Machine Learning

Chapter 1

Introduction to Machine Learning

- What is Machine Learning?
- Motivation: Why Machine Learning?
- Applications of Machine Learning
- Designing a Learning System
- Issues in Machine Learning

INTRODUCTION

Ever since computers were invented, we have wondered whether they might be made to learn. If we could understand how to program them to learn-to improve automatically with experience-the impact would be dramatic.

- Imagine computers learning from medical records which treatments are most effective for new diseases
- Houses learning from experience to optimize energy costs based on the particular usage patterns of their occupants.
- Personal software assistants learning the evolving interests of their users in order to highlight especially relevant stories from the online morning newspaper

A successful understanding of how to make computers learn would open up many new uses of computers and new levels of competence and customization

Some successful applications of machine learning

- Learning to recognize spoken words
- Learning to drive an autonomous vehicle
- Learning to classify new astronomical structures
- Learning to play world-class backgammon

Why is Machine Learning Important?

- Some tasks cannot be defined well, except by examples (e.g., recognizing

people).

- Relationships and correlations can be hidden within large amounts of data. Machine Learning/Data Mining may be able to find these relationships.
- Human designers often produce machines that do not work as well as desired in the environments in which they are used.
- The amount of knowledge available about certain tasks might be too large for explicit encoding by humans (e.g., medical diagnostic).
- Environments change over time.
- New knowledge about tasks is constantly being discovered by humans. It may be difficult to continuously re-design systems “by hand”.

Supervised Machine Learning

Supervised learning is the types of machine learning in which machines are trained using well "labelled" training data, and on basis of that data, machines predict the output. The labelled data means some input data is already tagged with the correct output.

In supervised learning, the training data provided to the machines work as the supervisor that teaches the machines to predict the output correctly. It applies the same concept as a student learns in the supervision of the teacher.

Supervised learning is a process of providing input data as well as correct output data to the machine learning model. The aim of a supervised learning algorithm is to **find a mapping function to map the input variable(x) with the output variable(y)**.

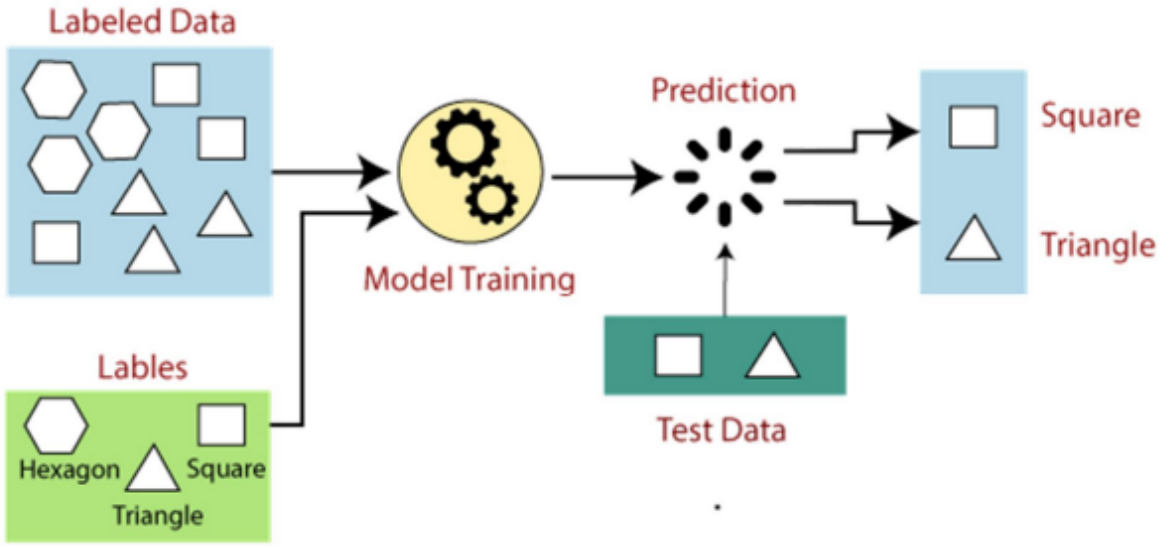
In the real-world, supervised learning can be used for **Risk Assessment, Image classification, Fraud Detection, spam filtering**, etc.

Difference between JDK, JRE, and JVM

How Supervised Learning Works?

In supervised learning, models are trained using labelled dataset, where the model learns about each type of data. Once the training process is completed, the model is tested on the basis of test data (a subset of the training set), and then it predicts the output.

The working of Supervised learning can be easily understood by the below example and diagram:



Suppose we have a dataset of different types of shapes which includes square, rectangle, triangle, and Polygon. Now the first step is that we need to train the model for each shape.

- If the given shape has four sides, and all the sides are equal, then it will be labelled as a **Square**.
- If the given shape has three sides, then it will be labelled as a **triangle**.
- If the given shape has six equal sides then it will be labelled as **hexagon**.

Now, after training, we test our model using the test set, and the task of the model is to identify the shape.

The machine is already trained on all types of shapes, and when it finds a new shape, it classifies the shape on the bases of a number of sides, and predicts the output.

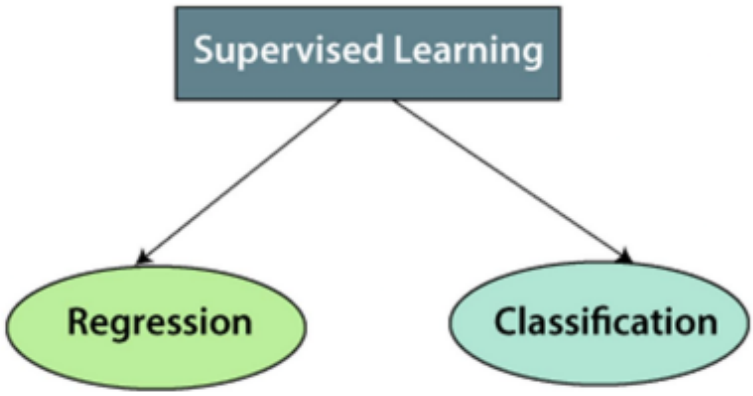
Steps Involved in Supervised Learning:

- First Determine the type of training dataset
- Collect/Gather the labelled training data.
- Split the training dataset into training **dataset, test dataset, and validation dataset**.
- Determine the input features of the training dataset, which should have enough knowledge so that the model can accurately predict the output.
- Determine the suitable algorithm for the model, such as support vector machine, decision tree, etc.

- Execute the algorithm on the training dataset. Sometimes we need validation sets as the control parameters, which are the subset of training datasets.
- Evaluate the accuracy of the model by providing the test set. If the model predicts the correct output, which means our model is accurate.

Types of supervised Machine learning Algorithms:

Supervised learning can be further divided into two types of problems:



1. Regression

Regression algorithms are used if there is a relationship between the input variable and the output variable. It is used for the prediction of continuous variables, such as Weather forecasting, Market Trends, etc. Below are some popular Regression algorithms which come under supervised learning:

- Linear Regression
- Regression Trees
- Non-Linear Regression
- Bayesian Linear Regression
- Polynomial Regression

2. Classification

Classification algorithms are used when the output variable is categorical, which means there are two classes such as Yes-No, Male-Female, True-false, etc.

Spam Filtering,

- Random Forest
- Decision Trees
- Logistic Regression
- Support vector Machines

Note: We will discuss these algorithms in detail in later chapters.

Advantages of Supervised learning:

- With the help of supervised learning, the model can predict the output on the basis of prior experiences.
- In supervised learning, we can have an exact idea about the classes of objects.
- Supervised learning model helps us to solve various real-world problems such as **fraud detection, spam filtering**, etc.

Disadvantages of supervised learning:

- Supervised learning models are not suitable for handling the complex tasks.
- Supervised learning cannot predict the correct output if the test data is different from the training dataset.
- Training required lots of computation times.
- In supervised learning, we need enough knowledge about the classes of object.

Unsupervised Machine Learning

In the previous topic, we learned supervised machine learning in which models are trained using labeled data under the supervision of training data. But there may be many cases in which we do not have labeled data and need to find the hidden patterns from the given dataset. So, to solve such types of cases in machine learning, we need unsupervised learning techniques.

What is Unsupervised Learning?

As the name suggests, unsupervised learning is a machine learning technique in which models are not supervised using training dataset. Instead, models itself find the hidden patterns and insights from the given data. It can be compared to learning which takes place in the human brain while learning new things. It can be defined as:

Unsupervised learning is a type of machine learning in which models are trained using unlabeled dataset and are allowed to act on that data without any supervision.

Unsupervised learning cannot be directly applied to a regression or classification problem because unlike supervised learning, we have the input data but no corresponding output data. The goal of unsupervised learning is to **find the underlying structure of dataset, group that data according to similarities, and represent that dataset in a compressed format**.

Example: Suppose the unsupervised learning algorithm is given an input dataset containing images of different types of cats and dogs. The algorithm is never trained upon the given dataset, which means it does not have any idea about the features of the dataset. The task of the unsupervised learning algorithm is to identify the image features on their own. Unsupervised learning algorithm will perform this task by clustering the image dataset into the groups according to similarities between images.

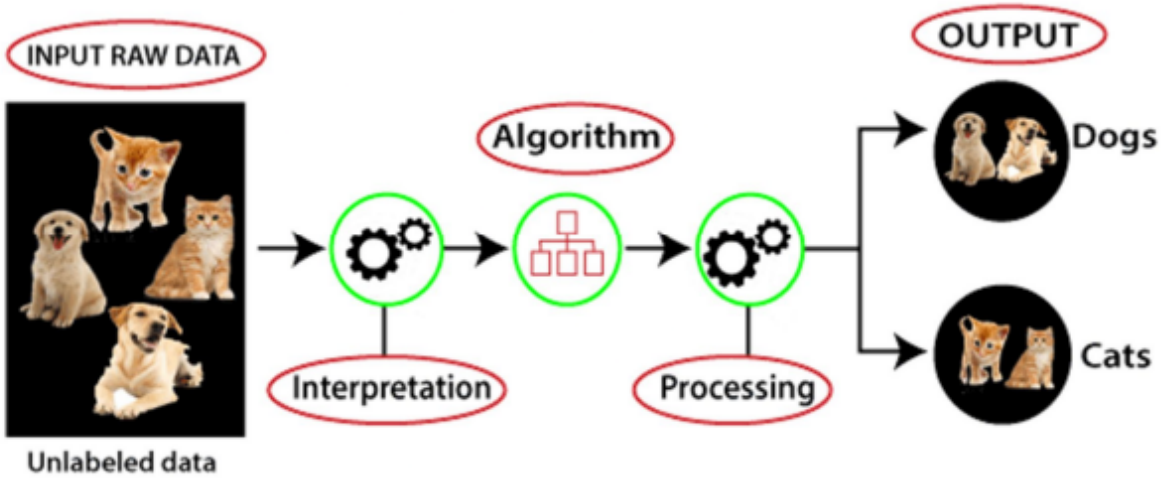
Why use Unsupervised Learning?

Below are some main reasons which describe the importance of Unsupervised Learning:

- Unsupervised learning is helpful for finding useful insights from the data.
- Unsupervised learning is much similar as a human learns to think by their own experiences, which makes it closer to the real AI.
- Unsupervised learning works on unlabeled and uncategorized data which make unsupervised learning more important.
- In real-world, we do not always have input data with the corresponding output so to solve such cases, we need unsupervised learning.

Working of Unsupervised Learning

Working of unsupervised learning can be understood by the below diagram:

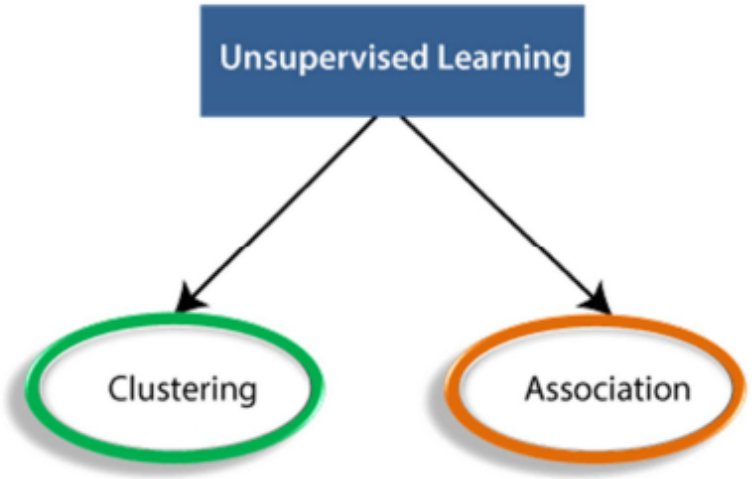


Here, we have taken an unlabeled input data, which means it is not categorized and corresponding outputs are also not given. Now, this unlabeled input data is fed to the machine learning model in order to train it. Firstly, it will interpret the raw data to find the hidden patterns from the data and then will apply suitable algorithms such as k-means clustering, Decision tree, etc.

Once it applies the suitable algorithm, the algorithm divides the data objects into groups according to the similarities and difference between the objects.

Types of Unsupervised Learning Algorithm:

The unsupervised learning algorithm can be further categorized into two types of problems:



- **Clustering**: Clustering is a method of grouping the objects into clusters such that objects with most similarities remains into a group and has less or no similarities with the objects of another group. Cluster analysis finds the commonalities between the data objects and categorizes them as per the presence and absence of those commonalities.
- **Association**: An association rule is an unsupervised learning method which is used for finding the relationships between variables in the large database. It determines the set of items that occurs together in the dataset. Association rule makes marketing strategy more effective. Such as people who buy X item (suppose a bread) are also tend to purchase Y (Butter/Jam) item. A typical example of Association rule is Market Basket Analysis.

Note: We will learn these algorithms in later chapters.

Unsupervised Learning algorithms:

Below is the list of some popular unsupervised learning algorithms:

- **K-means clustering**
- **KNN (k-nearest neighbors)**
- **Hierarchal clustering**

- **Anomaly detection**
- **Neural Networks**
- **Principle Component Analysis**
- **Independent Component Analysis**
- **Apriori algorithm**
- **Singular value decomposition**

Advantages of Unsupervised Learning

- Unsupervised learning is used for more complex tasks as compared to supervised learning because, in unsupervised learning, we don't have labeled input data.
- Unsupervised learning is preferable as it is easy to get unlabeled data in comparison to labeled data.

Disadvantages of Unsupervised Learning

- Unsupervised learning is intrinsically more difficult than supervised learning as it does not have corresponding output.
- The result of the unsupervised learning algorithm might be less accurate as input data is not labeled, and algorithms do not know the exact output in advance.

The main differences between Supervised and Unsupervised learning are given below:

Supervised Learning	Unsupervised Learning
Supervised learning algorithms are trained using labeled data.	Unsupervised learning algorithms are trained using unlabeled data.
Supervised learning model takes direct feedback to check if it is predicting correct output or not.	Unsupervised learning model does not take any feedback.
Supervised learning model predicts the output.	Unsupervised learning model finds the hidden patterns in data.

In supervised learning, input data is provided to the model along with the output.	In unsupervised learning, only input data is provided to the model.
The goal of supervised learning is to train the model so that it can predict the output when it is given new data.	The goal of unsupervised learning is to find the hidden patterns and useful insights from the unknown dataset.
Supervised learning needs supervision to train the model.	Unsupervised learning does not need any supervision to train the model.
Supervised learning can be categorized in **Classification** and **Regression** problems.	Unsupervised Learning can be classified in **Clustering** and **Associations** problems.
Supervised learning can be used for those cases where we know the input as well as corresponding outputs.	Unsupervised learning can be used for those cases where we have only input data and no corresponding output data.
Supervised learning model produces an accurate result.	Unsupervised learning model may give less accurate result as compared to supervised learning.
Supervised learning is not close to true Artificial intelligence as in this, we first train the model for each data, and then only it can predict the correct output.	Unsupervised learning is more close to the true Artificial Intelligence as it learns similarly as a child learns daily routine things by his experiences.
It includes various algorithms such as Linear Regression, Logistic Regression, Support Vector Machine, Multi-class Classification, Decision tree, Bayesian Logic, etc.	It includes various algorithms such as Clustering, KNN, and Apriori algorithm.

- *Note: The supervised and unsupervised learning both are the machine learning methods, and selection of any of these learning depends on the factors related to the structure and volume of your dataset and the use cases of the problem.*

WELL-POSED LEARNING PROBLEMS

Definition: A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.

To have a well-defined learning problem, three features needs to be identified:

1. The class of tasks
2. The measure of performance to be improved
3. The source of experience

Examples

1. ***Checkers game:*** A computer program that learns to play ***checkers*** might improve its performance as measured by its ability to win at the class of tasks involving playing checkers games, through experience obtained by playing games against itself.

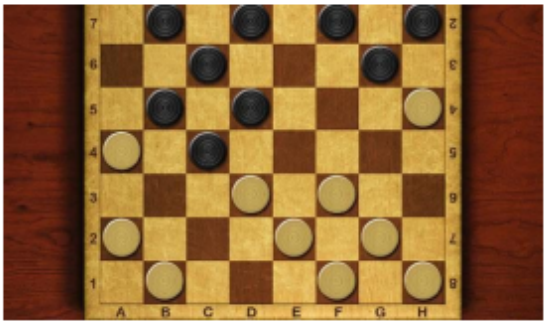


Fig: Checker game board

A checkers learning problem:

- Task T: playing checkers
- Performance measure P: percent of games won against opponents
- Training experience E: playing practice games against itself

2. *A handwriting recognition learning problem:*
    - Task T: recognizing and classifying handwritten words within images

- Performance measure P: percent of words correctly classified
- Training experience E: a database of handwritten words with given classific

3. *A robot driving learning problem*:
    - Task T: driving on public four-lane highways using vision sensors
    - Performance measure P: average distance travelled before an error (as judged by human overseer)
    - Training experience E: a sequence of images and steering commands recorded while observing a human driver

DESIGNING A LEARNING SYSTEM

The basic design issues and approaches to machine learning are illustrated by designing a program to learn to play checkers, with the goal of entering it in the world checkers tournament

1. Choosing the Training Experience
2. Choosing the Target Function
3. Choosing a Representation for the Target Function
4. Choosing a Function Approximation Algorithm
    1. Estimating training values
    2. Adjusting the weights
5. The Final Design

1. Choosing the Training Experience

- The first design choice is to choose the type of training experience from which the system will learn.
- The type of training experience available can have a significant impact on success or failure of the learner.

There are three attributes which impact on success or failure of the learner

1. Whether the training experience provides ***direct or indirect feedback*** regarding the choices made by the performance system.

    For example, in checkers game:

In learning to play checkers, the system might learn from ***direct training examples***

Consisting of ***individual checkers board states*** and ***the correct move for each***.

Indirect training examples consisting of the ***move sequences*** and ***final outcomes*** of various games played. The information about the correctness of specific moves early in the game must be inferred indirectly from the fact that the game was eventually won or lost.

Here the learner faces an additional problem of ***credit assignment***, or determining the degree to which each move in the sequence deserves credit or blame for the final outcome. Credit assignment can be a particularly difficult problem because the game can be lost even when early moves are optimal, if these are followed later by poor moves.
Hence, learning from direct training feedback is typically easier than learning from indirect feedback.

2. The degree to which the ***learner controls the sequence of training examples***

   <u>For example, in checkers game</u>:
   The learner might depends on the ***teacher*** to select informative board states and to provide the correct move for each.

   Alternatively, the learner might itself propose board states that it finds particularly confusing and ask the teacher for the correct move.

   The learner may have complete control over both the board states and (indirect) training classifications, as it does when it learns by playing against itself with ***no teacher present***.

3. How well it represents the ***distribution of examples*** over which the final system performance P must be measured

   <u>For example, in checkers game:</u>

In checkers learning scenario, the performance metric P is the percent of games the system wins in the world tournament.

If its training experience E consists only of games played against itself, there is a danger that this training experience might not be fully representative of the distribution of situations over which it will later be tested.
It is necessary to learn from a distribution of examples that is different from those on which the final system will be evaluated.

2. *Choosing the Target Function*

The next design choice is to determine exactly what type of knowledge will be learned and how this will be used by the performance program.

Let's consider a checkers-playing program that can generate the legal moves from any board state.
The program needs only to learn how to choose the best move from among these legal moves. We must learn to choose among the legal moves, the most obvious choice for the type of information to be learned is a program, or function, that chooses the best move for any given board state.

1. Let ***ChooseMove*** be the target function and the notation is

   $$ChooseMove : B \rightarrow M$$

   which indicate that this function accepts as input any board from the set of legal board states B and produces as output some move from the set of legal moves M.

   ChooseMove is a choice for the target function in checkers example, but this function will turn out to be very difficult to learn given the kind of indirect training experience available to our system

2. An alternative target function is an ***evaluation function*** that assigns a ***numerical score***
   to any given board state
   Let the target function V and the notation

$$V : B \rightarrow R$$

which denote that V maps any legal board state from the set B to some real value. Intend for this target function V to assign higher scores to better board states. If the system can successfully learn such a target function V, then it can easily use it to select the best move from any current board position.

Let us define the target value V(b) for an arbitrary board state b in B, as follows:

- If b is a final board state that is won, then V(b) = 100
- If b is a final board state that is lost, then V(b) = -100
- If b is a final board state that is drawn, then V(b) = 0
- If b is a not a final state in the game, then V(b) = V(b'),

Where b' is the best final board state that can be achieved starting from b and playing optimally until the end of the game

3. *Choosing a Representation for the Target Function*

Let's choose a simple representation - for any given board state, the function ***c*** will be calculated as a linear combination of the following board features:

- x_1: the number of black pieces on the board
- x_2: the number of red pieces on the board
- x_3: the number of black kings on the board
- x_4: the number of red kings on the board
- x_5: the number of black pieces threatened by red (i.e., which can be captured on red's next turn)
- x_6: the number of red pieces threatened by black

Thus, learning program will represent as a linear function of the form

$$\hat{V}(b) = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_3 + w_4 x_4 + w_5 x_5 + w_6 x_6$$

Where,

- w_0 through w_6 are numerical coefficients, or weights, to be chosen by the learning algorithm.

- Learned values for the weights w1 through w6 will determine the relative importance of the various board features in determining the value of the board
- The weight w_0 will provide an additive constant to the board value

4. *Choosing a Function Approximation Algorithm*

In order to learn the target function **f** we require a set of training examples, each describing a specific board state b and the training value $V_{train}(b)$ for b.

Each training example is an ordered pair of the form $(b, V_{train}(b))$.

For instance, the following training example describes a board state b in which black has won the game (note $x_2 = 0$ indicates that red has no remaining pieces) and for which the target function value $V_{train}(b)$ is therefore +100.

$$((x_1=3, x_2=0, x_3=1, x_4=0, x_5=0, x_6=0), +100)$$

Function Approximation Procedure

1. Derive training examples from the indirect training experience available to the learner
2. Adjusts the weights w_i to best fit these training examples

1. Estimating training values

   A simple approach for estimating training values for intermediate board states is to
   assign the training value of $V_{train}(b)$ for any intermediate board state b to be $\hat{V}$(Successor(b))

   Where ,
   - $\hat{V}$ is the learner's current approximation to V
   - Successor(b) denotes the next board state following b for which it is again the program's turn to move

Rule for estimating training values

$$V_{train}(b) \leftarrow \hat{V}(Successor(b))$$

2. Adjusting the weights

Specify the learning algorithm for choosing the weights w_i to best fit the set of training examples $\{(b, V_{train}(b))\}$

A first step is to define what we mean by the bestfit to the training data.

One common approach is to define the best hypothesis, or set of weights, as that which minimizes the squared error E between the training values and the values predicted by the hypothesis.

$$E \equiv \sum_{\langle b, V_{train}(b)\rangle \in \ training\ \ examples} (V_{train}(b) - \hat{V}(b))^2$$

Several algorithms are known for finding weights of a linear function that minimize E. One such algorithm is called the ***least mean squares, or LMS training rule***. For each observed training example it adjusts the weights a small amount in the direction that reduces the error on this training example

LMS weight update rule :- For each training example $(b, V_{train}(b))$

Use the current weights to calculate $\hat{V}(b)$
For each weight w_i, update it as

$$w_i \leftarrow w_i + \eta\ (V_{train}(b) - \hat{V}(b))\ x_i$$

Here η is a small constant (e.g., 0.1) that moderates the size of the weight update.

Working of weight update rule

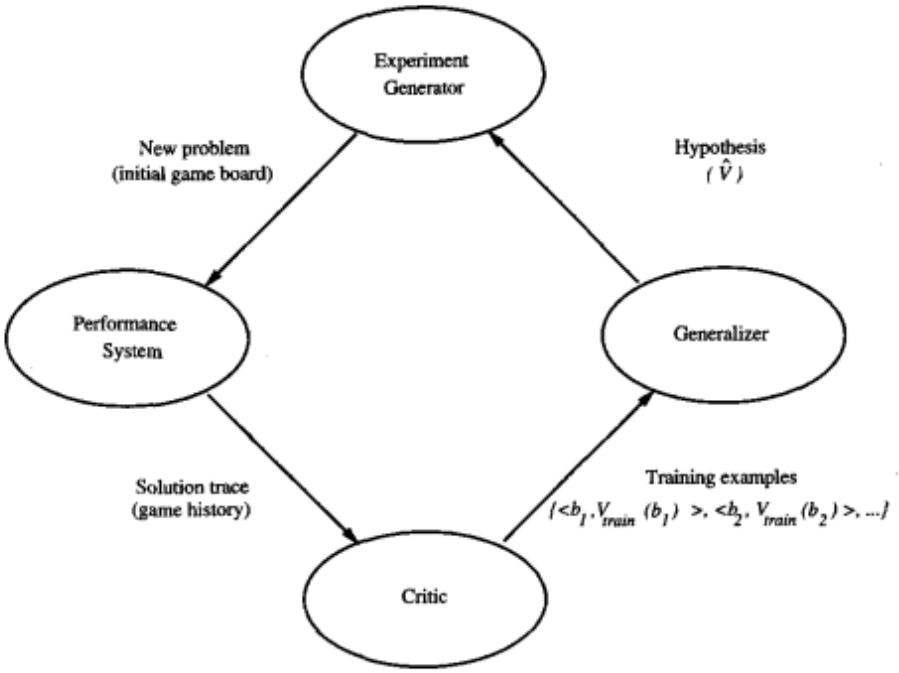
- When the error $(V_{train}(b) - \hat{V}(b))$ is zero, no weights are changed.
- When $(V_{train}(b) - \hat{V}(b))$ is positive (i.e., when $\hat{V}(b)$ is too low), then each

w_i is increased in proportion to the value of its corresponding feature. This will raise the value of $\hat{V}(b)$, reducing the error.

- If the value of some feature x_i is zero, then its weight is not altered regardless of the error, so that the only weights updated are those whose features actually occur on the training example board.

5. *The Final Design*

The final design of checkers learning system can be described by four distinct program modules that represent the central components in many learning systems



1. **The Performance System** is the module that must solve the given performance task by using the learned target function(s). It takes an instance of a new problem (new game) as input and produces a trace of its solution (game history) as output.

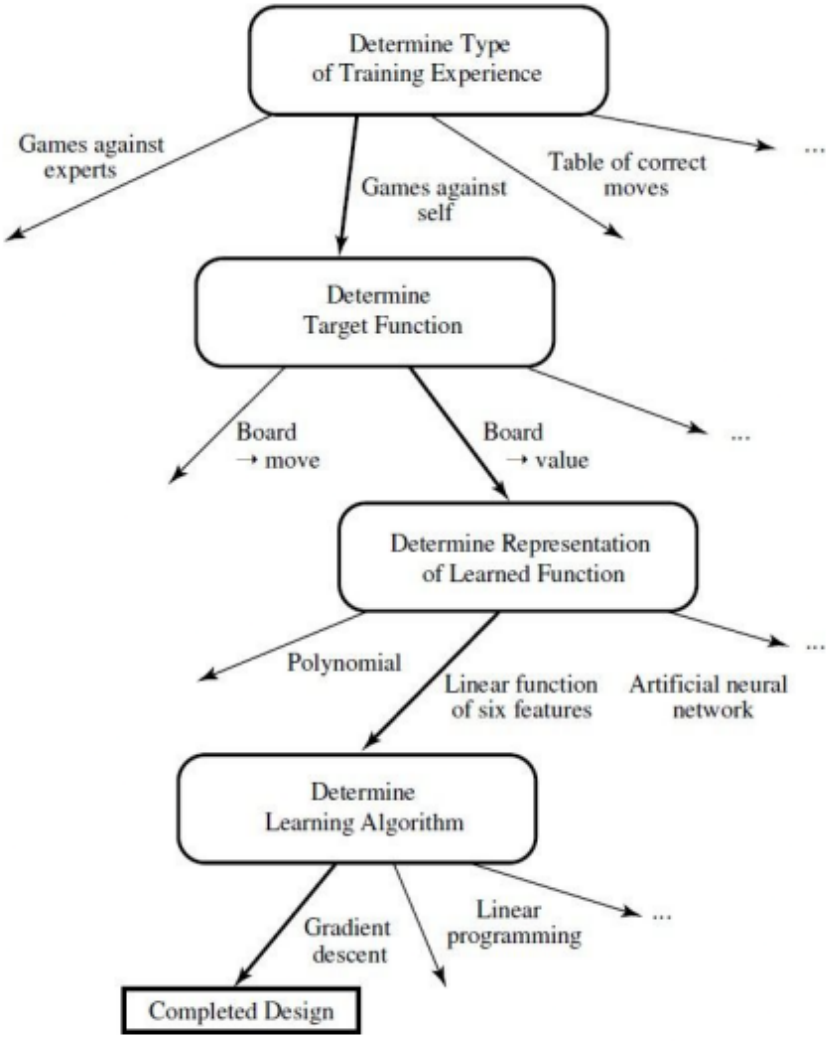
2. **The Critic** takes as input the history or trace of the game and produces as output a set of training examples of the target function

3. **The Generalizer** takes as input the training examples and produces an output

hypothesis that is its estimate of the target function. It generalizes from the specific training examples, hypothesizing a general function that covers these examples and other cases beyond the training examples.

4. **The Experiment Generator** takes as input the current hypothesis and outputs a new problem (i.e., initial board state) for the Performance System to explore. Its role is to pick new practice problems that will maximize the learning rate of the overall system.

The sequence of design choices made for the checkers program is summarized in below figure



PERSPECTIVES AND ISSUES IN MACHINE LEARNING

Issues in Machine Learning

The field of machine learning, and much of this book, is concerned with answering questions such as the following

- What algorithms exist for learning general target functions from specific training examples? In what settings will particular algorithms converge to the desired function, given sufficient training data? Which algorithms perform best for which types of problems and representations?
- How much training data is sufficient? What general bounds can be found to relate the confidence in learned hypotheses to the amount of training experience and the character of the learner's hypothesis space?
- When and how can prior knowledge held by the learner guide the process of generalizing from examples? Can prior knowledge be helpful even when it is only approximately correct?
- What is the best strategy for choosing a useful next training experience, and how does the choice of this strategy alter the complexity of the learning problem?
- What is the best way to reduce the learning task to one or more function approximation problems? Put another way, what specific functions should the system attempt to learn? Can this process itself be automated?
- How can the learner automatically alter its representation to improve its ability to represent and learn the target function?

Noise in Machine Learning

Real-world data, which is used to feed data mining algorithms, has a number of factors that can influence it. The existence of noise is a major factor in both of these problems. It's an inevitable problem, but one that a data-driven organization must fix.

Humans are prone to making mistakes when collecting data, and data collection instruments may be unreliable, resulting in dataset errors. The errors are referred to as noise. Data noise in machine learning can cause problems since the algorithm interprets the noise as a pattern and can start generalizing from it.

As a result, by using an algorithm, any data scientist must deal with the noise in data science. There are many widely used techniques used to extract the noise from any signal or dataset.

Data Preprocessing in Machine learning

Data preprocessing is a process of preparing the raw data and making it suitable for a machine learning model. It is the first and crucial step while creating a machine learning model.

When creating a machine learning project, it is not always a case that we come across the clean and formatted data. And while doing any operation with data, it is mandatory to clean it and put in a formatted way. So for this, we use data preprocessing task.

Why do we need Data Preprocessing?

A real-world data generally contains noises, missing values, and maybe in an unusable format which cannot be directly used for machine learning models. Data preprocessing is required tasks for cleaning the data and making it suitable for a machine learning model which also increases the accuracy and efficiency of a machine learning model.

It involves below steps:

- **Getting the dataset**
- **Importing libraries**
- **Importing datasets**
- **Finding Missing Data**
- **Encoding Categorical Data**
- **Splitting dataset into training and test set**
- **Feature scaling**

1) Get the Dataset

To create a machine learning model, the first thing we required is a dataset as a machine learning model completely works on data. The collected data for a particular problem in a proper format is known as the **dataset**.

Dataset may be of different formats for different purposes, such as, if we want to create a machine learning model for business purpose, then dataset will be different with the dataset required for a liver patient. So each dataset is different from another dataset. To use the dataset in our code, we usually put it into a CSV **file**. However, sometimes, we may also need to use an HTML or xlsx file.

What is a CSV File?

CSV stands for "**Comma-Separated Values**" files; it is a file format which allows us to save the tabular data, such as spreadsheets. It is useful for huge datasets and can use these datasets in programs.

We can also create our dataset by gathering data using various API with Python and put that data into a .csv file.

2) Importing Libraries

In order to perform data preprocessing using Python, we need to import some predefined Python libraries. These libraries are used to perform some specific jobs. There are three specific libraries that we will use for data preprocessing, which are:

Numpy: Numpy Python library is used for including any type of mathematical operation in the code. It is the fundamental package for scientific calculation in Python. It also supports to add large, multidimensional arrays and matrices. So, in Python, we can import it as:

```
import numpy as nm
```

Here we have used **nm**, which is a short name for Numpy, and it will be used in the whole program.

Matplotlib: The second library is **matplotlib**, which is a Python 2D plotting library, and with this library, we need to import a sub-library **pyplot**. This library is used to plot any type of charts in Python for the code. It will be imported as below:

```
import matplotlib.pyplot as mpt
```

Here we have used mpt as a short name for this library.

Pandas: The last library is the Pandas library, which is one of the most famous Python libraries and used for importing and managing the datasets. It is an open-source data manipulation and analysis library. It will be imported as below:

Here, we have used pd as a short name for this library. Consider the below image:

```
1 # importing libraries
2 import numpy as nm
3 import matplotlib.pyplot as mtp
4 import pandas as pd
5
```

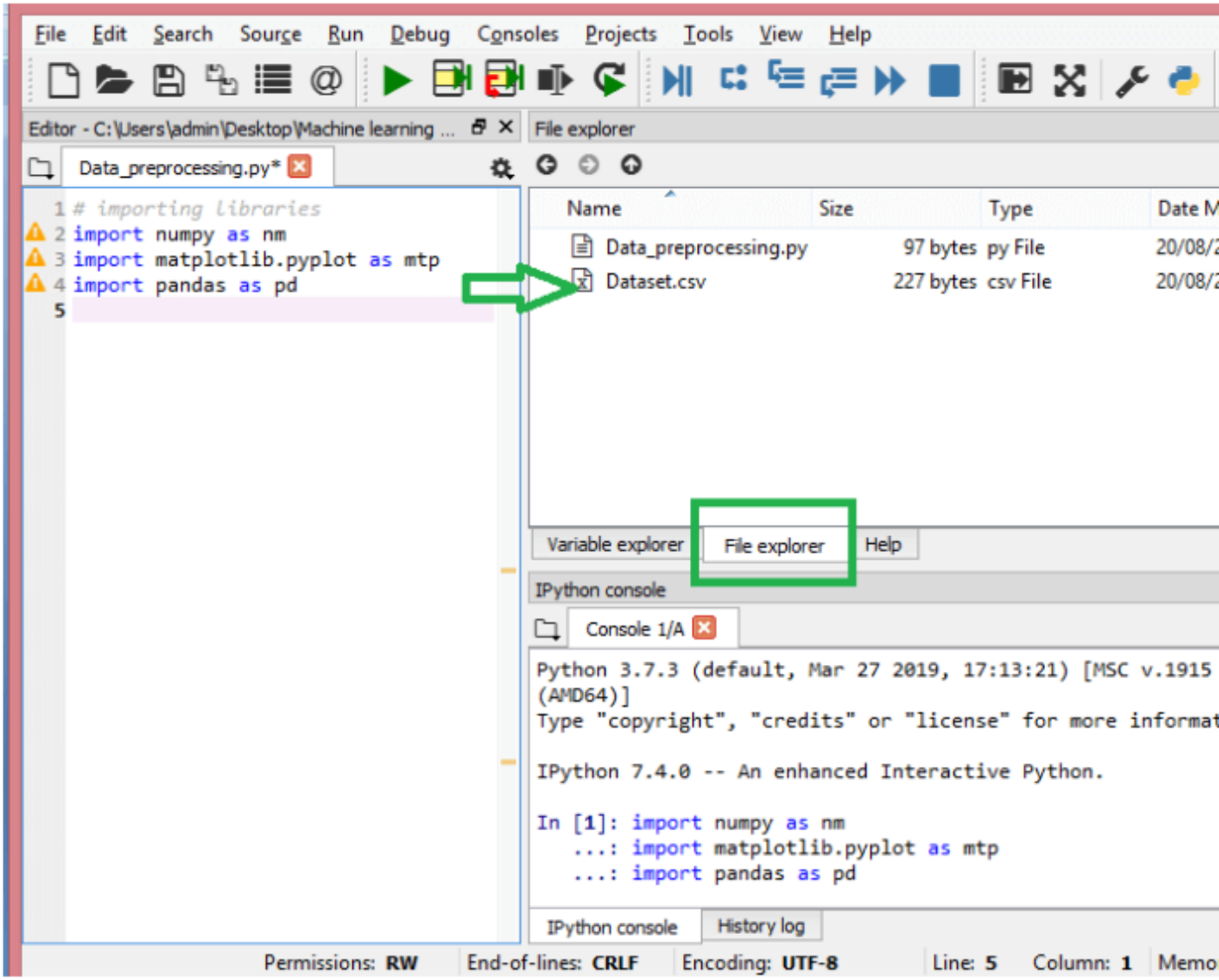
3) Importing the Datasets

Now we need to import the datasets which we have collected for our machine learning project. But before importing a dataset, we need to set the current directory as a working directory. To set a working directory in Spyder IDE, we need to follow the below steps:

1. Save your Python file in the directory which contains dataset.
2. Go to File explorer option in Spyder IDE, and select the required directory.
3. Click on F5 button or run option to execute the file.

Note: *We can set any directory as a working directory, but it must contain the required dataset.*

Here, in the below image, we can see the Python file along with required dataset. Now, the current folder is set as a working directory.



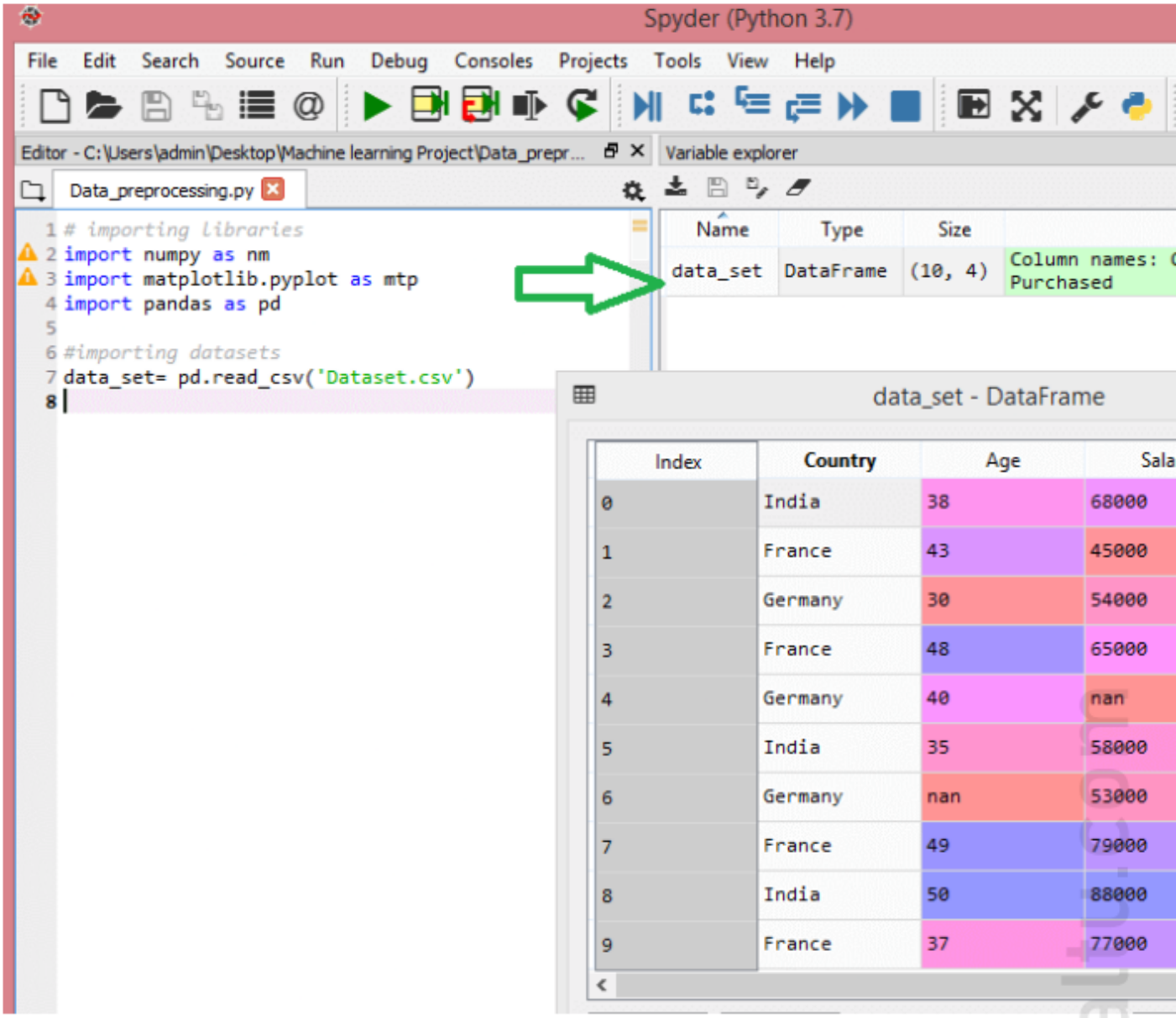
read_csv() function:

Now to import the dataset, we will use read_csv() function of pandas library, which is used to read a csv file and performs various operations on it. Using this function, we can read a csv file locally as well as through an URL.

We can use read_csv function as below:

1. data_set= pd.read_csv('Dataset.csv')

Here, **data_set** is a name of the variable to store our dataset, and inside the function, we have passed the name of our dataset. Once we execute the above line of code, it will successfully import the dataset in our code. We can also check the imported dataset by clicking on the section **variable explorer**, and then double click on **data_set**. Consider the below image:



As in the above image, indexing is started from 0, which is the default indexing in Python. We can also change the format of our dataset by clicking on the format option.

Extracting dependent and independent variables:

In machine learning, it is important to distinguish the matrix of features (independent variables) and dependent variables from dataset. In our dataset, there are three independent variables that are **Country, Age**, and **Salary**, and one is a dependent variable which is **Purchased**.

Extracting independent variable:

To extract an independent variable, we will use **iloc[ ]** method of Pandas library. It is used to extract the required rows and columns from the dataset.

```
x= data_set.iloc[:,:-1].values
```

In the above code, the first colon(:) is used to take all the rows, and the second colon(:) is for all the columns. Here we have used :-1, because we don't want to take the last column as it contains the dependent variable. So by doing this, we will get the matrix of features.

By executing the above code, we will get output as:

```
1. [['India' 38.0 68000.0]
2.  ['France' 43.0 45000.0]
3.  ['Germany' 30.0 54000.0]
4.  ['France' 48.0 65000.0]
5.  ['Germany' 40.0 nan]
6.  ['India' 35.0 58000.0]
7.  ['Germany' nan 53000.0]
8.  ['France' 49.0 79000.0]
9.  ['India' 50.0 88000.0]
10. ['France' 37.0 77000.0]]
```

As we can see in the above output, there are only three variables.

Extracting dependent variable:

To extract dependent variables, again, we will use Pandas .iloc[] method.

1. y= data_set.iloc[:,3].values

   Here we have taken all the rows with the last column only. It will give the array of dependent variables.

   By executing the above code, we will get output as:

   Output:

```
array(['No', 'Yes', 'No', 'No', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes'],
      dtype=object)
```

Note: *If you are using Python language for machine learning, then extraction is mandatory, but for R language it is not required.*

4) Handling Missing data:

The next step of data preprocessing is to handle missing data in the datasets. If our dataset contains some missing data, then it may create a huge problem for our machine learning model. Hence it is necessary to handle missing values present in the dataset.

Ways to handle missing data:

There are mainly two ways to handle missing data, which are:

By deleting the particular row: The first way is used to commonly deal with null values. In this way, we just delete the specific row or column which consists of null values. But this way is not so efficient and removing data may lead to loss of information which will not give the accurate output.

By calculating the mean: In this way, we will calculate the mean of that column or row which contains any missing value and will put it on the place of missing value. This strategy is useful for the features which have numeric data such as age, salary, year, etc. Here, we will use this approach.

To handle missing values, we will use **Scikit-learn** library in our code, which contains various libraries for building machine learning models. Here we will use **Imputer** class of **sklearn.preprocessing** library. Below is the code for it:

```
#handling missing data (Replacing missing data with the mean value)
from sklearn.preprocessing import Imputer
imputer= Imputer(missing_values ='NaN', strategy='mean', axis = 0)
#Fitting imputer object to the independent variables x.
imputerimputer= imputer.fit(x[:, 1:3])
#Replacing missing data with the calculated mean value
x[:, 1:3]= imputer.transform(x[:, 1:3])
```

Output:

```
array([['India', 38.0, 68000.0],
       ['France', 43.0, 45000.0],
       ['Germany', 30.0, 54000.0],
       ['France', 48.0, 65000.0],
       ['Germany', 40.0, 65222.22222222222],
       ['India', 35.0, 58000.0],
       ['Germany', 41.111111111111114, 53000.0],
       ['France', 49.0, 79000.0],
       ['India', 50.0, 88000.0],
       ['France', 37.0, 77000.0]], dtype=object
```

As we can see in the above output, the missing values have been replaced with the means of rest column values.

5) Encoding Categorical data:

Categorical data is data which has some categories such as, in our dataset; there are two categorical variable, **Country**, and **Purchased**.

Since machine learning model completely works on mathematics and numbers, but if our dataset would have a categorical variable, then it may create trouble while building the model. So it is necessary to encode these categorical variables into numbers.

For Country variable:

Firstly, we will convert the country variables into categorical data. So to do this, we will use **LabelEncoder()** class from **preprocessing** library.

```
#Catgorical data
#for Country Variable
from sklearn.preprocessing import LabelEncoder
label_encoder_x= LabelEncoder()
x[:, 0]= label_encoder_x.fit_transform(x[:, 0])
```

Output:

```
Out[15]:
 array([[2, 38.0, 68000.0],
        [0, 43.0, 45000.0],
       [1, 30.0, 54000.0],
       [0, 48.0, 65000.0],
       [1, 40.0, 65222.22222222222],
       [2, 35.0, 58000.0],
       [1, 41.111111111111114, 53000.0],
       [0, 49.0, 79000.0],
       [2, 50.0, 88000.0],
      [0, 37.0, 77000.0]], dtype=object)
```

Explanation:

In above code, we have imported **LabelEncoder** class of **sklearn library**. This class has successfully encoded the variables into digits.

But in our case, there are three country variables, and as we can see in the above output, these variables are encoded into 0, 1, and 2. By these values, the machine learning model may assume that there is some correlation between these variables which will produce the wrong output. So to remove this issue, we will use **dummy encoding**.

Dummy Variables:

Dummy variables are those variables which have values 0 or 1. The 1 value gives the presence of that variable in a particular column, and rest variables become 0. With dummy encoding, we will have a number of columns equal to the number of categories.

In our dataset, we have 3 categories so it will produce three columns having 0 and 1 values. For Dummy Encoding, we will use **OneHotEncoder** class of **preprocessing** library.

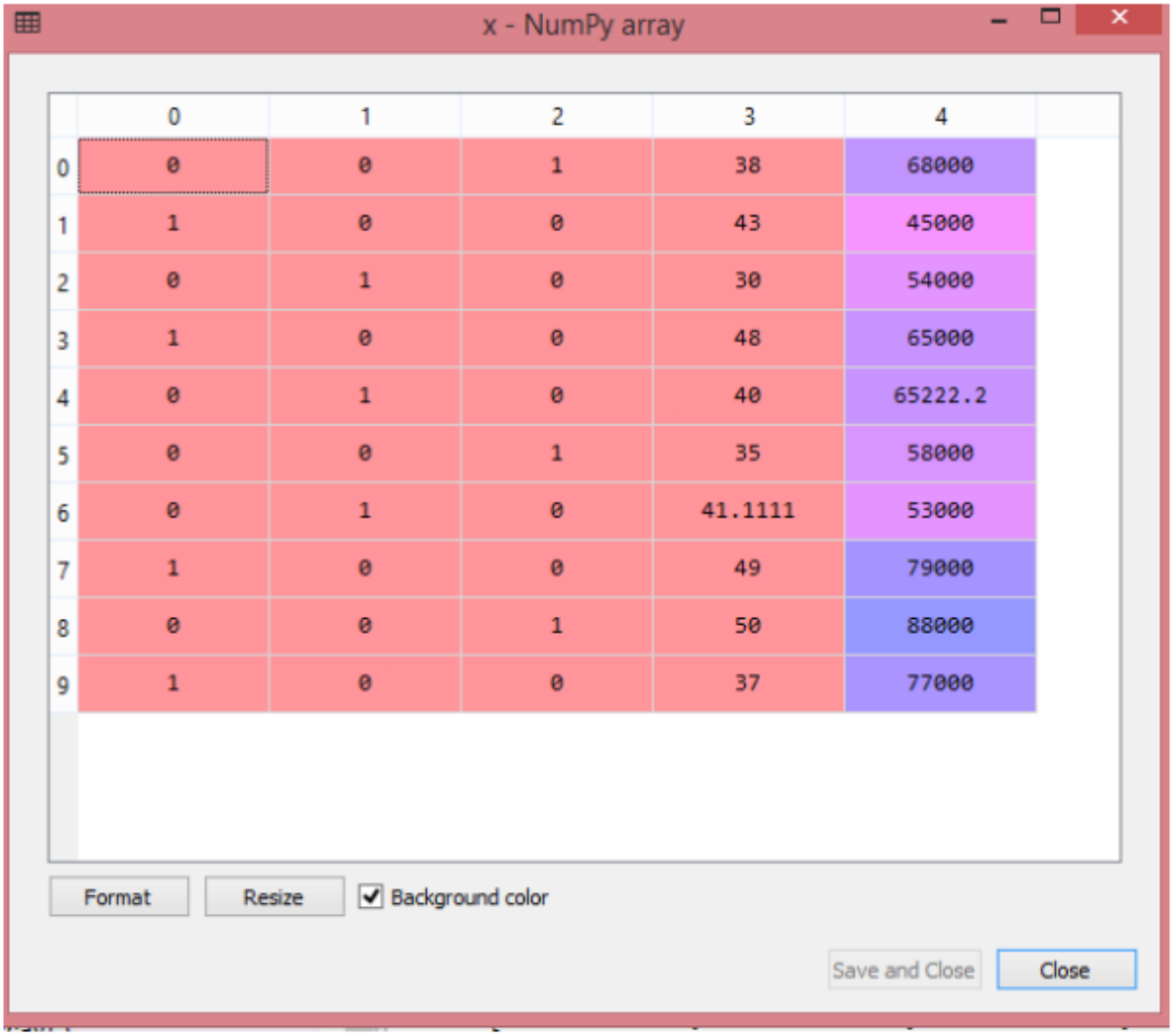
```
#for Country Variable
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
label_encoder_x= LabelEncoder()
x[:, 0]= label_encoder_x.fit_transform(x[:, 0])
#Encoding for dummy variables
onehot_encoder= OneHotEncoder(categorical_features= [0])
x= onehot_encoder.fit_transform(x).toarray()
```

Output:

```
array([[0.00000000e+00, 0.00000000e+00, 1.00000000e+00, 3.80000000e+01,
        6.80000000e+04],
       [1.00000000e+00, 0.00000000e+00, 0.00000000e+00, 4.30000000e+01,
        4.50000000e+04],
       [0.00000000e+00, 1.00000000e+00, 0.00000000e+00, 3.00000000e+01,
        5.40000000e+04],
       [1.00000000e+00, 0.00000000e+00, 0.00000000e+00, 4.80000000e+01,
        6.50000000e+04],
       [0.00000000e+00, 1.00000000e+00, 0.00000000e+00, 4.00000000e+01,
        6.52222222e+04],
       [0.00000000e+00, 0.00000000e+00, 1.00000000e+00, 3.50000000e+01,
        5.80000000e+04],
       [0.00000000e+00, 1.00000000e+00, 0.00000000e+00, 4.11111111e+01,
        5.30000000e+04],
       [1.00000000e+00, 0.00000000e+00, 0.00000000e+00, 4.90000000e+01,
        7.90000000e+04],
       [0.00000000e+00, 0.00000000e+00, 1.00000000e+00, 5.00000000e+01,
        8.80000000e+04],
       [1.00000000e+00, 0.00000000e+00, 0.00000000e+00, 3.70000000e+01,
        7.70000000e+04]])
```

As we can see in the above output, all the variables are encoded into numbers 0 and 1 and divided into three columns.

It can be seen more clearly in the variables explorer section, by clicking on x option as:



	0	1	2	3	4
0	0	0	1	38	68000
1	1	0	0	43	45000
2	0	1	0	30	54000
3	1	0	0	48	65000
4	0	1	0	40	65222.2
5	0	0	1	35	58000
6	0	1	0	41.1111	53000
7	1	0	0	49	79000
8	0	0	1	50	88000
9	1	0	0	37	77000

For Purchased Variable:

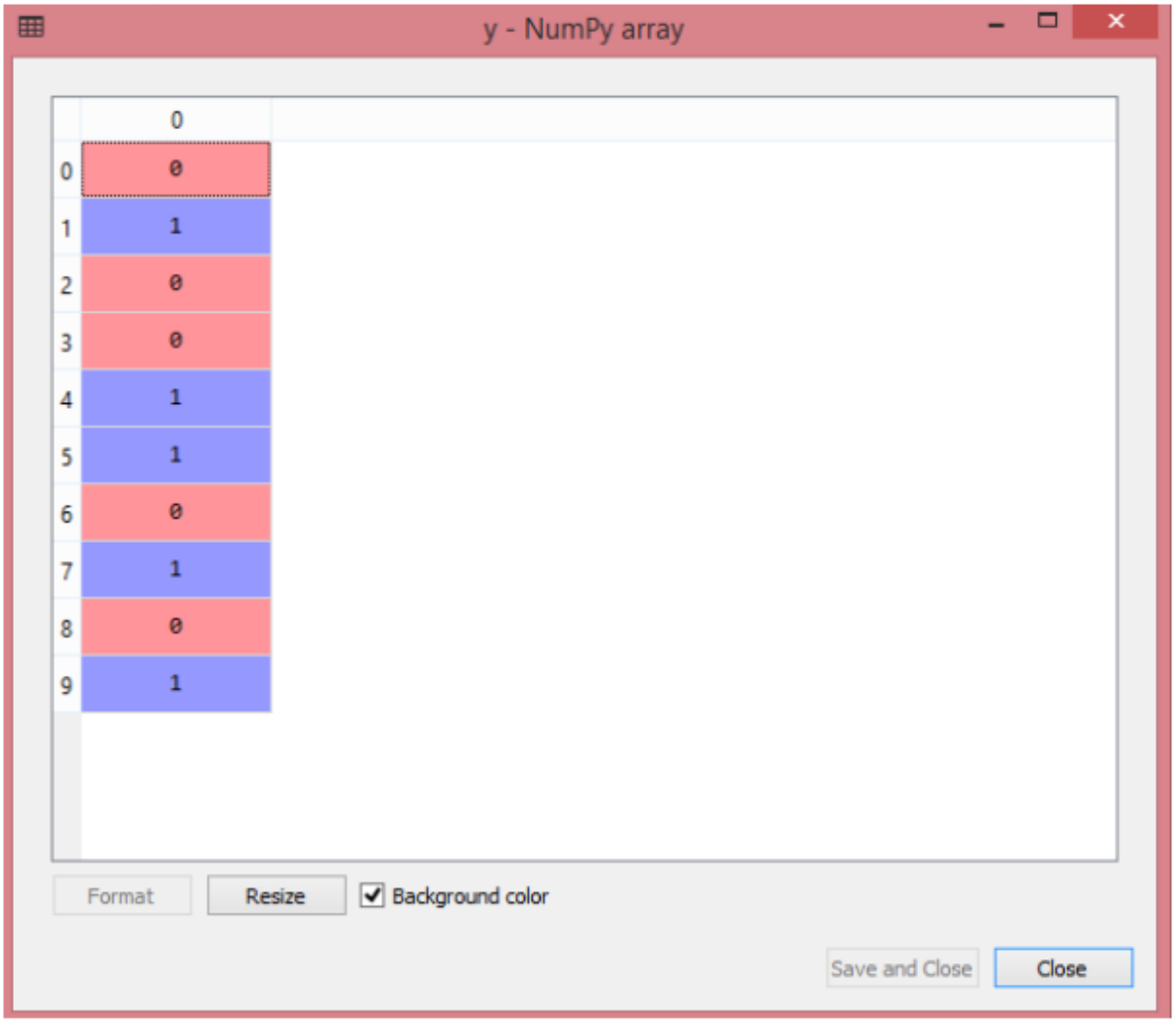
1. labelencoder_y= LabelEncoder()
2. y= labelencoder_y.fit_transform(y)

For the second categorical variable, we will only use labelencoder object of **LableEncoder** class. Here we are not using **OneHotEncoder** class because the purchased variable has only two categories yes or no, and which are automatically encoded into 0 and 1.

Output:

```
Out[17]: array([0, 1, 0, 0, 1, 1, 0, 1, 0, 1])
```

It can also be seen as:

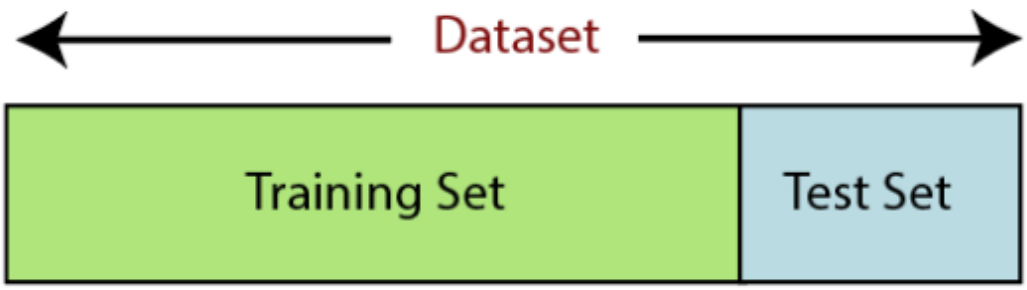


6) Splitting the Dataset into the Training set and Test set

In machine learning data preprocessing, we divide our dataset into a training set and test set. This is one of the crucial steps of data preprocessing as by doing this, we can enhance the performance of our machine learning model.

Suppose, if we have given training to our machine learning model by a dataset and we test it by a completely different dataset. Then, it will create difficulties for our model to understand the correlations between the models.

If we train our model very well and its training accuracy is also very high, but we provide a new dataset to it, then it will decrease the performance. So we always try to make a machine learning model which performs well with the training set and also with the test dataset. Here, we can define these datasets as:



Training Set: A subset of dataset to train the machine learning model, and we already know the output.

Test set: A subset of dataset to test the machine learning model, and by using the test set, model predicts the output.

For splitting the dataset, we will use the below lines of code:

1. from sklearn.model_selection import train_test_split
2. x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.2, random_state=0)

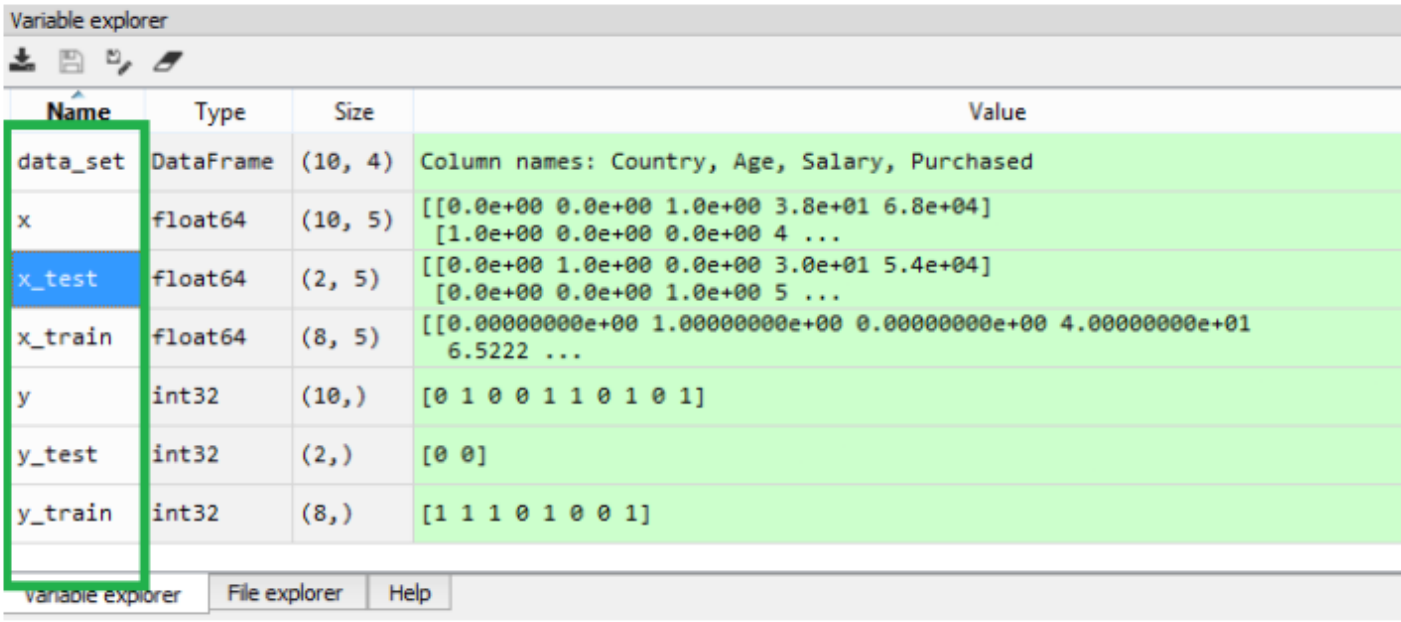
Explanation:

- In the above code, the first line is used for splitting arrays of the dataset into random train and test subsets.
- In the second line, we have used four variables for our output that are
  - **x_train:** features for the training data
  - **x_test:** features for testing data
  - **y_train:** Dependent variables for training data
  - **y_test:** Independent variable for testing data

- In **train_test_split() function**, we have passed four parameters in which first two are for arrays of data, and **test_size** is for specifying the size of the test set. The test_size maybe .5, .3, or .2, which tells the dividing ratio of training and testing sets.
- The last parameter **random_state** is used to set a seed for a random generator so that you always get the same result, and the most used value for this is 42.

Output:

By executing the above code, we will get 4 different variables, which can be seen under the variable explorer section.

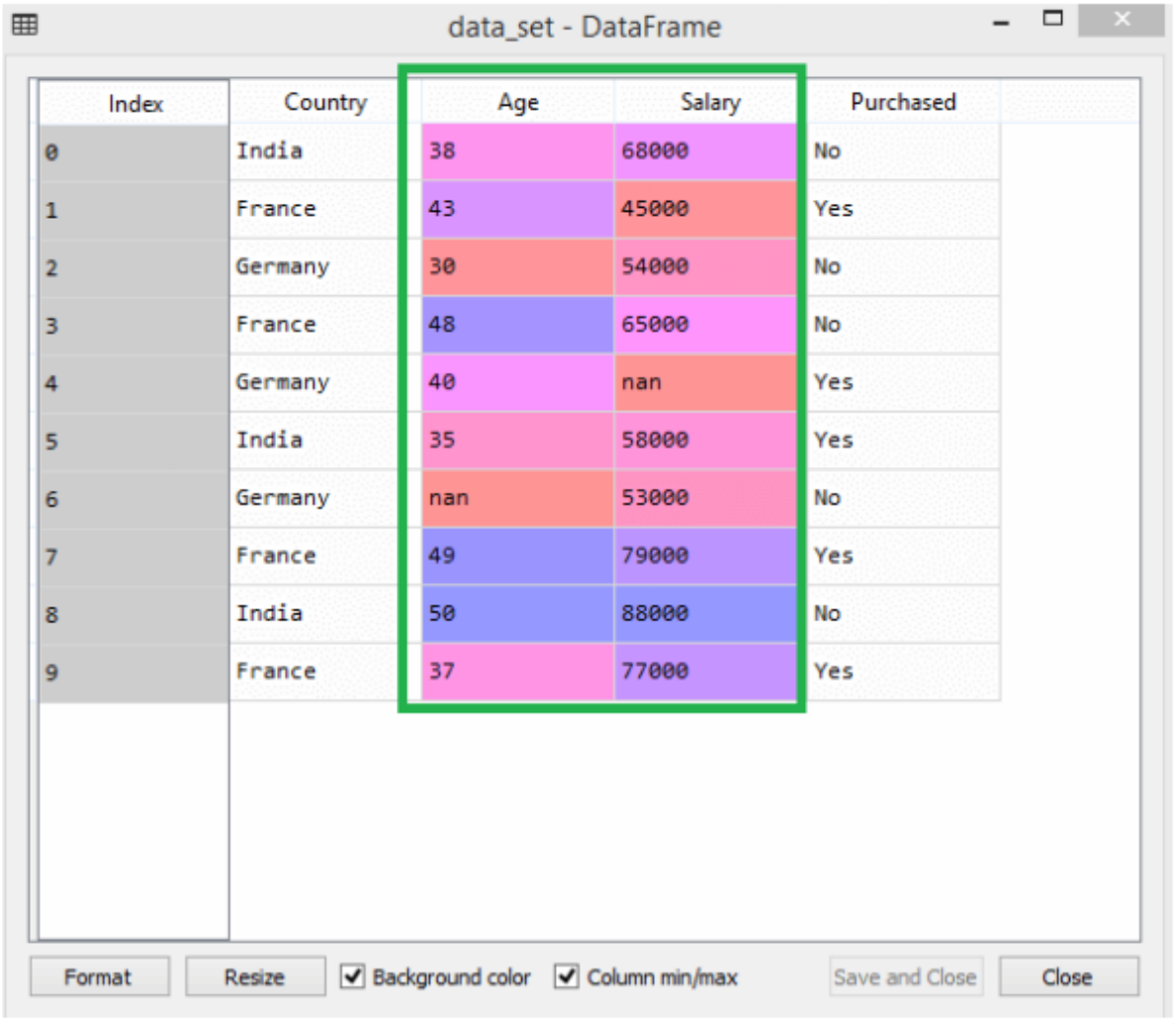


As we can see in the above image, the x and y variables are divided into 4 different variables with corresponding values.

7) Feature Scaling

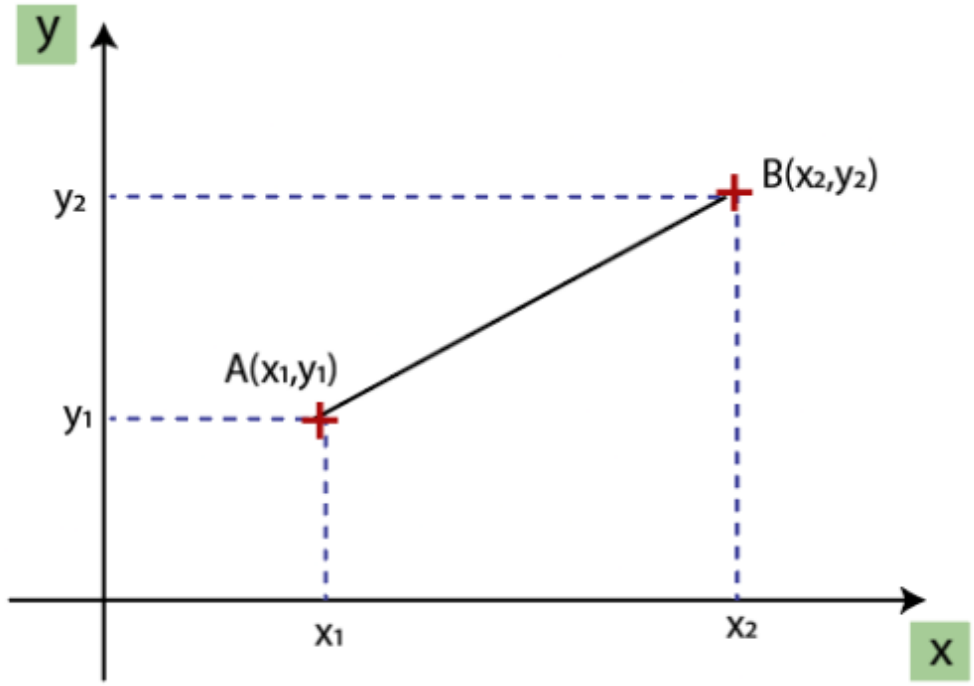
Feature scaling is the final step of data preprocessing in machine learning. It is a technique to standardize the independent variables of the dataset in a specific range. In feature scaling, we put our variables in the same range and in the same scale so that no any variable dominate the other variable.

Consider the below dataset:



As we can see, the age and salary column values are not on the same scale. A machine learning model is based on **Euclidean distance**, and if we do not scale the variable, then it will cause some issue in our machine learning model.

Euclidean distance is given as:

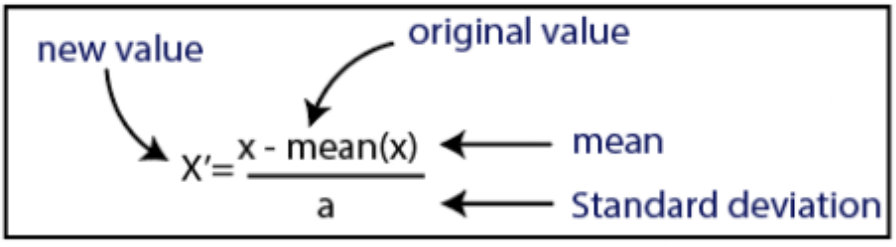


Euclidean Distance Between A and B= $\sqrt{(x_2-x_1)^2+(y_2-y_1)^2}$

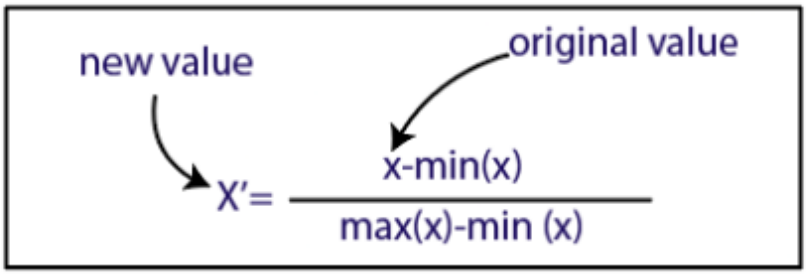
If we compute any two values from age and salary, then salary values will dominate the age values, and it will produce an incorrect result. So to remove this issue, we need to perform feature scaling for machine learning.

There are two ways to perform feature scaling in machine learning:

Standardization



Normalization



Here, we will use the standardization method for our dataset.

For feature scaling, we will import ***StandardScaler*** class of ***sklearn.preprocessing*** library as:

```
1. from sklearn.preprocessing import StandardScaler
```

Now, we will create the object of **StandardScaler** class for independent variables or features. And then we will fit and transform the training dataset.

```
1. st_x= StandardScaler()
2. x_train= st_x.fit_transform(x_train)
```

For test dataset, we will directly apply **transform()** function instead of **fit_transform()** because it is already done in training set.

1. x_test= st_x.transform(x_test)

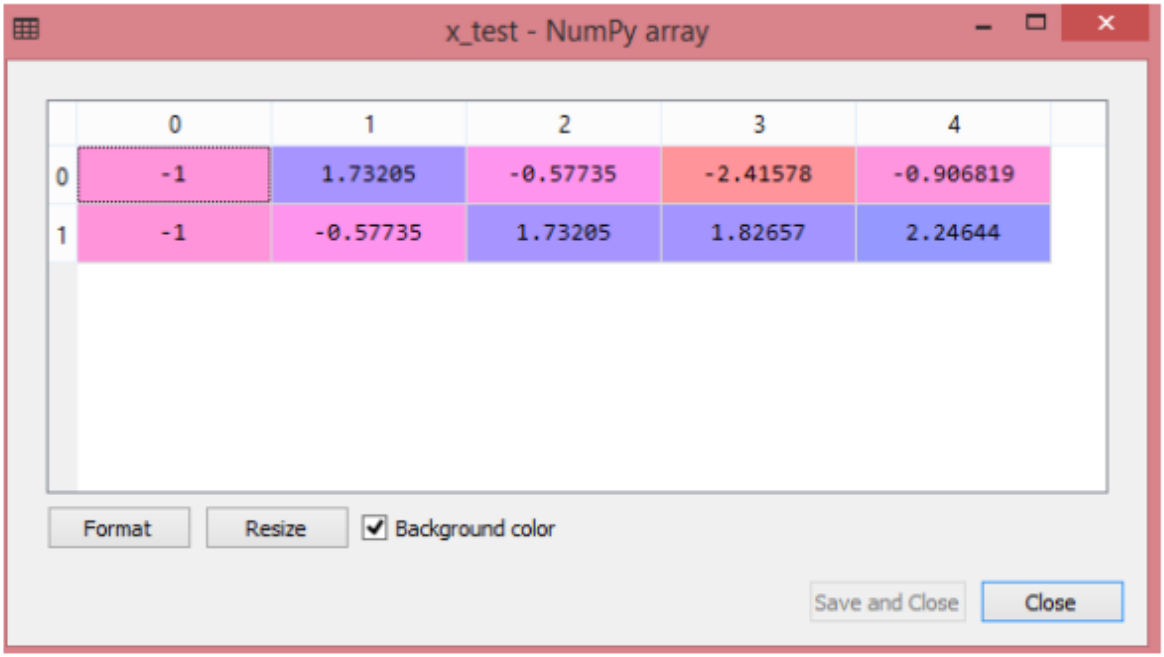
Output:

By executing the above lines of code, we will get the scaled values for x_train and x_test as:

x_train:

	0	1	2	3	4
0	-1	1.73205	-0.57735	-0.294607	0.133962
1	1	-0.57735	-0.57735	-0.930959	1.22627
2	1	-0.57735	-0.57735	0.341745	-1.7415
3	-1	1.73205	-0.57735	-0.0589215	-0.999562
4	1	-0.57735	-0.57735	1.61445	1.41175
5	1	-0.57735	-0.57735	1.40233	0.113352
6	-1	-0.57735	1.73205	-0.718842	0.391581
7	-1	-0.57735	1.73205	-1.35519	-0.535848

x_test:



As we can see in the above output, all the variables are scaled between values -1 to 1.

Note: *Here, we have not scaled the dependent variable because there are only two values 0 and 1. But if these variables will have more range of values, then we will also need to scale those variables.*

Combining all the steps:

Now, in the end, we can combine all the steps together to make our complete code more understandable.

```
# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd

#importing datasets
data_set= pd.read_csv('Dataset.csv')

#Extracting Independent Variable
x= data_set.iloc[:, :-1].values

#Extracting Dependent variable
```

```
y= data_set.iloc[:, 3].values

#handling missing data(Replacing missing data with the mean value)
from sklearn.preprocessing import Imputer
imputer= Imputer(missing_values ='NaN', strategy='mean', axis = 0)

#Fitting imputer object to the independent varibles x.
imputerimputer= imputer.fit(x[:, 1:3])

#Replacing missing data with the calculated mean value
x[:, 1:3]= imputer.transform(x[:, 1:3])

#for Country Variable
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
label_encoder_x= LabelEncoder()
x[:, 0]= label_encoder_x.fit_transform(x[:, 0])

#Encoding for dummy variables
onehot_encoder= OneHotEncoder(categorical_features= [0])
x= onehot_encoder.fit_transform(x).toarray()

#encoding for purchased variable
labelencoder_y= LabelEncoder()
y= labelencoder_y.fit_transform(y)
  # Splitting the dataset into training and test set.
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.2, random_state=0)

#Feature Scaling of datasets
from sklearn.preprocessing import StandardScaler
st_x= StandardScaler()
x_train= st_x.fit_transform(x_train)
x_test= st_x.transform(x_test)
```

In the above code, we have included all the data preprocessing steps together. But there are some steps or lines of code which are not necessary for all machine learning models. So we can exclude them from our code to make it reusable for all models.

Training Validation and Test data

- **Training data.** This type of data builds up the machine learning algorithm. The data scientist feeds the algorithm input data, which corresponds to an expected output. The model evaluates the data repeatedly to learn more about the data's behavior and then adjusts itself to serve its intended purpose.
- **Validation data.** During training, validation data infuses new data into the model that it hasn't evaluated before. Validation data provides the first test against unseen data, allowing data scientists to evaluate how well the model makes predictions based on the new data. Not all data scientists use validation data, but it can provide some helpful information to optimize hyperparameters, which influence how the model assesses data.
- **Test data.** After the model is built, testing data once again validates that it can make accurate predictions. If training and validation data include labels to monitor performance metrics of the model, the testing data should be unlabeled. Test data provides a final, real-world check of an unseen dataset to confirm that the ML algorithm was trained effectively.

While each of these three datasets has its place in creating and training ML models, it's easy to see some overlap between them. The difference between training data vs. test data is clear: one trains a model, the other confirms it works correctly,

Bias and Variance

The prediction error for any machine learning algorithm can be broken down into two parts:

- Bias Error
- Variance Error

Bias Error

Bias are the simplifying assumptions made by a model to make the target function easier to learn.

Generally, linear algorithms have a high bias making them fast to learn and easier to understand but generally less flexible. In turn, they have lower predictive performance on complex problems that fail to meet the simplifying assumptions of the algorithms bias.

- **Low Bias**: Suggests less assumptions about the form of the target function.
- **High-Bias**: Suggests more assumptions about the form of the target function.

  Examples of **low-bias** machine learning algorithms include: Decision Trees, k-Nearest Neighbors and Support Vector Machines.

  Examples of **high-bias** machine learning algorithms include: Linear Regression, Linear Discriminant Analysis and Logistic Regression.

Variance Error

Variance is the amount that the estimate of the target function will change if different training data was used.

The target function is estimated from the training data by a machine learning algorithm, so we should expect the algorithm to have some variance. Ideally, it should not change too much from one training dataset to the next, meaning that the algorithm is good at picking out the hidden underlying mapping between the inputs and the output variables.

Machine learning algorithms that have a high variance are strongly influenced by the specifics of the training data. This means that the specifics of the training have influences the number and types of parameters used to characterize the mapping function.

- **Low Variance**: Suggests small changes to the estimate of the target function with changes to the training dataset.

- **High Variance**: Suggests large changes to the estimate of the target function with changes to the training dataset.
  Generally, nonlinear machine learning algorithms that have a lot of flexibility have a high variance.
  For example, decision trees have a high variance, that is even higher if the trees are not pruned before use.

  Examples of **low-variance** machine learning algorithms include: Linear Regression, Linear Discriminant Analysis and Logistic Regression.
  Examples of **high-variance** machine learning algorithms include: Decision Trees, k-Nearest Neighbors and Support Vector Machines.

Bias-Variance Trade-Off

The goal of any supervised machine learning algorithm is to achieve low bias and low variance. In turn the algorithm should achieve good prediction performance.

You can see a general trend in the examples above:

- **Linear** machine learning algorithms often have a high bias but a low variance.
- **Nonlinear** machine learning algorithms often have a low bias but a high variance.
  The parameterization of machine learning algorithms is often a battle to balance out bias and variance.

Below are two examples of configuring the bias-variance trade-off for specific algorithms:

- The k-nearest neighbors algorithm has low bias and high variance, but the trade-off can be changed by increasing the value of k which increases the number of neighbors that contribute t the prediction and in turn increases the bias of the model.
- The support vector machine algorithm has low bias and high variance, but the trade-off can be changed by increasing the C parameter that influences the number of violations of the margin allowed in the training data which increases the bias but decreases the variance.
  There is no escaping the relationship between bias and variance in machine learning.

- Increasing the bias will decrease the variance.
- Increasing the variance will decrease the bias.
  There is a trade-off at play between these two concerns and the algorithms you choose and the way you choose to configure them are finding different balances in this trade-off for your problem

In reality, we cannot calculate the real bias and variance error terms because we do not know the actual underlying target function. Nevertheless, as a framework, bias and variance provide the tools

to understand the behavior of machine learning algorithms in the pursuit of predictive performance.

Summary

- Bias is the simplifying assumptions made by the model to make the target function easier to approximate.
- Variance is the amount that the estimate of the target function will change given different training data.

Chapter 2

Introduction to supervised Learning

Classification Problems

Linear Regression

Decision tree representation

Support vector machine (SVM)

Classification

In machine learning, classification refers to a predictive modeling problem where a class label is predicted for a given example of input data.

- The goal of classification is to use an object's characteristics to identify which class (or group) it belongs to.
- A linear classifier achieves this by making a classification decision based on the value of a linear combination of the characteristics.
- An object's characteristics are also known as feature values and are typically presented to the machine in a vector called a feature vector.
- Such classifiers work well for practical problems such as document classification, and more generally for problems with many variables (features), reaching accuracy levels comparable to non-linear classifiers while taking less time to train and use.

Examples of classification problems include:

- Given an example, classify if it is spam or not.
- Given a handwritten character, classify it as one of the known characters.
- Given recent user behavior, classify as churn or not.

From a modeling perspective, classification requires a training dataset with many examples of inputs and outputs from which to learn.

A model will use the training dataset and will calculate how to best map examples of input data to specific class labels. As such, the training dataset must be sufficiently representative of the problem and have many examples of each class label.

Class labels are often string values, e.g. "*spam*," "*not spam*," and must be mapped to numeric values before being provided to an algorithm for modeling. This is often referred to as label encoding, where a unique integer is assigned to each class label, e.g. "*spam*" = 0, "*no spam*" = 1. There are many different types of classification algorithms for modeling classification predictive modeling problems.

There is no good theory on how to map algorithms onto problem types; instead, it is generally recommended that a practitioner use controlled experiments and discover which algorithm and algorithm configuration results in the best performance for a given classification task.

Classification predictive modeling algorithms are evaluated based on their results. Classification accuracy is a popular metric used to evaluate the performance of a model based on the predicted class labels. Classification accuracy is not perfect but is a good starting point for many classification tasks.
Instead of class labels, some tasks may require the prediction of a probability of class membership for each example. This provides additional uncertainty in the prediction that an application or user can then interpret. A popular diagnostic for evaluating predicted probabilities is the ROC Curve.

What is the Classification Algorithm?

The Classification algorithm is a Supervised Learning technique that is used to identify the category of new observations on the basis of training data. In Classification, a program learns from the given dataset or observations and then classifies new observation into a number of classes or groups. Such as, **Yes or No, 0 or 1, Spam or Not Spam, cat or dog,** etc. Classes can be called as targets/labels or categories.

Unlike regression, the output variable of Classification is a category, not a value, such as "Green or Blue", "fruit or animal", etc. Since the Classification algorithm is a Supervised learning technique, hence it takes labeled input data, which means it contains input with the corresponding output.

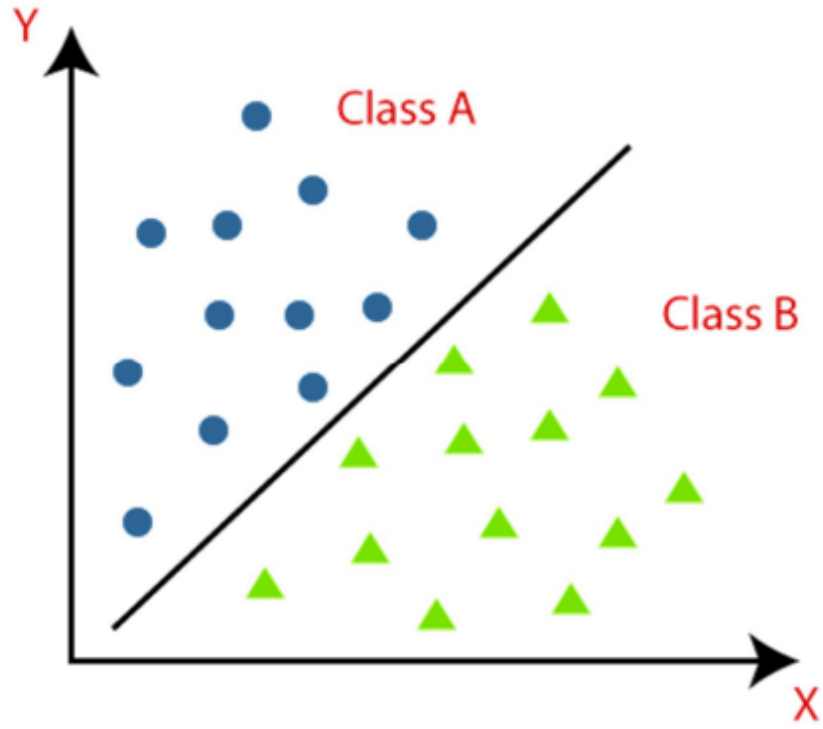
In classification algorithm, a discrete output function(y) is mapped to input variable(x).

y=f(x), where y = categorical output

The best example of an ML classification algorithm is **Email Spam Detector**.

The main goal of the Classification algorithm is to identify the category of a given dataset, and these algorithms are mainly used to predict the output for the categorical data.

Classification algorithms can be better understood using the below diagram. In the below diagram, there are two classes, class A and Class B. These classes have features that are similar to each other and dissimilar to other classes.



The algorithm which implements the classification on a dataset is known as a classifier. There are two types of Classifications:

- **Binary Classifier:** If the classification problem has only two possible outcomes, then it is called as Binary Classifier.
  Examples: YES or NO, MALE or FEMALE, SPAM or NOT SPAM, CAT or DOG, etc.
- **Multi-class Classifier:** If a classification problem has more than two outcomes, then it is called as Multi-class Classifier.
  Example: Classifications of types of crops, Classification of types of music.

Learners in Classification Problems:

In the classification problems, there are two types of learners:

Lazy Learners: Lazy Learner firstly stores the training dataset and wait until it receives the test dataset. In Lazy learner case, classification is done on the basis of the most related data stored in the training dataset. It takes less time in training but more time for predictions.
Example: K-NN algorithm, Case-based reasoning

1. **Eager Learners:** Eager Learners develop a classification model based on a training dataset before receiving a test dataset. Opposite to Lazy learners, Eager Learner takes more time in learning, and less time in prediction. **Example:** Decision Trees, Naïve Bayes, ANN.

Types of ML Classification Algorithms:

Classification Algorithms can be further divided into the Mainly two category:

- **Linear Models**
  - Logistic Regression
  - Support Vector Machines
- **Non-linear Models**
  - K-Nearest Neighbours
  - Kernel SVM
  - Naïve Bayes
  - Decision Tree Classification
  - Random Forest Classification

Use cases of Classification Algorithms

Classification algorithms can be used in different places. Below are some popular use cases of Classification Algorithms:

- Email Spam Detection
- Speech Recognition
- Identifications of Cancer tumor cells.
- Drugs Classification
- Biometric Identification, etc.

Regression Analysis in Machine learning

Regression analysis is a statistical method to model the relationship between a dependent (target) and independent (predictor) variables with one or more independent variables. More specifically, Regression analysis helps us to understand how the value of the dependent variable is changing corresponding to an independent variable when other independent variables are held fixed. It predicts continuous/real values such as **temperature, age, salary, price,** etc.

We can understand the concept of regression analysis using the below example:

Example: Suppose there is a marketing company A, who does various advertisement every year and get sales on that. The below list shows the advertisement made by the company in the last 5 years and the corresponding sales:

Advertisement	Sales
$90	$1000
$120	$1300
$150	$1800
$100	$1200
$130	$1380
$200	??

Now, the company wants to do the advertisement of $200 in the year 2019 **and wants to know the prediction about the sales for this year**. So to solve such type of prediction problems in machine learning, we need regression analysis.

C++ vs Java

Regression is a supervised learning technique which helps in finding the correlation between variables and enables us to predict the continuous output variable based on the one or more predictor variables. It is mainly used for **prediction, forecasting, time series modeling, and determining the causal-effect relationship between variables**.

In Regression, we plot a graph between the variables which best fits the given datapoints, using this plot, the machine learning model can make predictions about the data. In simple words, ***"Regression shows a line or curve that passes through all the datapoints on target-predictor graph in such a way that the vertical distance between the datapoints and the regression line is minimum."*** The distance between datapoints and line tells whether a model has captured a strong relationship or not.

Some examples of regression can be as:

- Prediction of rain using temperature and other factors
- Determining Market trends
- Prediction of road accidents due to rash driving.

Terminologies Related to the Regression Analysis:

- **Dependent Variable:** The main factor in Regression analysis which we want to predict or understand is called the dependent variable. It is also called **target variable**.
- **Independent Variable:** The factors which affect the dependent variables or which are used to predict the values of the dependent variables are called independent variable, also called as a **predictor**.
- **Outliers:** Outlier is an observation which contains either very low value or very high value in comparison to other observed values. An outlier may hamper the result, so it should be avoided.
- **Multicollinearity:** If the independent variables are highly correlated with each other than other variables, then such condition is called Multicollinearity. It should not be present in the dataset, because it creates problem while ranking the most affecting variable.
- **Underfitting and Overfitting:** If our algorithm works well with the training dataset but not well with test dataset, then such problem is called **Overfitting**. And if our algorithm does not perform well even with training dataset, then such problem is called **underfitting**.

Why do we use Regression Analysis?

As mentioned above, Regression analysis helps in the prediction of a continuous variable. There are various scenarios in the real world where we need some future predictions such as weather condition, sales prediction, marketing trends, etc., for such case we need some technology which can make predictions more accurately. So for such case we need Regression analysis which is a statistical method and used in machine learning and data science. Below are some other reasons for using Regression analysis:

- Regression estimates the relationship between the target and the independent variable.
- It is used to find the trends in data.
- It helps to predict real/continuous values.
- By performing the regression, we can confidently determine the **most important factor, the least important factor, and how each factor is affecting the other factors**.

Types of Regression

There are various types of regressions which are used in data science and machine learning. Each type has its own importance on different scenarios, but at the core, all the regression methods analyze the effect of the independent variable on dependent variables. Here we are discussing some important types of regression which are given below:

- **Linear Regression**
- **Logistic Regression**
- **Polynomial Regression**
- **Support Vector Regression**
- **Decision Tree Regression**
- **Random Forest Regression**
- **Ridge Regression**
- **Lasso Regression:**

➢ Naïve Bayes Classifier Algorithm

- Naïve Bayes algorithm is a supervised learning algorithm, which is based on **Bayes theorem** and used for solving classification problems.
- It is mainly used in *text classification* that includes a high-dimensional training dataset.
- Naïve Bayes Classifier is one of the simple and most effective Classification algorithms which helps in building the fast machine learning models that can make quick predictions.
- **It is a probabilistic classifier, which means it predicts on the basis of the probability of an object**.
- Some popular examples of Naïve Bayes Algorithm are **spam filtration, Sentimental analysis, and classifying articles**.

Why is it called Naïve Bayes?

The Naïve Bayes algorithm is comprised of two words Naïve and Bayes, Which can be described as:

- **Naïve**: It is called Naïve because it assumes that the occurrence of a certain feature is independent of the occurrence of other features. Such as if the fruit is identified on the bases of color, shape, and taste, then red, spherical, and sweet fruit is recognized as an apple. Hence each feature individually contributes to identify that it is an apple without depending on each other.
- **Bayes**: It is called Bayes because it depends on the principle of Bayes' Theorem

.

Bayes' Theorem:

- Bayes' theorem is also known as **Bayes' Rule** or **Bayes' law**, which is used to determine the probability of a hypothesis with prior knowledge. It depends on the conditional probability.
- The formula for Bayes' theorem is given as:

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

Where,

P(A|B) is Posterior probability: Probability of hypothesis A on the observed event B.

P(B|A) is Likelihood probability: Probability of the evidence given that the probability of a hypothesis is true.

Woman Trying to Surf With Wave Machine Falls Over Backwards and Faceplants to the Water

P(A) is Prior Probability: Probability of hypothesis before observing the evidence.

P(B) is Marginal Probability: Probability of Evidence.

Working of Naïve Bayes' Classifier:

Working of Naïve Bayes' Classifier can be understood with the help of the below example:

Suppose we have a dataset of **weather conditions** and corresponding target variable "**Play**". So using this dataset we need to decide that whether we should play or not on a particular day according to the weather conditions. So to solve this problem, we need to follow the below steps:

1. Convert the given dataset into frequency tables.
2. Generate Likelihood table by finding the probabilities of given features.
3. Now, use Bayes theorem to calculate the posterior probability.

Problem: If the weather is sunny, then the Player should play or not?

Solution: To solve this, first consider the below dataset:

	Outlook	Play
0	Rainy	Yes
1	Sunny	Yes
2	Overcast	Yes
3	Overcast	Yes
4	Sunny	No
5	Rainy	Yes
6	Sunny	Yes
7	Overcast	Yes

8	Rainy	No
9	Sunny	No
10	Sunny	Yes
11	Rainy	No
12	Overcast	Yes
13	Overcast	Yes

Frequency table for the Weather Conditions:

Weather	Yes	No
Overcast	5	0
Rainy	2	2
Sunny	3	2
Total	10	4

Likelihood table weather condition:

Weather	No	Yes	
Overcast	0	5	5/14= 0.35
Rainy	2	2	4/14=0.29
Sunny	2	3	5/14=0.35
All	4/14=0.29	10/14=0.71	

Applying Bayes'theorem:

P(Yes|Sunny)= P(Sunny|Yes)*P(Yes)/P(Sunny)

P(Sunny|Yes)= 3/10= 0.3

P(Sunny)= 0.35

P(Yes)=0.71

So P(Yes|Sunny) = 0.3*0.71/0.35= **0.60**

P(No|Sunny)= P(Sunny|No)*P(No)/P(Sunny)

P(Sunny|NO)= 2/4=0.5

P(No)= 0.29

P(Sunny)= 0.35

So P(No|Sunny)= 0.5*0.29/0.35 = **0.41**

So as we can see from the above calculation that **P(Yes|Sunny)>P(No|Sunny)**

Hence on a Sunny day, Player can play the game.

Advantages of Naïve Bayes Classifier:

- Naïve Bayes is one of the fast and easy ML algorithms to predict a class of datasets.
- It can be used for Binary as well as Multi-class Classifications.
- It performs well in Multi-class predictions as compared to the other Algorithms.
- It is the most popular choice for **text classification problems**.

Disadvantages of Naïve Bayes Classifier:

- Naive Bayes assumes that all features are independent or unrelated, so it cannot learn the relationship between features.

Applications of Naïve Bayes Classifier:

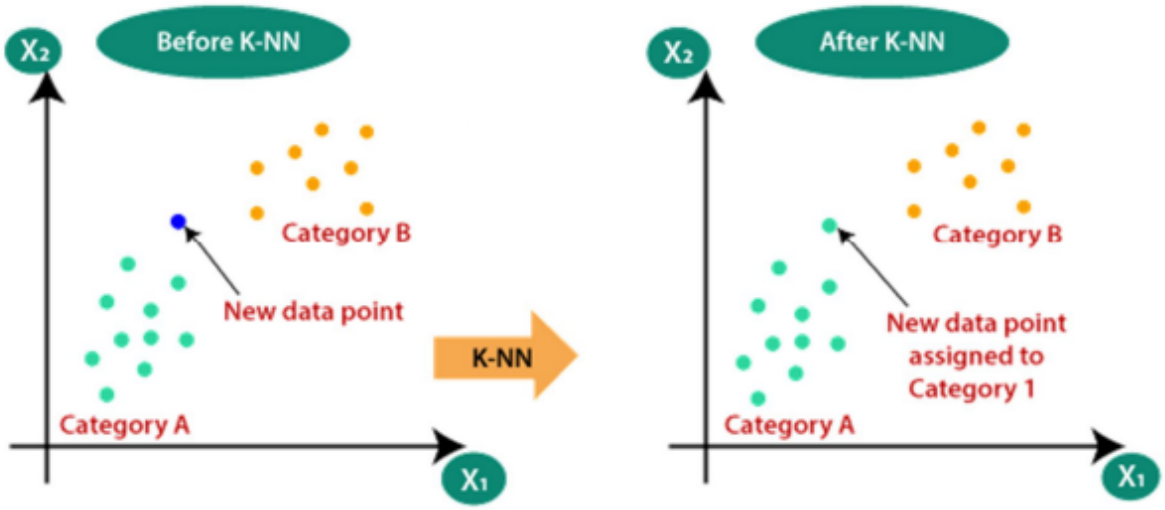
- It is used for **Credit Scoring**.
- It is used in **medical data classification**.
- It can be used in **real-time predictions** because Naïve Bayes Classifier is an eager learner.
- It is used in Text classification such as **Spam filtering** and **Sentiment analysis**.

K-Nearest Neighbor(KNN) Algorithm for Machine Learning

- K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique.
- K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories.
- K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm.
- K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems.
- K-NN is a **non-parametric algorithm**, which means it does not make any assumption on underlying data.
- It is also called a **lazy learner algorithm** because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset.
- KNN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.
- **Example:** Suppose, we have an image of a creature that looks similar to cat and dog, but we want to know either it is a cat or dog. So for this identification, we can use the KNN algorithm, as it works on a similarity measure. Our KNN model will find the similar features of the new data set to the cats and dogs images and based on the most similar features it will put it in either cat or dog category.

Why do we need a K-NN Algorithm?

Suppose there are two categories, i.e., Category A and Category B, and we have a new data point x1, so this data point will lie in which of these categories. To solve this type of problem, we need a K-NN algorithm. With the help of K-NN, we can easily identify the category or class of a particular dataset. Consider the below diagram:

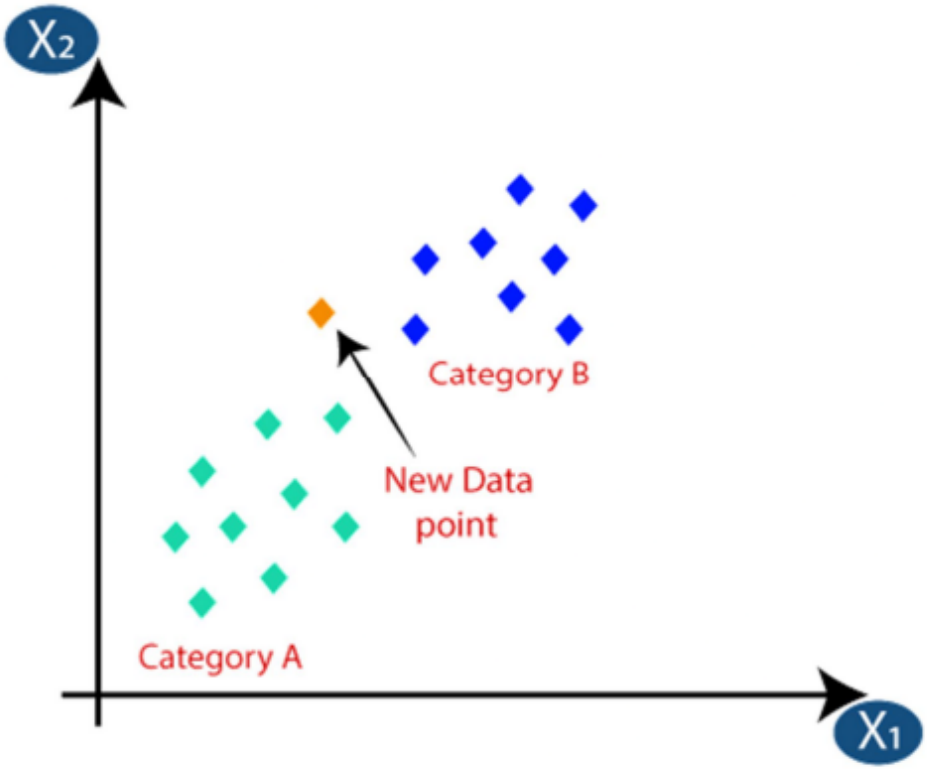


How does K-NN work?

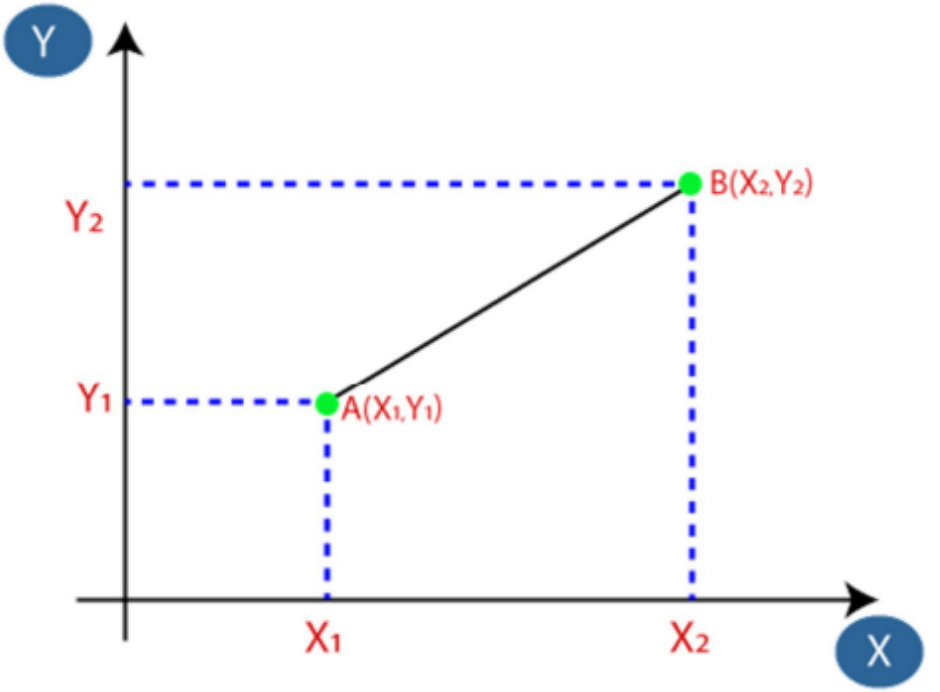
The K-NN working can be explained on the basis of the below algorithm:

- **Step-1:** Select the number K of the neighbors
- **Step-2:** Calculate the Euclidean distance of **K number of neighbors**
- **Step-3:** Take the K nearest neighbors as per the calculated Euclidean distance.
- **Step-4:** Among these k neighbors, count the number of the data points in each category.
- **Step-5:** Assign the new data points to that category for which the number of the neighbor is maximum.
- **Step-6:** Our model is ready.

Suppose we have a new data point and we need to put it in the required category. Consider the below image:

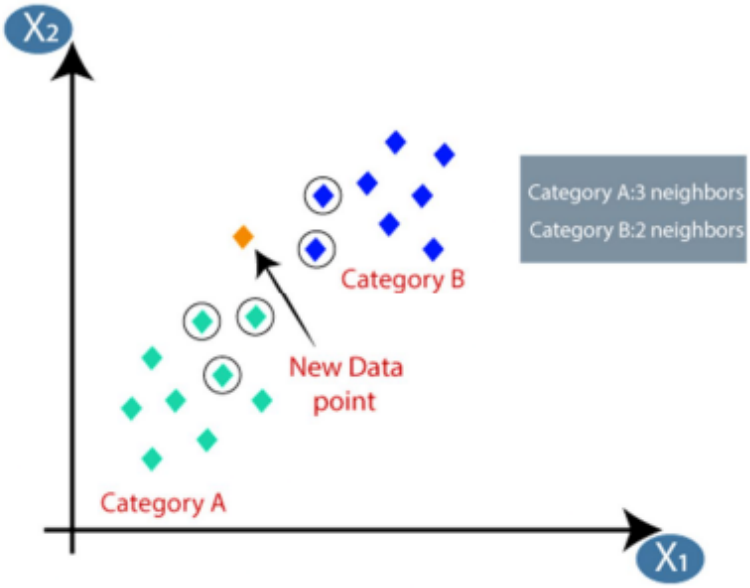


- Firstly, we will choose the number of neighbors, so we will choose the k=5.
- Next, we will calculate the **Euclidean distance** between the data points. The Euclidean distance is the distance between two points, which we have already studied in geometry. It can be calculated as:



Euclidean Distance between A_1 and B_2 = $\sqrt{(X_2-X_1)^2+(Y_2-Y_1)^2}$

- By calculating the Euclidean distance we got the nearest neighbors, as three nearest neighbors in category A and two nearest neighbors in category B. Consider the below image:



- As we can see the 3 nearest neighbors are from category A, hence this new data point must belong to category A.

How to select the value of K in the K-NN Algorithm?

Below are some points to remember while selecting the value of K in the K-NN algorithm:

- There is no particular way to determine the best value for "K", so we need to try some values to find the best out of them. The most preferred value for K is 5.
- A very low value for K such as K=1 or K=2, can be noisy and lead to the effects of outliers in the model.
- Large values for K are good, but it may find some difficulties.

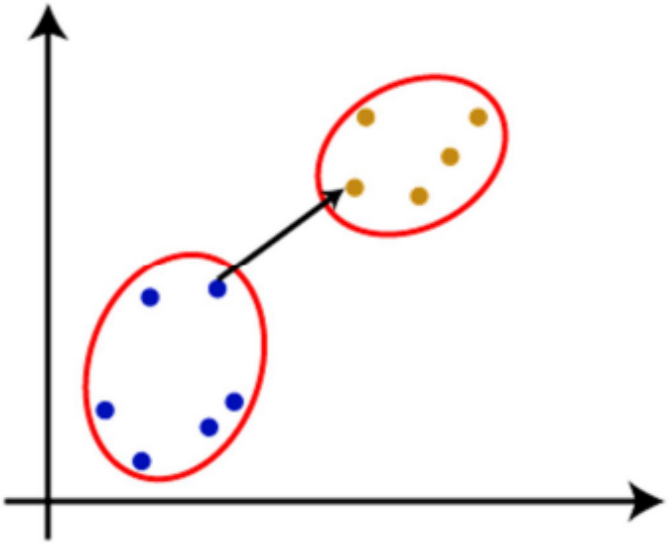
Advantages of KNN Algorithm:

- It is simple to implement.
- It is robust to the noisy training data
- It can be more effective if the training data is large.

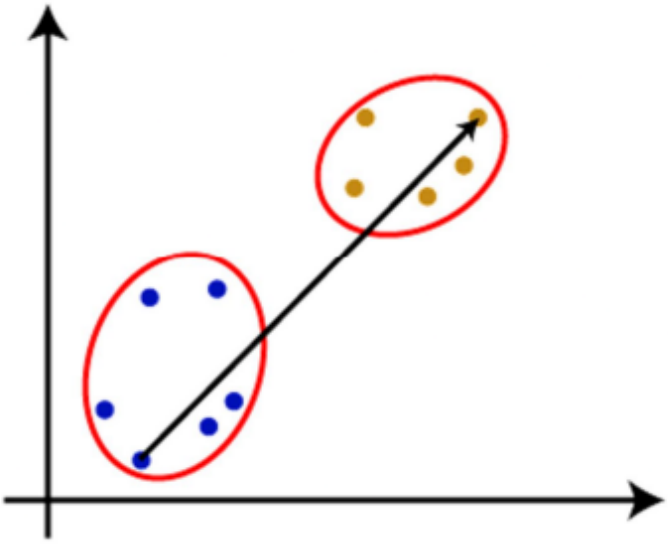
Disadvantages of KNN Algorithm:

- Always needs to determine the value of K which may be complex some time.
- The computation cost is high because of calculating the distance between the data points for all the training samples.

1. **Single Linkage:** It is the Shortest Distance between the closest points of the clusters. Consider the below image:

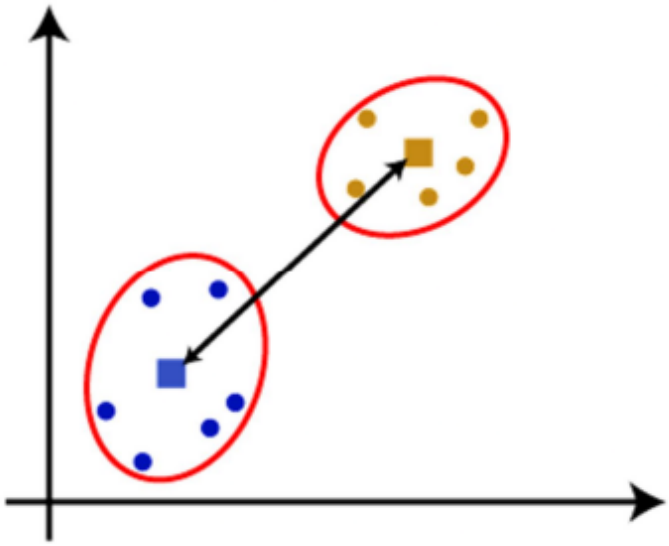


2. **Complete Linkage:** It is the farthest distance between the two points of two different clusters. It is one of the popular linkage methods as it forms tighter clusters than single-linkage.



3. **Average Linkage:** It is the linkage method in which the distance between each pair of datasets is added up and then divided by the total number of datasets to calculate the average distance between two clusters. It is also one of the most popular linkage methods.

4. **Centroid Linkage:** It is the linkage method in which the distance between the centroid of the clusters is calculated. Consider the below image:

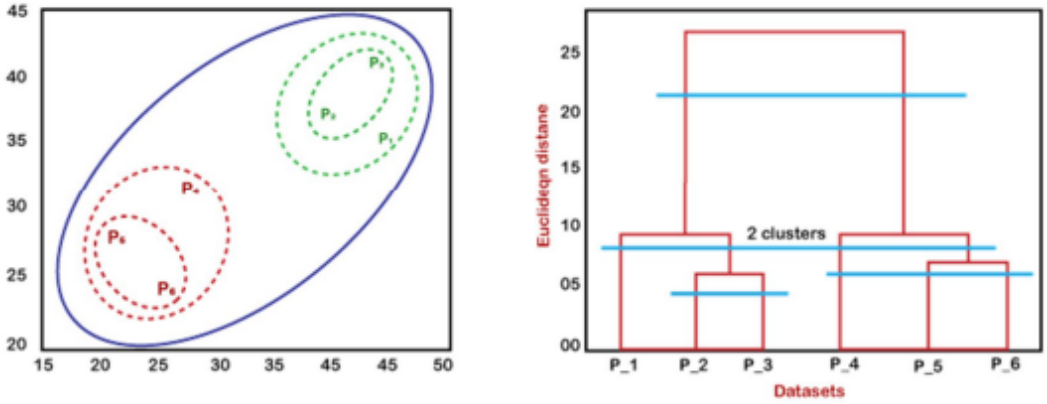


From the above-given approaches, we can apply any of them according to the type of problem or business requirement.

Woking of Dendrogram in Hierarchical clustering

The dendrogram is a tree-like structure that is mainly used to store each step as a memory that the HC algorithm performs. In the dendrogram plot, the Y-axis shows the Euclidean distances between the data points, and the x-axis shows all the data points of the given dataset.

The working of the dendrogram can be explained using the below diagram:



In the above diagram, the left part is showing how clusters are created in agglomerative clustering, and the right part is showing the corresponding dendrogram.

- As we have discussed above, firstly, the datapoints P2 and P3 combine together and form a cluster, correspondingly a dendrogram is created, which connects P2 and P3 with a rectangular shape. The hight is decided according to the Euclidean distance between the data points.
- In the next step, P5 and P6 form a cluster, and the corresponding dendrogram is created. It is higher than of previous, as the Euclidean distance between P5 and P6 is a little bit greater than the P2 and P3.
- Again, two new dendrograms are created that combine P1, P2, and P3 in one dendrogram, and P4, P5, and P6, in another dendrogram.
- At last, the final dendrogram is created that combines all the data points together.

We can cut the dendrogram tree structure at any level as per our requirement.

Introduction to Dimensionality Reduction Technique

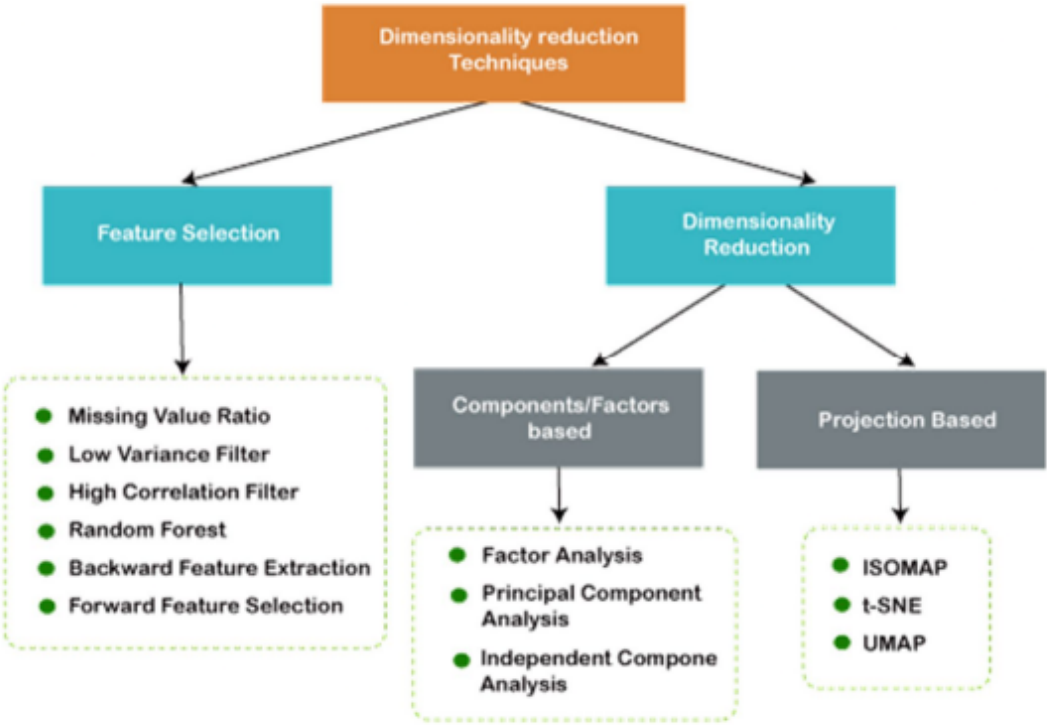
What is Dimensionality Reduction?

The number of input features, variables, or columns present in a given dataset is known as dimensionality, and the process to reduce these features is called dimensionality reduction.

A dataset contains a huge number of input features in various cases, which makes the predictive modeling task more complicated. Because it is very difficult to visualize or make predictions for the training dataset with a high number of features, for such cases, dimensionality reduction techniques are required to use.

Dimensionality reduction technique can be defined as, ***"It is a way of converting the higher dimensions dataset into lesser dimensions dataset ensuring that it provides similar information."*** These techniques are widely used in machine learning for obtaining a better fit predictive model while solving the classification and regression problems.

It is commonly used in the fields that deal with high-dimensional data, such as **speech recognition, signal processing, bioinformatics, etc. It can also be used for data visualization, noise reduction, cluster analysis**, etc.



The Curse of Dimensionality

Handling the high-dimensional data is very difficult in practice, commonly known as the *curse of dimensionality*. If the dimensionality of the input dataset increases, any machine learning algorithm and model becomes more complex. As the number of features increases, the number of samples also gets increased proportionally, and the chance of overfitting also increases. If the machine learning model is trained on high-dimensional data, it becomes overfitted and results in poor performance.

Hence, it is often required to reduce the number of features, which can be done with dimensionality reduction.

Benefits of applying Dimensionality Reduction

Some benefits of applying dimensionality reduction technique to the given dataset are given below:

- By reducing the dimensions of the features, the space required to store the dataset also gets reduced.
- Less Computation training time is required for reduced dimensions of features.
- Reduced dimensions of features of the dataset help in visualizing the data quickly.
- It removes the redundant features (if present) by taking care of multicollinearity.

Disadvantages of dimensionality Reduction

There are also some disadvantages of applying the dimensionality reduction, which are given below:

- Some data may be lost due to dimensionality reduction.
- In the PCA dimensionality reduction technique, sometimes the principal components required to consider are unknown.

Approaches of Dimension Reduction

There are two ways to apply the dimension reduction technique, which are given below:

Feature Selection

Feature selection is the process of selecting the subset of the relevant features and leaving out the irrelevant features present in a dataset to build a model of high accuracy. In other words, it is a way of selecting the optimal features from the input dataset.

Three methods are used for the feature selection:

1. Filters Methods

In this method, the dataset is filtered, and a subset that contains only the relevant features is taken. Some common techniques of filters method are:

- **Correlation**
- **Chi-Square Test**
- **ANOVA**
- **Information Gain, etc.**

2. Wrappers Methods

The wrapper method has the same goal as the filter method, but it takes a machine learning model for its evaluation. In this method, some features are fed to the ML model, and evaluate the performance. The performance decides whether to add those features or remove to increase the accuracy of the model. This method is more accurate than the filtering method but complex to work. Some common techniques of wrapper methods are:

- Forward Selection
- Backward Selection
- Bi-directional Elimination

3. Embedded Methods: Embedded methods check the different training iterations of the machine learning model and evaluate the importance of each feature. Some common techniques of Embedded methods are:

- **LASSO**
- **Elastic Net**
- **Ridge Regression, etc.**

Feature Extraction:

Feature extraction is the process of transforming the space containing many dimensions into space with fewer dimensions. This approach is useful when we want to keep the whole information but use fewer resources while processing the information.

Some common feature extraction techniques are:

a. Principal Component Analysis
b. Linear Discriminant Analysis
c. Kernel PCA
d. Quadratic Discriminant Analysis

Common techniques of Dimensionality Reduction

a. **Principal Component Analysis**
b. **Backward Elimination**
c. **Forward Selection**
d. **Score comparison**
e. **Missing Value Ratio**
f. **Low Variance Filter**
g. **High Correlation Filter**
h. **Random Forest**
i. **Factor Analysis**
j. **Auto-Encoder**

Principal Component Analysis (PCA)

Principal Component Analysis is a statistical process that converts the observations of correlated features into a set of linearly uncorrelated features with the help of orthogonal transformation. These new transformed features are called the **Principal Components**. It is one of the popular tools that is used for exploratory data analysis and predictive modeling.

PCA works by considering the variance of each attribute because the high attribute shows the good split between the classes, and hence it reduces the dimensionality. Some real-world applications of PCA are ***image processing, movie recommendation system, optimizing the power allocation in various communication channels.***

Backward Feature Elimination

The backward feature elimination technique is mainly used while developing Linear Regression or Logistic Regression model. Below steps are performed in this technique to reduce the dimensionality or in feature selection:

- In this technique, firstly, all the n variables of the given dataset are taken to train the model.
- The performance of the model is checked.
- Now we will remove one feature each time and train the model on n-1 features for n times, and will compute the performance of the model.
- We will check the variable that has made the smallest or no change in the performance of the model, and then we will drop that variable or features; after that, we will be left with n-1 features.
- Repeat the complete process until no feature can be dropped.

In this technique, by selecting the optimum performance of the model and maximum tolerable error rate, we can define the optimal number of features require for the machine learning algorithms.

Forward Feature Selection

Forward feature selection follows the inverse process of the backward elimination process. It means, in this technique, we don't eliminate the feature; instead, we will find the best features that can produce the highest increase in the performance of the model. Below steps are performed in this technique:

- We start with a single feature only, and progressively we will add each feature at a time.
- Here we will train the model on each feature separately.
- The feature with the best performance is selected.
- The process will be repeated until we get a significant increase in the performance of the model.

Missing Value Ratio

If a dataset has too many missing values, then we drop those variables as they do not carry much useful information. To perform this, we can set a threshold level, and if a variable has missing values more than that threshold, we will drop that variable. The higher the threshold value, the more efficient the reduction.

Low Variance Filter

As same as missing value ratio technique, data columns with some changes in the data have less information. Therefore, we need to calculate the variance of each variable, and all data columns with variance lower than a given threshold are dropped because low variance features will not affect the target variable.

High Correlation Filter

High Correlation refers to the case when two variables carry approximately similar information. Due to this factor, the performance of the model can be degraded. This correlation between the independent numerical variable gives the calculated value of the correlation coefficient. If this value is higher than the threshold value, we can remove one of the variables from the dataset. We can consider those variables or features that show a high correlation with the target variable.

Random Forest

Random Forest is a popular and very useful feature selection algorithm in machine learning. This algorithm contains an in-built feature importance package, so we do not need to program it separately. In this technique, we need to generate a large set of trees against the target variable, and with the help of usage statistics of each attribute, we need to find the subset of features.

Random forest algorithm takes only numerical variables, so we need to convert the input data into numeric data using **hot encoding**.

Factor Analysis

Factor analysis is a technique in which each variable is kept within a group according to the correlation with other variables, it means variables within a group can have a high correlation between themselves, but they have a low correlation with variables of other groups.

We can understand it by an example, such as if we have two variables Income and spend. These two variables have a high correlation, which means people with high income spends more, and vice versa. So, such variables are put into a group, and that group is known as the **factor**. The number of these factors will be reduced as compared to the original dimension of the dataset.

Auto-encoders

One of the popular methods of dimensionality reduction is auto-encoder, which is a type of ANN or artificial neural network, and its main aim is to copy the inputs to their outputs. In this,

the input is compressed into latent-space representation, and output is occurred using this representation. It has mainly two parts:

- **Encoder:** The function of the encoder is to compress the input to form the latent-space representation.
- **Decoder:** The function of the decoder is to recreate the output from the latent-space representation.

Principal Component Analysis

Principal Component Analysis is an unsupervised learning algorithm that is used for the dimensionality reduction in machine learning

. It is a statistical process that converts the observations of correlated features into a set of linearly uncorrelated features with the help of orthogonal transformation. These new transformed features are called the **Principal Components**. It is one of the popular tools that is used for exploratory data analysis and predictive modeling. It is a technique to draw strong patterns from the given dataset by reducing the variances.

PCA generally tries to find the lower-dimensional surface to project the high-dimensional data.

PCA works by considering the variance of each attribute because the high attribute shows the good split between the classes, and hence it reduces the dimensionality. Some real-world applications of PCA are ***image processing, movie recommendation system, optimizing the power allocation in various communication channels.*** It is a feature extraction technique, so it contains the important variables and drops the least important variable.

The PCA algorithm is based on some mathematical concepts such as:

- Variance and Covariance
- Eigenvalues and Eigen factors

Some common terms used in PCA algorithm:

- **Dimensionality:** It is the number of features or variables present in the given dataset. More easily, it is the number of columns present in the dataset.
- **Correlation:** It signifies that how strongly two variables are related to each other. Such as if one changes, the other variable also gets changed. The correlation value ranges from -1 to +1. Here, -1 occurs if variables are inversely proportional to each other, and +1 indicates that variables are directly proportional to each other.
- **Orthogonal:** It defines that variables are not correlated to each other, and hence the correlation between the pair of variables is zero.
- **Eigenvectors:** If there is a square matrix M, and a non-zero vector v is given. Then v will be eigenvector if Av is the scalar multiple of v.
- **Covariance Matrix:** A matrix containing the covariance between the pair of variables is called the Covariance Matrix.

Principal Components in PCA

As described above, the transformed new features or the output of PCA are the Principal Components. The number of these PCs are either equal to or less than the original features present in the dataset. Some properties of these principal components are given below:

- The principal component must be the linear combination of the original features.
- These components are orthogonal, i.e., the correlation between a pair of variables is zero.
- The importance of each component decreases when going to 1 to n, it means the 1 PC has the most importance, and n PC will have the least importance.

Steps for PCA algorithm

1. **Getting the dataset**
   Firstly, we need to take the input dataset and divide it into two subparts X and Y, where X is the training set, and Y is the validation set.
2. **Representing data into a structure**
   Now we will represent our dataset into a structure. Such as we will represent the two-dimensional matrix of independent variable X. Here each row corresponds to the data items, and the column corresponds to the Features. The number of columns is the dimensions of the dataset.
3. **Standardizing the data**
   In this step, we will standardize our dataset. Such as in a particular column, the features with high variance are more important compared to the features with lower variance.
   If the importance of features is independent of the variance of the feature, then we will divide each data item in a column with the standard deviation of the column. Here we will name the matrix as Z.
4. **Calculating the Covariance of Z**
   To calculate the covariance of Z, we will take the matrix Z, and will transpose it. After transpose, we will multiply it by Z. The output matrix will be the Covariance matrix of Z.
5. **Calculating the Eigen Values and Eigen Vectors**
   Now we need to calculate the eigenvalues and eigenvectors for the resultant covariance matrix Z. Eigenvectors or the covariance matrix are the directions of the axes with high information. And the coefficients of these eigenvectors are defined as the eigenvalues.

6. **Sorting the Eigen Vectors**

   In this step, we will take all the eigenvalues and will sort them in decreasing order, which means from largest to smallest. And simultaneously sort the eigenvectors accordingly in matrix P of eigenvalues. The resultant matrix will be named as P*.

7. **Calculating the new features Or Principal Components**

   Here we will calculate the new features. To do this, we will multiply the P* matrix to the Z. In the resultant matrix Z*, each observation is the linear combination of original features. Each column of the Z* matrix is independent of each other.

8. **Remove less or unimportant features from the new dataset.**

   The new feature set has occurred, so we will decide here what to keep and what to remove. It means, we will only keep the relevant or important features in the new dataset, and unimportant features will be removed out.

Applications of Principal Component Analysis

- PCA is mainly used as the dimensionality reduction technique in various AI applications such **as computer vision, image compression, etc.**
- It can also be used for finding hidden patterns if data has high dimensions. Some fields where PCA is used are Finance, data mining, Psychology, etc.

Chapter 5

Measures for performance evaluation of ML algorithm

Classification accuracy

Confusion matrix

Misclassification costs

Sensitivity and specificity

ROC curve

Recall and precision

Box plot and Confidence interval

Confusion Matrix in Machine Learning

The confusion matrix is a matrix used to determine the performance of the classification models for a given set of test data. It can only be determined if the true values for test data are known. The matrix itself can be easily understood, but the related terminologies may be confusing. Since it shows the errors in the model performance in the form of a matrix, hence also known as an **error matrix**. Some features of Confusion matrix are given below:

- For the 2 prediction classes of classifiers, the matrix is of 2*2 table, for 3 classes, it is 3*3 table, and so on.
- The matrix is divided into two dimensions, that are **predicted values** and **actual values** along with the total number of predictions.
- Predicted values are those values, which are predicted by the model, and actual values are the true values for the given observations.
- It looks like the below table:

n = total predictions	Actual: No	Actual: Yes
Predicted: No	True Negative	False Positive
Predicted: Yes	False Negative	True Positive

The above table has the following cases:

- **True Negative:** Model has given prediction No, and the real or actual value was also No.
- **True Positive:** The model has predicted yes, and the actual value was also true.
- **False Negative:** The model has predicted no, but the actual value was Yes, it is also called as **Type-II error**.
- **False Positive:** The model has predicted Yes, but the actual value was No. It is also called a **Type-I error.**

Need for Confusion Matrix in Machine learning

- It evaluates the performance of the classification models, when they make predictions on test data, and tells how good our classification model is.
- It not only tells the error made by the classifiers but also the type of errors such as it is either type-I or type-II error.
- With the help of the confusion matrix, we can calculate the different parameters for the model, such as accuracy, precision, etc.

Example: We can understand the confusion matrix using an example.

Suppose we are trying to create a model that can predict the result for the disease that is either a person has that disease or not. So, the confusion matrix for this is given as:

n = 100	Actual: No	Actual: Yes	
Predicted: No	TN: 65	FP: 3	68
Predicted: Yes	FN: 8	TP: 24	32
	73	27	

From the above example, we can conclude that:

- The table is given for the two-class classifier, which has two predictions "Yes" and "NO." Here, Yes defines that patient has the disease, and No defines that patient does not has that disease.
- The classifier has made a total of **100 predictions**. Out of 100 predictions, **89 are true predictions**, and **11 are incorrect predictions**.
- The model has given prediction "yes" for 32 times, and "No" for 68 times. Whereas the actual "Yes" was 27, and actual "No" was 73 times.

Calculations using Confusion Matrix:

We can perform various calculations for the model, such as the model's accuracy, using this matrix. These calculations are given below:

- **Classification Accuracy:** It is one of the important parameters to determine the accuracy of the classification problems. It defines how often the model predicts the correct output. It can be calculated as the ratio of the number of correct predictions made by the classifier to all number of predictions made by the classifiers. The formula is given below:

$$\text{Accuracy}=\frac{TP+TN}{TP+FP+FN+TN}$$

- **Misclassification rate:** It is also termed as Error rate, and it defines how often the model gives the wrong predictions. The value of error rate can be calculated as the number of incorrect predictions to all number of the predictions made by the classifier. The formula is given below:

$$\text{Error rate}=\frac{FP+FN}{TP+FP+FN+TN}$$

- **Precision:** It can be defined as the number of correct outputs provided by the model or out of all positive classes that have predicted correctly by the model, how many of them were actually true. It can be calculated using the below formula:

$$\text{Precision}=\frac{TP}{TP+FP}$$

- **Recall:** It is defined as the out of total positive classes, how our model predicted correctly. The recall must be as high as possible.

$$\text{Recall} = \frac{TP}{TP+FN}$$

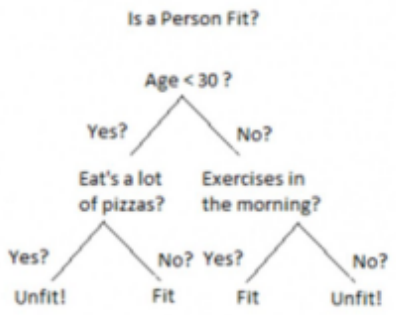
- **F-measure:** If two models have low precision and high recall or vice versa, it is difficult to compare these models. So, for this purpose, we can use F-score. This score helps us to evaluate the recall and precision at the same time. The F-score is maximum if the recall is equal to the precision. It can be calculated using the below formula:

$$\textbf{F-measure} = \frac{2*Recall*Precision}{Recall+Precision}$$

Other important terms used in Confusion Matrix:

- **Null Error rate:** It defines how often our model would be incorrect if it always predicted the majority class. As per the accuracy paradox, it is said that "*the best classifier has a higher error rate than the null error rate.*"
- **ROC Curve:** The ROC is a graph displaying a classifier's performance for all possible thresholds. The graph is plotted between the true positive rate (on the Y-axis) and the false Positive rate (on the x-axis).

Introduction Decision Trees are a type of Supervised Machine Learning (that is you explain what the input is and what the corresponding output is in the training data) where the data is continuously split according to a certain parameter. The tree can be explained by two entities, namely decision nodes and leaves. The leaves are the decisions or the final outcomes. And the



decision nodes are where the data is split. An example of a decision tree can be explained using above binary tree. Let's say you want to predict whether a person is fit given their information like age, eating habit, and physical activity, etc. The decision nodes here are questions like 'What's the age?', 'Does he exercise?', 'Does he eat a lot of pizzas'? And the leaves, which are outcomes like either 'fit', or 'unfit'. In this case this was a binary classification problem (a yes no type problem). There are two main types of Decision Trees:

1. **Classification trees** (Yes/No types)

What we've seen above is an example of classification tree, where the outcome was a variable like 'fit' or 'unfit'. Here the decision variable is **Categorical**.

2. **Regression trees** (Continuous data types)

Here the decision or the outcome variable is **Continuous**, e.g. a number like 123. **Working** Now that we know what a Decision Tree is, we'll see how it works internally. There are many algorithms out there which construct Decision Trees, but one of the best is called as **ID3 Algorithm**. ID3 Stands for **Iterative Dichotomiser 3**. Before discussing the ID3 algorithm, we'll go through few definitions. **Entropy** Entropy, also called as Shannon Entropy is denoted by H(S) for a finite set S, is the measure of the amount of uncertainty or randomness in data.

$$H(S) = \sum_{x \in X} p(x) \log_2 \frac{1}{p(x)}$$

Intuitively, it tells us about the predictability of a certain event. Example, consider a coin toss whose probability of heads is 0.5 and probability of tails is 0.5. Here the entropy is the highest possible, since there's no way of determining what the outcome might be. Alternatively, consider a coin which has heads on both the sides, the entropy of such an event can be predicted perfectly since we know beforehand that it'll always be heads. In other words, this event has **no randomness** hence it's entropy is zero. In particular, lower values imply less

uncertainty while higher values imply high uncertainty. **Information Gain** Information gain is also called as Kullback-Leibler divergence denoted by IG(S,A) for a set S is the effective change in entropy after deciding on a particular attribute A. It measures the relative change in entropy with respect to the independent variables. $IG(S, A) = H(S) - H(S, A)$

Alternatively, $$IG(S,A) = H(S) - \sum_{i=0}^{n} P(x) * H(x)$$ where IG(S, A) is the information gain by applying feature A. H(S) is the Entropy of the entire set, while the second term calculates the Entropy after applying the feature A, where P(x) is the probability of event x. Let's understand this with the help of an example Consider a piece of data collected over the course of 14 days where the features are Outlook, Temperature, Humidity, Wind and the outcome variable is whether Golf was played on the day. Now, our job is to build a predictive model which takes in above 4 parameters and predicts whether Golf will be played on the day. We'll build a decision tree to do that using **ID3 algorithm.**

Day	Outlook	Temperature	Humidity	Wind	Play Golf
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

ID3 Algorithm will perform following tasks recursively

1. Create root node for the tree
2. If all examples are positive, return leaf node 'positive'
3. Else if all examples are negative, return leaf node 'negative'
4. Calculate the entropy of current state H(S)
5. For each attribute, calculate the entropy with respect to the attribute 'x' denoted by H(S, x)
6. Select the attribute which has maximum value of IG(S, x)
7. Remove the attribute that offers highest IG from the set of attributes
8. Repeat until we run out of all attributes, or the decision tree has all leaf nodes.

Now we'll go ahead and grow the decision tree. The initial step is to calculate H(S), the Entropy of the current state. In the above example, we can see in total there are 5 No's and 9 Yes's.

Yes	No	Total
9	5	14

$$Entropy(S) = \sum_{x \in X} p(x) \log_2 \frac{1}{p(x)}$$

$$Entropy(S) = -\left(\frac{9}{14}\right)\log_2\left(\frac{9}{14}\right) - \left(\frac{5}{14}\right)\log_2\left(\frac{5}{14}\right)$$

$$= 0.940$$

Remember that the Entropy is 0 if all members belong to the same class, and 1 when half of them belong to one class and other half belong to other class that is perfect randomness. Here it's 0.94 which means the distribution is fairly random. **Now the next step is to choose the attribute that gives us highest possible Information Gain** which we'll choose as the root node. Let's start with 'Wind' where 'x' are the possible values for an attribute. Here, attribute 'Wind' takes two possible values in the sample data, hence x = {Weak, Strong} We'll have to calculate:

$$IG(S, Wind) = H(S) - \sum_{i=0}^{n} P(x) * H(x)$$

1. $H(S_{weak})$
2. $H(S_{strong})$
3. $P(S_{weak})$
4. $P(S_{strong})$
5. $H(S) = 0.94$ which we had already calculated in the previous example

Amongst all the 14 examples we have **8 places where the wind is weak and 6 where the wind is Strong**.

Wind = Weak	Wind = Strong	Total
8	6	14

$$P(S_{weak}) = \frac{Number\ of\ Weak}{Total}$$

$$= \frac{8}{14}$$

$$P(S_{strong}) = \frac{Number\ of\ Strong}{Total}$$

$$= \frac{6}{14}$$

Now out of the 8 Weak examples, 6 of them were 'Yes' for Play Golf and 2 of them were 'No' for 'Play Golf'. So, we have,

$$Entropy(S_{weak}) = -\left(\frac{6}{8}\right)\log_2\left(\frac{6}{8}\right) - \left(\frac{2}{8}\right)\log_2\left(\frac{2}{8}\right)$$

$$= 0.811$$

Similarly, out of 6 Strong examples, we

have **3 examples where the outcome was 'Yes' for Play Golf and 3 where we had 'No' for Play Golf**. Remember, here half items belong to one class while other half belong to other. Hence we have perfect randomness. Now we have all the pieces required to calculate the Information Gain, Which tells us the Information Gain by considering 'Wind' as the feature and give us information gain of **0.048**. Now we must similarly calculate the Information Gain for all the features. We can clearly see that IG(S, Outlook) has the highest information gain of 0.246, **hence we chose Outlook attribute as the root node**. At this point, the decision tree looks like. Here we observe that whenever the outlook is Overcast, Play Golf is always 'Yes', it's no coincidence by any chance, the simple tree resulted because of **the highest information gain is given by the attribute Outlook**. Now how do we proceed from this point? We can simply apply **recursion**, you might want to look at the algorithm steps described earlier. Now that we've used Outlook, we've got three of them remaining Humidity, Temperature, and Wind. And, we had three possible values of Outlook: Sunny, Overcast, Rain. Where the Overcast node already ended up having leaf node 'Yes', so we're left with two subtrees to compute: Sunny and Rain. Next step would be computing $H(S_{sunny})$. Table where the value of Outlook is Sunny looks like:

$$Entropy(S_{strong}) = -\left(\frac{3}{6}\right)\log_2\left(\frac{3}{6}\right) - \left(\frac{3}{6}\right)\log_2\left(\frac{3}{6}\right)$$

$$= 1.000$$

$$IG(S, Wind) = H(S) - \sum_{i=0}^{n} P(x) * H(x)$$

$$IG(S, Wind) = H(S) - P(S_{weak}) * H(S_{weak}) - P(S_{strong}) * H(S_{strong})$$

$$= 0.940 - \left(\frac{8}{14}\right)(0.811) - \left(\frac{6}{14}\right)(1.00)$$

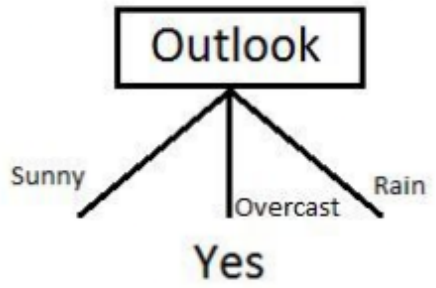
$$= 0.048$$

$$IG(S, Outlook) = 0.246$$

$$IG(S, Temperature) = 0.029$$

$$IG(S, Humidity) = 0.151$$

$$IG(S, Wind) = 0.048 \text{ (Previous example)}$$



Temperature	Humidity	Wind	Play Golf

Hot	High	Weak	No
Hot	High	Strong	No
Mild	High	Weak	No
Cool	Normal	Weak	Yes
Mild	Normal	Strong	Yes

$$H(S_{sunny}) = \left(\frac{3}{5}\right)\log_2\left(\frac{3}{5}\right) - \left(\frac{2}{5}\right)\log_2\left(\frac{2}{5}\right) = 0.96$$

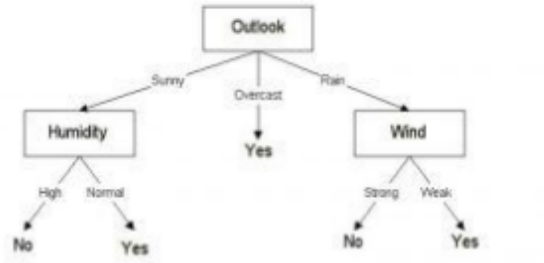
In the similar fashion, we compute the following values

$$IG(S_{sunny}, Humidity) = 0.96$$

$$IG(S_{sunny}, Temperature) = 0.57$$

$$IG(S_{sunny}, Wind) = 0.019$$

As we can see the **highest Information Gain is given by Humidity**. Proceeding in the same way with S_{rain} will give us Wind as the one with highest information gain. The final Decision Tree looks something like this. The final Decision Tree looks something like this.



Code: Let's see an example in Python

```
import pydotplus
from sklearn.datasets import load_iris
from sklearn import tree
from IPython.display import Image, display
__author__ = "Mayur Kulkarni <mayur.kulkarni@xoriant.com>"

def load_data_set():
    """
    Loads the iris data set

    :return:      data set instance
    """
    iris = load_iris()
    return iris

def train_model(iris):
    """
    Train decision tree classifier
```

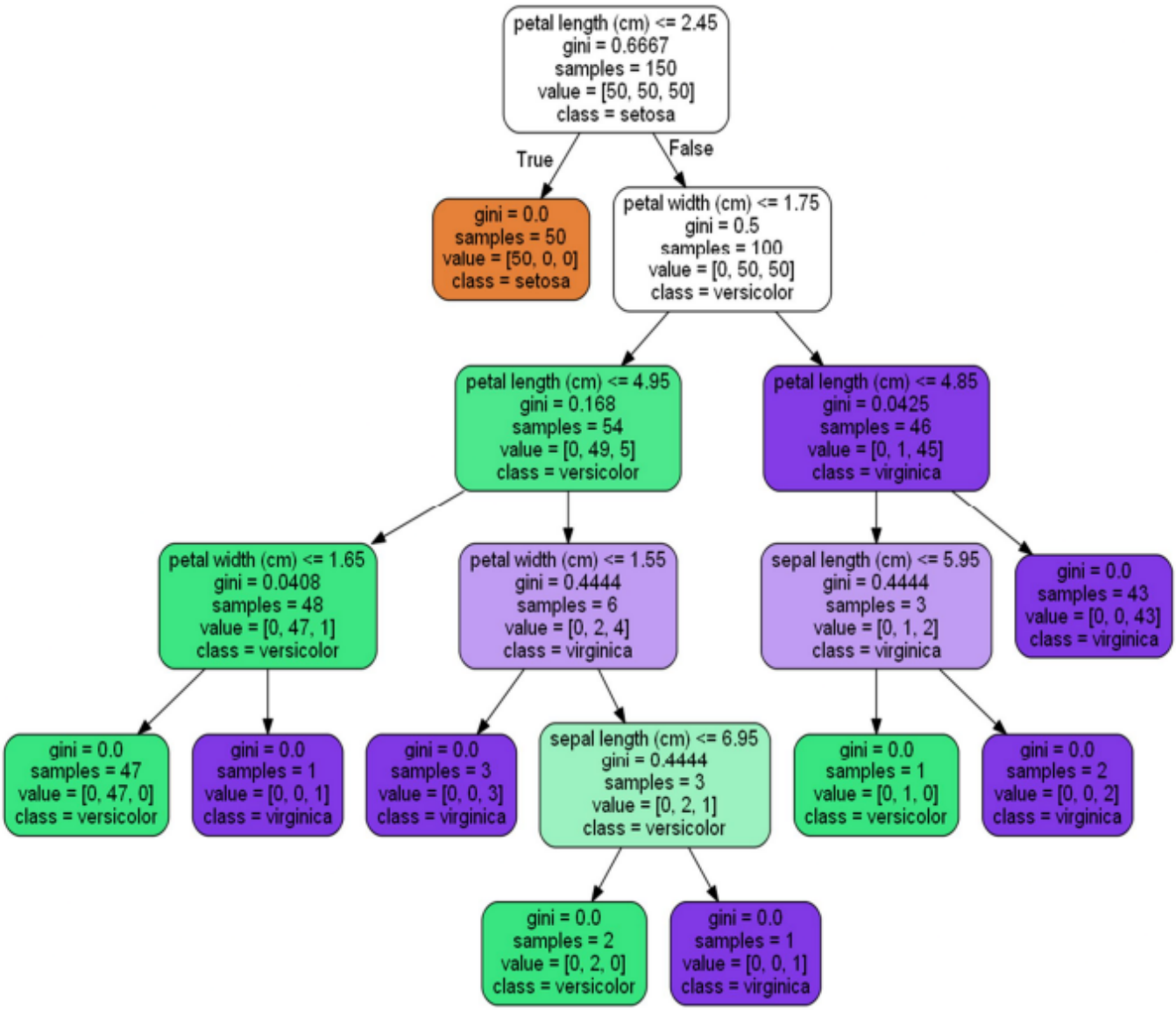
```
    :param iris:    iris data set instance
    :return:        classifier instance
    """
    clf = tree.DecisionTreeClassifier()
    clf = clf.fit(iris.data, iris.target)
    return clf

def display_image(clf, iris):
    """
    Displays the decision tree image

    :param clf:     classifier instance
    :param iris:    iris data set instance
    """
    dot_data = tree.export_graphviz(clf, out_file=None,
                                    feature_names=iris.feature_names,
                                    class_names=iris.target_names,
                                    filled=True, rounded=True)

    graph = pydotplus.graph_from_dot_data(dot_data)
    display(Image(data=graph.create_png()))

if __name__ == '__main__':
    iris_data = load_iris()
    decision_tree_classifier = train_model(iris_data)
    display_image(clf=decision_tree_classifier, iris=iris_data)
```



Conclusion: Below is the summary of what we've studied in this blog:

1. Entropy to measure discriminatory power of an attribute for classification task. It defines the amount of randomness in attribute for classification task. Entropy is minimal means the attribute appears close to one class and have a good discriminatory power for classification
2. Information Gain to rank attribute for filtering at given node in the tree. The ranking is based on high information gain entropy in decreasing order.
3. The recursive ID3 algorithm that creates a decision tree.

CS 2750 Machine Learning
Lecture 11

Support vector machines

Milos Hauskrecht
milos@cs.pitt.edu
5329 Sennott Square

Outline

Outline:

- **Support vector machines**
- Linearly separable classes. Algorithms.
- Maximum margin hyperplane.
- Support vectors.
- Support vector machines.

- Extensions to the non-separable case.
- Kernel functions.

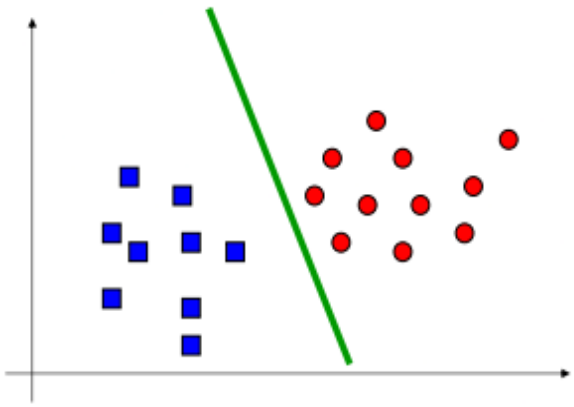
Linearly separable classes

There is a **hyperplane** that separates training instances with no error

Hyperplane:

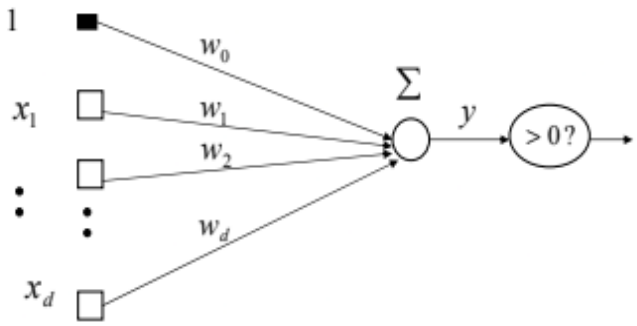
$\mathbf{w}^T\mathbf{x} + w_0 = 0$

Class (+1) $\mathbf{w}^T\mathbf{x} + w_0 > 0$
Class (-1) $\mathbf{w}^T\mathbf{x} + w_0 < 0$



Algorithms for linearly separable set

- **Hyperplane** $\mathbf{w}^T\mathbf{x} + w_0 = 0$



- We can use **gradient methods** for sigmoidal switching functions and learn the weights
- Recall that we learn the linear decision boundary

Algorithms for linearly separable sets

- **Linear program solution:**
  - Find weights that satisfy the following constraints:

$$\mathbf{w}^T \mathbf{x}_i + w_0 \geq 0 \quad \text{For all i, such that} \quad y_i = +1$$

$$\mathbf{w}^T \mathbf{x}_i + w_0 \leq 0 \quad \text{For all i, such that} \quad y_i = -1$$

Property: if there is a hyperplane separating the examples, the linear program finds the solution

Other methods:

Fisher linear discriminant

Optimal separating hyperplane

- There are multiple hyperplanes that separate the data points
  - Which one to choose?
- **Maximum margin** choice: the maximum distance of $d_+ + d_-$
  - where d_+ is the shortest distance of a positive example from the hyperplane (similarly d_- for negative examples)