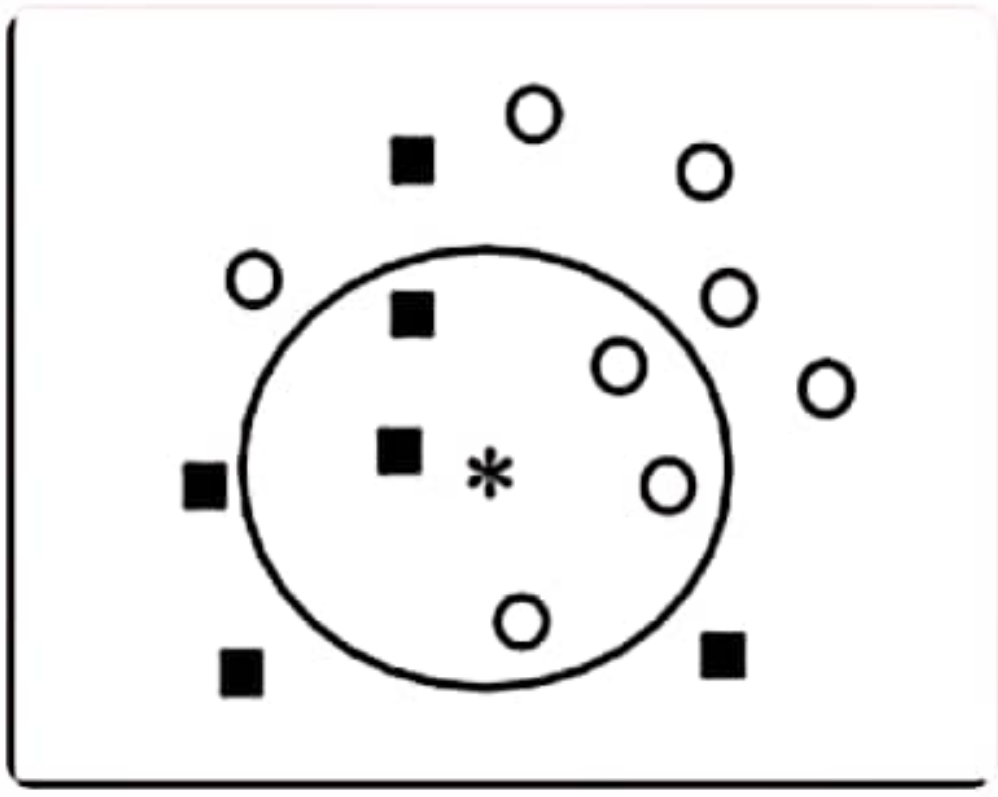


Fig. 1

k-Nearest neighbor classification

As *k*-NN does not require the off-line training stage, it main computation is the on-line 'searching' for the *k* nearest neighbours of a given testing example. Although using different *k* values are likely to produce different classificatio results, 1-NN is usually used as a benc mark for the other classifiers since it c provide reasonable classification perfc mances in many pattern classification problems (Jain et al. 2000).

↑ Back to Top

To measure the distance between points A and B in a feature space, various distance functions have been used in the literature, in which the Euclidean distance function is the most widely used one. Let A and B are represented by feature vectors $A = (x_1, x_2, \ldots, x_m)$ and $B = (y_1, y_2, \ldots, y_m)$, where m is the dimensionality of the feature space. To calculate the distance between A and B, the normalized Euclidean metric is generally used by

$$dist\,(A, B) = \sqrt{\frac{\sum_{i=1}^{m} (x_i - y_i)^2}{m}} \qquad 1$$

$$sim\,(A, B) = \frac{\vec{A} \cdot \vec{B}}{\left|\vec{A}\right|\left|\vec{B}\right|} \qquad 2$$

where the numerator represents the dot product of the vectors $\vec{A}$ and $\vec{B}$, while the denominator is the product of their Euclidean lengths.

Some other distance functions are also available for k-NN classification, such as Minkowsky[1] (Batchelor 1978), correlation, and Chi square (Michalski et al. 1981).

$$dist\_Minkowsky\,(A, B) = \left(\sum_{i=1}^{m} |x_i - y_i|^r\right)^{1/r} \qquad 3$$

$$dist\_correlation(A,B)$$

$$= \frac{\sum_{i=1}^{m}(x_i-\mu_i)(y_i-\mu_i)}{\sqrt{\sum_{i=1}^{m}(x_i-\mu_i)^2 \sum_{i=1}^{m}(y_i-\mu_i)^2}} \quad (4)$$

$$dist\_Chi\text{-}square(A,B) =$$

$$\sum_{i=1}^{m} \frac{1}{sum_i}\left(\frac{x_i}{size_Q} - \frac{y_i}{size_I}\right)^2 \quad (5)$$

Naive Bayes Classifiers

This article explores Naive Bayes classifiers, a family of algorithms based on Bayes' Theorem. Despite the "naive" assumption of feature independence, these classifiers are widely utilized for their simplicity and efficiency in machine learning. The article delves into theory, implementation, and applications, shedding light on their practical utility despite oversimplified assumptions.

What is Naive Bayes classifiers?

Naive Bayes classifiers are a collection of classification algorithms based on Bayes' Theorem. It is not a single algorithm but a family of algorithms where all of them share a common principle, i.e. every pair of features being classified is independent of each other. To start with, let us consider a dataset.

Training of CNN ir TensorFlow

The MNIST database (**Modified Nationa Institute of Standard Technology database**) is an extensive database of handwritten digits which is used for training various image processing systems. It was created by "**reintegrating**" samples from the origina dataset of the **MNIST**.

If we are familiar with the building blocks of Connects, we are ready to build one with TensorFlow. We use the MNIST dataset for image classification.

Preparing the data is the same as in the previous tutorial. We can run code and jump directly into the architecture of CNN.

```
import numpy as np
import tensorflow as tf


from sklearn.datasets import fetch_mldata
#Change USERNAME by the username of the machine
##Windows USER
mnist = fetch_mldata('C:\\Users\\USERNAME\\Downloads
```

```
## Mac User
mnist = fetch_mldata('/Users/USERNAME/Downloads/MN
print(mnist.data.shape)
print(mnist.target.shape)
from sklearn.model_selection import train_test_split
A_train, A_test, B_train, B_test = train_test_split(mnist.data,
B_train  = B_train.astype(int)
```

```
A_train, A_test, B_train, B_test = train_test_split(mnist.data,
B_train  = B_train.astype(int)
B_test  = B_test.astype(int)
batch_size =len(X_train)
print(A_train.shape, B_train.shape,B_test.shape )
## rescale
from sklearn.preprocessing import MinMaxScaler
```

```
scaler = MinMaxScaler()

# Train the Dataset

X_train_scaled = scaler.fit_transform(A_train.astype(np.float65))
```

```
#test the dataset
X_test_scaled = scaler.fit_transform(A_test.astype(np.float65))
feature_columns = [tf.feature_column.numeric_column('x',s
X_train_scaled.shape[1:]
```

What is Regularization? 12

Regularization is one of the most important concepts of machine learning. It is a technique to prevent the model from overfitting by adding extra information to it.

Sometimes the machine learning model performs well with the training data but does not perform well with the test data. It means the model is not able to predict the output when deals with unseen data by introducing noise in the output, and hence the model is called overfitted. This problem can be deal with the help of a regularization technique.

This technique can be used in such a way that it will allow to maintain all variables or features in the model by reducing the magnitude of the variables. Hence, it maintains accuracy as well as a generalization of the model.

How does Regularization Work?

Regularization works by adding a penalty or complexity term to the complex model. Let's consider the simple linear regression equation:

$$y= \beta0+\beta1x1+\beta2x2+\beta3x3+\ldots+\beta nxn +b$$

In the above equation, Y represents the value to be predicted

X1, X2, ...Xn are the features for Y.

β0,β1,.....βn are the weights or magnitude attached to the features, respectively. Here represents the bias of the model, and b represents the intercept.

Linear regression models try to optimize the β0 and b to minimize the cost function. The equation for the cost function for the linear model is given below:

$$\sum_{i=1}^{M}(y_i - y'_i)^2 = \sum_{i=1}^{M}(y_i - \sum_{j=0}^{n}\beta_j * Xij)^2$$

Now, we will add a loss function and optimize parameter to make the model that can predict the accurate value of Y. The loss function for the linear regression is called as **RSS or Residual sum of squares.**

What is accuracy?

- Visual example
- Accuracy paradox
- Pros and cons

Confusion matrix

What is precision?

- Visual example
- Pros and cons

What is recall?

- Visual example
- Pros and cons

Practical considerations

Accuracy, precision, and recall in ML monitoring

Accuracy, precision, and recall in Python

14

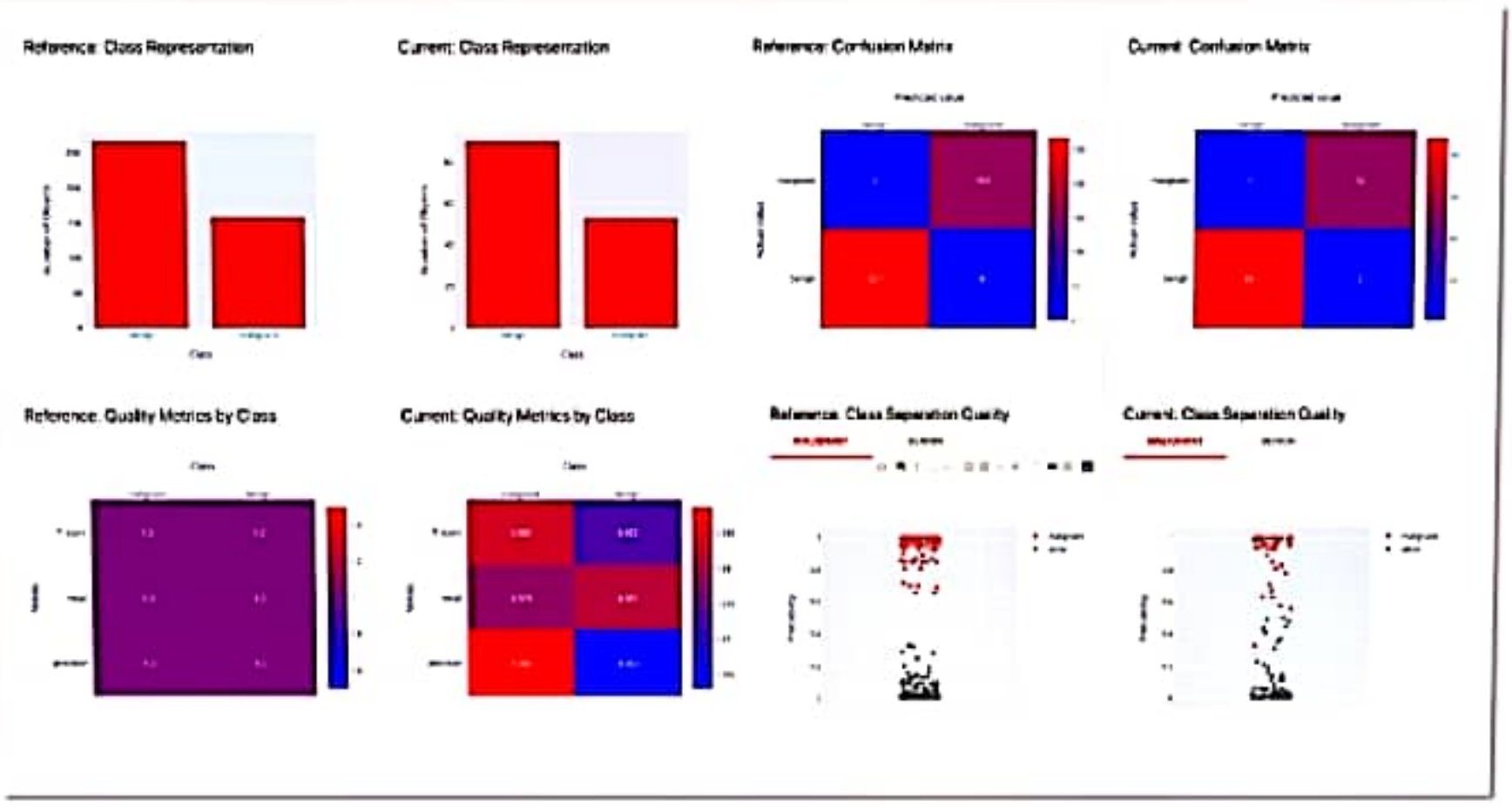
This article is a part of the Classification Metrics guide.

Accuracy, precision, and recall help evaluate the quality of classification models in machine learning. Each metric reflects a different aspect of the model quality, and depending on the use case, you might prefer one or another. This chapter explains the pros and cons.

We will also demonstrate how to calculate accuracy, precision, and recall using the open-source Evidently Python library.

TL;DR

- **Accuracy** shows how often a classification ML model is correct **overall**.
- **Precision** shows how often an ML model is correct when **predicting the target class.**
- **Recall** shows whether an ML model can find **all objects of the target class**.



Get an instant model quality report

Looking to understand your ML model quality? Try Evidently, an open-source Python library with 3m+ downloads. Get an interactive performance report with just a couple of lines of code.

Get started →

A classification model aims to assign a pre-defined label to the objects in the input data. For example, you might want to predict if a user will stop using a certain software product. You will then create an ML model that classifies all users into “churner” or “non-churner” categories.

elements of a set.

- How to create a Python set with set literals, the `set()` constructor function, and the set comprehension syntax.
- How to add elements to and remove elements from Python sets
- How to combine sets with the union, intersection, difference, and symmetric difference Python set operations.
- How to compare Python sets with the subset, superset, and disjoint relationships.

What Are Sets in Python?

In Python, a set is an unordered, mutable collection of unique elements. Mathematical set operations like union, intersection, and difference can be performed on them. Two different sets can be compared with the subset, superset, and disjoint relations.

This example set uses the *set literal* syntax and contains the even integers between 2 and 8:

```
{2, 4, 6, 8}
```

Sets can contain elements of different types. This set contains some integers and some strings:

```
{2, "hello", 4, "world"}
```

Getting Started with Python Sets and Python Set Operations

Updated April 25, 2023, by Nathaniel Stickman

Pythons sets are unordered collections modeled on mathematical sets, in which elements are unique. Python sets support the logical operations of mathematical sets, like union, intersection, and difference. This example code declares a new set and adds the elements of a second set to it, showing that the updated set contains unique elements:

```
example_set = {1, 3, 5, 7}

example_set.update({1, 2, 3, 4})

print(example_set)
```

```
{1, 2, 3, 4, 5, 7}
```

This guide explains:

```
{2, "hello", 4, "world"}
```

The members of a set must be *hashable*. Hashable objects are generally *immutable*. Hashable objects must implement the `__eq__()` method, so two different hashable objects and can be compared to check if they are equal in value to each other. For example, two string objects can be compared to check if they are the same string.

> **Note**
>
> Technically, a hashable object's `__hash__()` function must always return the same value for the lifetime of the object, rather than the object itself being immutable. This blog post describes why, in practice, hashable objects used with sets are also immutable.

Integers and strings are built-in Python types that are hashable and can be stored in a set. Mutable container types like lists, dictionaries, and sets are not hashable, because their contents can change. A Python tuple is hashable if each of its values is hashable.

Create a Python Set

Python provides a few ways to create sets:

- A Python set can be declared with the *set literal* syntax. A set literal consists of a comma-separated list of unique, hashable objects wrapped in curly braces.

hashable if each of its values is hashable.

Create a Python Set

Python provides a few ways to create sets:

- A Python set can be declared with the *set literal* syntax. A set literal consists of a comma-separated list of unique, hashable objects wrapped in curly braces.

```
example_set = {2, 4, 6, 8}
```

An empty set cannot be expressed with the set literal syntax. An empty pair of curly braces `{}` represents an empty Python dictionary. Use the `set()` constructor (below) to create an empty set.

- The `set()` constructor function accepts a Python list (or any other iterable object) as input and returns a set with unique values from the list. The items in the input list/iterable must be hashable.

```
example_set = set([2, 4, 6, 8, , (
print(example_set)
```

```
{8, 2, 4, 6}
```

If no argument is passed to the constructor function, an empty set is created:

Python offers a datatype called set whose elements must be unique. It can be used to perform different set operations like union, intersection, difference and symmetric difference.

Source Code

```
# Program to perform different set

# define three sets
E = {0, 2, 4, 6, 8};
N = {1, 2, 3, 4, 5};

# set union
print("Union of E and N is",E | N)

# set intersection
print("Intersection of E and N is",

# set difference
print("Difference of E and N is",E

# set symmetric difference
print("Symmetric difference of E an
```

Python Program to Illustrate Different Set Operations

To understand this example, you should have the knowledge of the following Python programming topics:

- Python Sets
- Python Basic Input and Output

Python offers a datatype called set whose elements must be unique. It can be used to perform different set operations like union, intersection, difference and symmetric

Output

```
Union of E and N is {0, 1, 2, 3, 4,
Intersection of E and N is {2, 4}
Difference of E and N is {8, 0, 6}
Symmetric difference of E and N is
```

In this program, we take two different sets and perform different set operations on them. This can equivalently done by using set methods.

To learn more, visit Python Set Methods.

This is an example of while loop in C programming language - In this C program, we are going to print numbers from 1 to 10 using while loop.

Submitted by **IncludeHelp**, on March 07, 2018

Using while loop, in this C program we are going to print the numbers from 1 to 10.

To print the numbers from 1 to 10,

- We will declare a variable for loop counter (`number`).
- We will check the condition whether loop counter is less than or equal to 10, if condition is true numbers will be printed.
- If condition is false – loop will be terminated.

Program to print numbers from 1 to 10 using while loop in C

Copy

```
#include <stdio.h>

int main()
```

Copy

```c
#include <stdio.h>

int main()
{

        //loop counter declaration
        int number;

        //assign initial value
        //from where we want to prin
        number =1;

        //print statement
        printf("Numbers from 1 to 10

        //while loop, that will prin
        //from 1 to 10
        while(number<=10)
        {
                //printing the numbe
                printf("%d ",number)
                //increasing loop co
                number++;
        }

        return 0;
}
```

Output

```
Numbers from 1 to 10:
1 2 3 4 5 6 7 8 9 10
```

Calculate the Factorial of a Number using Loop

In this article, we will discuss how to calculate the factorial of a number with its working example in the R Programming Language using R while loop. In R programming, loops are essential constructs that allow us to repeat a set of instructions multiple times. The while loop is one such construct that repeatedly executes a block of code until a certain condition is met. This loop is particularly useful when you want to iterate over elements without knowing the exact number of iterations in advance. In this article, we will explore how to use the while loop to print the factorial of a number and understand its step-by-step implementation.

Formula of factorial:

```
fact(N)=N*fact(N-1)
```

Find the greatest number in given three numbers

```
#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>
```

Returns :

ndarray of ones having given shape, order

Python

```
# Python Program illustrating
# numpy.ones method

import numpy as geek

b = geek.ones(2, dtype = int)
print("Matrix b : \n", b)

a = geek.ones([2, 2], dtype = i
print("\nMatrix a : \n", a)

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

Output :

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

numpy.ones() in Python

The **numpy.ones()** function returns a new array of given shape and type, with ones.

```
Syntax: numpy.ones(shape, dtype = None, or
```

Parameters :

```
shape : integer or sequence of integers
order  : C_contiguous or F_contiguous
         C-contiguous order in memory(last
         C order means that operating row-
         FORTRAN-contiguous order in memor
         F order means that column-wise op
dtype : [optional, float(byDefault)] Data
```

Returns :

```
ndarray of ones having given shape, order
```

Making a list of evenly spaced numbers in a certain range in python

Asked 12 years, 7 months ago

Modified 1 year, 5 months ago Viewed 200k times

65

What is a pythonic way of making list of arbitrary length containing evenly spaced numbers (not just whole integers) between given bounds? For instance:

```
my_func(0,5,10) # ( lower_bound
# [ 0, 0.5, 1, 1.5, 2, 2.5, 3, 3
```

Note the `Range()` function only deals with integers. And this:

```
def my_func(low,up,leng):
    list = []
    step = (up - low) / float(le
    for i in range(leng):
        list.append(low)
        low = low + step
    return list
```

seems too complicated. Any ideas?

```
# Python Program illustrating
# numpy.ones method

import numpy as geek

b = geek.ones(2, dtype = int)
print("Matrix b : \n", b)

a = geek.ones([2, 2], dtype = i
print("\nMatrix a : \n", a)

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

Output :

```
Matrix b :
 [1 1]

Matrix a :
 [[1 1]
 [1 1]]

Matrix c :
 [[ 1.  1.  1.]
 [ 1.  1.  1.]
 [ 1.  1.  1.]]
```

with integers. And this:

```
def my_func(low,up,leng):
    list = []
    step = (up - low) / float(le
    for i in range(leng):
        list.append(low)
        low = low + step
    return list
```

---

seems too complicated. Any ideas?

list range python

Share Improve this question Follow

asked Jul 13, 2011 at 18:23

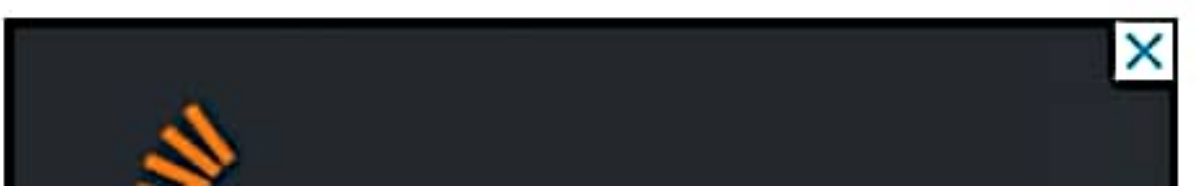
Double AA

5,819 • 16 • 44 • 56

There was some nice solutions: stackoverflow.com/questions/477486/... – Pill Jul 13, 2011 at 18:29

2 Note that this only works accurately for some sequences due to the (necessary) inaccuracy of floating point numbers. – user395760 Jul 13, 2011 at 18:30

Add a comment



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

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

    return 0;
}
```

Sample output:

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

Example 2: Calculate factorial of 10

R

```r
factorial <- function(n) {
  result <- 1
  while (n > 0) {
    result <- result * n
    n <- n - 1
  }
  return(result)
}

factorial(10)
```

Output:

```
[1] 3628800
```

- First we will initialize result=1.
- Create a while loop with the condition 10 > 0.
- This loop will run as long as 10 is greater than zero.
- Multiply the result by the value of n.
- Decrement the value of 10 by 1.
- After the loop, return the value of result as the calculated factorial.

- First we will initialize result= 1.
- Create a while loop with the condition 10 > 0.
- This loop will run as long as 10 is greater than zero.
- Multiply the result by the value of n.
- Decrement the value of 10 by 1.
- After the loop, return the value of result as the calculated factorial.

Example 3: Calculate factorial using user defined value

R

```
number <- as.integer(readline("
factorial <- 1
var <- 2

while (var <= number) {
  factorial <- factorial * var
  var <- var + 1
}

cat("Factorial:", factorial, "\
```

Output:

ers

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;

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

```r
factorial <- function(n) {
  result <- 1
  while (n > 0) {
    result <- result * n
    n <- n - 1
  }
  return(result)
}

factorial(5)
```

Output:

```
[1]120
```

- First we will initialize result=1.
- Then we create a while loop with the condition 5 > 0.
- This loop will run as long as 5 is greater than zero.
- Multiply the result by the value of 5.
- Decrement the value of n by 1.
- After the loop, return the value of result as the calculated factorial.

Example 2: Calculate factorial of 10

using Loop

In this article, we will discuss how to calculate the factorial of a number with its working example in the R Programming Language using R while loop. In R programming, loops are essential constructs that allow us to repeat a set of instructions multiple times. The while loop is one such construct that repeatedly executes a block of code until a certain condition is met. This loop is particularly useful when you want to iterate over elements without knowing the exact number of iterations in advance. In this article, we will explore how to use the while loop to print the factorial of a number and understand its step-by-step implementation.

Formula of factorial:

```
fact(N)=N*fact(N-1)
```

Formula of factorial:

```
fact(N)=N*fact(N-1)
```

Syntax:

```
factorial <- function(n) {
  result <- 1
  while (n > 0) {
    result <- result * n
    n <- n - 1
  }
  return(result)
}
```

Example 1: Calculate factorial of 5

R

```
factorial <- function(n) {
  result <- 1
  while (n > 0) {
    result <- result * n
    n <- n - 1
  }
```

(necessary) inaccuracy of floating point numbers. – user395760 Jul 13, 2011 at 18:30

Add a comment

7 Answers

Sorted by:

Highest score (default)

93

Given numpy, you could use linspace:

Including the right endpoint (5):

```
In [46]: import numpy as np
In [47]: np.linspace(0,5,10)
Out[47]:
array([ 0.        ,  0.55555556,
        2.77777778,  3.33333333,
```

Excluding the right endpoint:

terminated.

Program to print numbers from 1 to 10 using while loop in C

Copy

```
#include <stdio.h>

int main()
{

    //loop counter declaration
    int number;

    //assign initial value
    //from where we want to prin
    number =1;

    //print statement
    printf("Numbers from 1 to 10

    //while loop, that will prin
    //from 1 to 10
    while(number<=10)
    {
            //printing the numbe
            printf("%d ",number)
            //increasing loop co
            number++;
    }
```

93

Given numpy, you could use linspace:

Including the right endpoint (5):

```
In [46]: import numpy as np
In [47]: np.linspace(0,5,10)
Out[47]:
array([ 0.        ,  0.55555556,
        2.77777778,  3.33333333,
```

Excluding the right endpoint:

```
In [48]: np.linspace(0,5,10,endp
Out[48]: array([ 0. ,  0.5,  1.
```

Share Improve this answer Follow

edited Jul 13, 2011 at 19:52

answered Jul 13, 2011 at 18:37

unutbu

858k • 190 • 1814 • 1694

14 @Double AA: True; if you need a list, you could use `np.linspace(0,5,10).tolist()` . – unutbu Jul 13, 2011 at 19:52

Add a comment

You can use the following approach:

Program to print numbers from 1 to 10 using while loop in C

Copy

```
#include <stdio.h>

int main()
{

        //loop counter declaration
        int number;

        //assign initial value
        //from where we want to print
        number =1;

        //print statement
        printf("Numbers from 1 to 10:

        //while loop, that will print
        //from 1 to 10
        while(number<=10)
        {
                //printing the number
                printf("%d ",number);
                //increasing loop cou
                number++;
        }

        return 0;
}
```

53

You can use the following approach:

```
[lower + x*(upper-lower)/length
```

lower and/or upper must be assigned as floats for this approach to work.

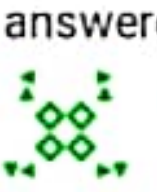
Share Improve this answer Follow

edited Jul 13, 2011 at 22:10

milancurcic

6,233 • 2 • 34 • 47

answered Jul 13, 2011 at 18:26



Howard

38.9k • 9 • 66 • 85

9 Nice to have a concise stock python alternative in case numpy is not available. – feedMe Feb 9, 2016 at 14:25

6 this updated version produces the *exact* same output as numpy linspace: `[lower + x*(upper-lower)/(length-1) for x in range(length)` – Trav May 10, 2018 at 14:40

@Trav it does not provide the exact same. The results are not numerically identical for when ascending linspace, and for descending (example: np.linspace(1,0.2,4)) it is very off. Still, its a good effort to get same ballpark numbers for ascending case

answered May 10, 2019 at 15:53

Robin Dinse

1,511 • 14 • 20

Add a comment

1

You can use the folowing code:

```
def float_range(initVal, itemCou
    for x in xrange(itemCount):
        yield initVal
        initVal += step

[x for x in float_range(1, 3, 0.
```

Share Improve this answer Follow

answered Jul 13, 2011 at 19:54

Artsiom Rudzenka

28.3k • 4 • 34 • 53

Add a comment

0

Similar to Howard's answer but a bit more efficient:

```
def my_func(low, up, leng):
    step = ((up-low) * 1.0 / ler
    return [low+i*step for i in
```

Share Improve this answer Follow

answered Jul 13, 2011 at 18:33

Peter Collingridge

10.9k • 3 • 45 • 62

29 Aug 2023

Numpy's np.linspace: Evenly Spaced Sequences in Python

Posted in Programming / Coding, Python By Gabriel Ramuglia On August 29, 2023

Ever wondered how to generate a sequence of evenly spaced numbers in Python? Just like a skilled mathematician, np.linspace can do this for you. This function is a part of the Numpy library, a powerful tool used by data scientists, researchers, and Python enthusiasts around the globe.

In this guide, we will walk you through the use of np.linspace, from basic usage to advanced techniques. Whether you are a beginner just starting out with Python or a seasoned programmer looking to refine your skills, this article is designed to help you understand and effectively use np.linspace in your projects.

input

Python3

```
# Python program to print Even

start = int(input("Enter the st
end = int(input("Enter the end
# iterating each number in list
for num in range(start, end + 1

    # checking condition
    if num % 2 == 0:
        print(num, end=" ")
```

Output:

```
Enter the start of range: 4
Enter the end of range: 10
4 6 8 10
```

The **time complexity** of this program is O(n), where n is the number of integers in the range.

The **space complexity** of this program is also O(1), which is constant.

Example#3 Taking range limit user input and using skip sequence number in range function

Python3

```
x=[i for i in range(4,15+1) if
print(*x)
```

Output

```
4 6 8 10 12 14
```

Method: Using enumerate function

Python3

```
a=4;b=15;l=[]
for i in range(a,b+1):
  l.append(i)
print([a for j,a in enumerate(l
```

Output

```
[4, 6, 8, 10, 12, 14]
```

Method: Using pass

Output

```
[4, 6, 8, 10, 12, 14]
```

Time Complexity: time complexity of the program is O(N).

Space Complexity:
The program creates a list of integers between a and b, which takes O(N) space in the worst case. Then, it creates a new list to store even numbers returned by the filter function. The space used by this new list depends on the number of even numbers in the input range, which is at most N/2. Therefore, the space complexity of the program is O(N).

Method: Using list comprehension

Python3

```
x=[i for i in range(4,15+1) if
print(*x)
```

Example#3 Taking range limit user input and using skip sequence number in range function which generates the all-even number.

Python3

```
# Python program to print Even

start = int(input("Enter the st
end = int(input("Enter the end
#creating even starting range
start = start+1 if start&1 else

#create a list and printing ele
#contains Even numbers in range
[ print( x ) for x in range(sta
```

Output:

```
Enter the start of range: 4
Enter the end of range: 10
4 6 8 10
```

The **time complexity** of this program is O(n), where n is the number of integers in the range.

The **space complexity** of this program is also

Output

```
4 6 8 10 12 14
```

Time Complexity: O(b-a)

Method: Using Numpy.Array

Python3

```
# Python code To print all even
# in a given range using numpy
import numpy as np
# Declaring Range
a=4;b=15
li= np.array(range(a, b+1))
# printing odd numbers using nu
even_num = li[li%2==0];
print(even_num)
```

Output:

```
[ 4  6  8 10 12 14]
```

Method: Using bitwise and operator

Python3

the number and 1 is equal to the number itself.

4. If the above condition is not satisfied, then it means the number is even, so print it.
5. The loop continues until all the numbers in the range have been checked.

Python3

```
# Python program to print even
#using bitwise | operator

start, end = 4, 19

# iterating each number in list
for num in range(start, end + 1

    # checking condition
    if not (num | 1)==num:
        print(num, end = " ")
#this code is contributed by tv
```

Output

4 6 8 10 12 14 16 18

Method: Using bitwise and operator

Python3

```
# Python program to print even
#using bitwise & operator

start, end = 4, 19

# iterating each number in list

for num in range(start, end + 1

    # checking condition

    if not (num & 1):

        print(num, end = " ")
#this code is contributed by vi
```

Output

```
4 6 8 10 12 14 16 18
```

Time Complexity: O(n)

Auxiliary Space:O(1)

4 6 8 10 12 14 16 18

Time Complexity: O(n)

Auxiliary Space:O(1)

Method: Using bitwise or operator

Algorithm:

1. Take the start and end values for the range.
2. Using a for loop, iterate through each number in the range.
3. Check if the bitwise OR operation between the number and 1 is equal to the number itself.
4. If the above condition is not satisfied, then it means the number is even, so print it.
5. The loop continues until all the numbers in the range have been checked.

Python3

```
# Python program to print even
#using bitwise | operator

start, end = 4, 19

# iterating each number in list
```

Space Complexity:

The program uses a constant amount of extra space, regardless of the size of the input range. Therefore, the space complexity is O(1).

Method: Using the lambda function

Python3

```
# Python code To print all even
# in a given range using the la
a=4;b=15
li=[]
for i in range(a,b+1):
    li.append(i)
# printing odd numbers using th
even_num = list(filter(lambda x
print(even_num)
```

Output

```
[4, 6, 8, 10, 12, 14]
```

Time Complexity: time complexity of the program is O(N).

Python3

```
a=4;b=15
for i in range(a,b+1):
  if i%2!=0:
    pass
  else:
    print(i,end="  ")
```

Output

```
4  6  8  10  12  14
```

Time Complexity: O(b-a)

Method: Using Numpy.Array

Python3

```
# Python code To print all even
# in a given range using numpy
import numpy as np
# Declaring Range
```

where n is the number of integers in the range.

The **space complexity** of this program is also O(1), which is constant.

Method: Using recursion

Python3

```
def even(num1,num2):
    if num1>num2:
        return
    print(num1,end="  ")
    return even(num1+2,num2)
num1=4;num2=15
even(num1,num2)
```

Output

```
4  6  8  10  12  14
```

Time Complexity:

The time complexity of the program is O(N/2), where N is the difference between num2 and num1 since the program only checks and prints even numbers between num1 and num2.

Python program to print all even numbers in a range

Given starting and end points, write a Python program to print all even numbers in that given range.

Example:

```
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.

Python3

```
# Python program to print all e
```

Python3

```
# Python program to print all e
for even_numbers in range(4,15,
  #here inside range function f
  #second denotes end and
  #third denotes the interval
    print(even_numbers,end=' ')
```

Output

4 6 8 10 12 14

Time Complexity: O(n)

Auxiliary Space: O(1)

Example #2: Taking range limit from user input

Python3

```
# Python program to print Even
start = int(input("Enter the st
end = int(input("Enter the end
# iterating each number in list
```

– Bandham Manikanta Dec 25, 2019 at 16:37

Add a comment

4

```
f = 0.5
a = 0
b = 9
d = [x * f for x in range(a, b)]
```

would be a way to do it.

Share Improve this answer Follow

answered Jul 13, 2011 at 18:27

marc

6,123 • 1 • 28 • 33

Add a comment

2

Numpy's `r_` convenience function can also create evenly spaced lists with syntax `np.r_[start:stop:steps]`. If `steps` is a real number (ending on `j`), then the end point is included, equivalent to `np.linspace(start, stop, step, endpoint=1)`, otherwise not.

```
>>> np.r_[-1:1:6j, [0]*3, 5, 6]
array([-1. , -0.6, -0.2,  0.2,
```

You can also directly concatente other arrays and also scalars:

14

Similar to unutbu's answer, you can use numpy's arange function, which is analog to Python's intrinsic function `range`. Notice that the end point is not included, as in `range`:

```
>>> import numpy as np
>>> a = np.arange(0,5, 0.5)
>>> a
array([ 0. ,  0.5,  1. ,  1.5,
>>> a = np.arange(0,5, 0.5) # re
>>> a.tolist() # if you prefer i
[0.0, 0.5, 1.0, 1.5, 2.0, 2.5, 3
```

Share Improve this answer Follow

edited May 23, 2018 at 15:16

answered Jul 13, 2011 at 19:46

milancurcic

6,233 • 2 • 34 • 47

Or simply: a = N.arange(0, 5 ,
0.5).tolist() – pyd Oct 30, 2014 at