TECHNICAL MANUAL STYLE

Last updated: 02/27/2023

This page intentionally left blank.

Contents

1 Primer Concepts

Since the invention of computers or machines, their capability to perform various tasks has experienced an exponential growth. Humans have developed the power of computer systems in terms of their diverse working domains, their incresing speed, and reducing size with respect to time.

A branch of Computer Science named Artificial Intelligence pursues creating the computers or machines as intelligent as human beings.

1.1 Basic Concept of Artificial Intelligence (AI)

According to the father of Artificial Intelligence, John McCarthy, it is "The sceince and engineering of making intelligent machines, especially intelligent computer programs".

Artificail Intelligence is a way of making a computer, a computer-controlled robot, or a software think intelligently, in the similar manner to intelligent human think. AI is accomplished by studying how human brain thinks and how humans learn, decide, and work while trying to solve a problem, and then using the ourcomes of this study as a basis of developing intelligent software and systems.

While exploiting the power of AI started with intention of creating similar intelligence in machines that we find and regard high in humans.

1.2 The Necessity of Learning AI

As we know that AI purues creating the machines as intelligent as human beings. There are numerous reasons for us to study AI. The reasons are as follows -

1.2.1 AI can learn through data

In our daily life, we deal with huge amount of data and human brain cannot keep track of so much data. That is why we need to automate the things. For doing automation, we need to study AI because it can learn from data and can do the repetitive tasks with accuracy and without tiredness.

1.2.2 AI can teach itself

It is very necessary that a system should teach itself because the data itself keeps changing and the knowledge which is derived from such data must be updated constantly. We can use AI to fulfill this purpose because an AI enabled system can teach itself.

1.2.3 AI can respond in real time

Artificial intelligence with the help of neural networks can analyze the data more deeply. Due to this capability, AI can think and respond to the situations which are based on the conditions in real time.

1.2.4 AI achieves accuracy

With the help of deep neural networks, AI can acieve tremendous accuracy. AI helps in the field of medicine to diagnose diseases such as cancer from the MRIs of patients.

1.2.5 AI can organize data to get most out of it

The data is an intellectual property for the system which are using self-learning algorithms. We need AI to index and organize the data in a way that it always gives the best results.

1.2.6 Understanding Intelligence

With AI, smart systems can be built. We need to undestand the concept of intelligence so that our brain can construct another intelligence system like itself.

1.3 What is Intelligence?

The ability of a system to calculate, reason, perceive relationships and analogies, learn from experience, store and retrieve information from memory, solve problems, comprehend complex ideas, use natural language fluently, classify, generalize, and adapt new situations.

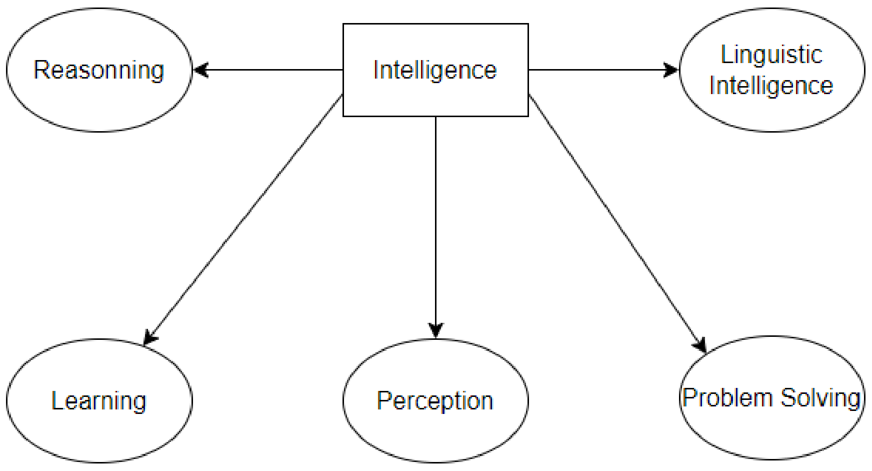
Sr.No	Intelligence & Description	Example
1	**Linguistic intelligence.** The ability to speak, recognize, and use mechanisms of phonology (speech sounds), syntax (grammar), and semantics (meaning).	Narrators, Operators
2	**Musical intelligence.** The ability to create, communicate with, and undestand meanings made of sound, understanding of pitchm rhythm.	Musicians, Singers, Composers
3	**Logical-mathemetical intelligence.** The ability to use and undestand relationships in the absence of action or objects. It is also the ability to understand complex and abstract ideas.	Mathematicians, Scientists
4	**Spatial intelligence.** The ability to perceive visual or spacial information, change it, and re-create visual images without reference to the objects, construct 3D images, and to move and rotate them.	Map readers, Astronauts, Physicists
5	**Bodily-Kinesthetic intelligence.** The abiility to use complete or part of the body to solve problems or fashion products, control over fine and coarse motor skills, and manipulate the objects.	Players, Dancers
6	**Intra-personal intelligence.** The ability to distinguish among one's own feelings, intentions, and motivations.	Gautam Buddhha
7	**Interpersonal intelligence.** The ability to recognize and make distinctions among other people's feelings, beliefs, and intentions.	Mass Communicators, Interviewers

You can say a machine or a system is artificially intelligent when it is equipped with a least one or all intelligence in it.

1.4 What is Intelligence Composed Of?

The intelligence is intangible. It is composed of:

- Reasonning
- Learning
- Problem Solving
- Perception
- Linguistic Intelligence



Intelligence

1.4.1 Reasoning

It is the set of processes that enables us to provide besis for judgement, making decisions, and prediction. There are broadly two types:

1. **Inductive Reasoning**

- It conducts specific observations to makes broad general statements.
- Even if all of the premises are true in a statement, inductive reasoning allows for the conclusion to be false.

Example: "Nita is a teacher, Nita is studious. Therefore, All teachers are studious."

2. **Deductive Reasoning**

- It starts with a general statement and examines the possibilities to reach a specific, logical conclusion.
- If something is true of a class of things in general, it is also true for all members of that class.

Example: "All women of age above 60 years are grandmothers. Shalini is 65 years. Therefore, Shalini is a grandmother."

1.4.2 Learning - 1

The ability of learning is prossessed by humans, particular species of animals, and AI-enabled systems. Learning is categorized as follows:

2. **Auditory Learning**

- It is learning by listening and hearing. For example, students listening to recorded audio lectures.

3. **Episodic Learning**

- To learn by remembering sequences of events that one has witnessed or experienced. This is linear and orderly.

3. **Motor Learning**

- It is learning by precise movement of muscles. For example, picking objects, writing, etc.

4. **Observational Learning**

- To learn by watching and imitating others. For example, child tries to learn by mimicking her parent.

5. **Perceptual Larning**

- It is learning to recognize stimuli that one has seen before. For example, identifying and classifying objects and situations.

6. **Relational Learning**

- It involves learning to differentiate among various stimuli on the basis of relational properties, rather than absolute properties. For Example, Adding 'little less' salt at the time of cooking potatoes that came up salty last time, when cooked with adding say a tablespoon of salt.

   – **Spatial Learning** - It is learning through visual stimuli such as images, colors, maps, etc. For example, A person can create roadmap in mind before actually following the road.
   – **Stimulus-Response Learning** - It is learning to perform a particular behavior when a certain stimulus is present. For example, a dog raises its ear on hearing doorbell.

7. **Problem Solving**

- It is the process in which one perceives and tries to arrive at a desired solution from a present situation by taking some path, which is blocked by known or unknown hurdles.
- Problem solving also includes **decision making**, which is the process of selecting the best suitable alternative out of multiple alternatives to reach the desired goal.

8. **Perception**

- It is the process of aquiring, interpreting, selecting, and organizing sensory information.
- Perception presumes **sensing**. In humans, perception is aided by sensory organs. In the domain of AI, perception mechanism puts the data aquired by the sensors together in a meaningful manner.

9. **Linguistic Intelligence**

- It is one's ability to use, comprehend, speak, and write the verbal and written language. It is important in interpersonal communication.

1.5 What's Involved in AI

Artificial intelligence is a vast area of study. This field of study helps in finding solutions to real world problems. Let us now see the different fields of study within AI:

3. **Machine Learning**
- It is one of the most popular fields of AI. The basic concept of this field is to make the machine learning from data as the human beings can learn from his/her experience. It contains learning models on the basis of which the predictions can be made on unknown data.

4. **Logic**
- It is another important field of study in which mathematical logic is used to execute the computer programs. It contains rules and facts to perform pattern matching, semantic analysis, etc.

4. **Searching**
- This field of study is basically used in games like chess, tic-tac-toe. Search algorithms give the optimal solution after searching the whole search space.

5. **Artificial neural networks**
- This is a network of efficient computing systems the central theme of which is borrowed from the analogy of biological neural networks. ANN can be used in robotics, speech recognition, speech processing, etc.

6. **Genetic Algorithm**
- Genetic algorithms help in solving problems with the assistance of more than one program. The result would be besed on selectinmg the fittest.

7. **Knowledge Representation**
- It is the field of study with the help of which we can represent the facts in a way the machine that is undestandable to the machine. The more efficiently knowledge is represented, the more system would be intelligent.

1.6 Application of AI

In this section, we will see the different fields supported by AI.

4. **Gaming**
- AI plays crucial role in strategic games such as chess, poker, tic-tac-toe, etc., where machine can think of large number of possibile positions based on heuristic knowledge.

5. **Natural Language Processing**
- It is possible to interact with the computer that understand natural language spoken by humans.

5. **Expert Systems**
- There are some appluications which integrate machine, software, and special information to impact reasoning and advising. They provide explanation and advice to the users.

6. **Vision Systems**

- Thease systems understand, interpret, and comprehend visual input on the computer. For example:
  - A spying aeroplane takes photohraphs, which are used to figure out spatial information or map of the areas.
  - Doctors use clinical expert system to diagnose the patient.
  - Police use computer software that can recognize the face of criminal with the stored portrait made by forensic artist.

7. **Speech Recognition**

- Some intelligent systems are capable of hearing and comprehending thelanguage in terms of sentences and their meanings while a human talks to it. It can handle different accents, slang words, noise in the background, change in human's noise due to cold, etc.

8. **Handwriting Recognition**

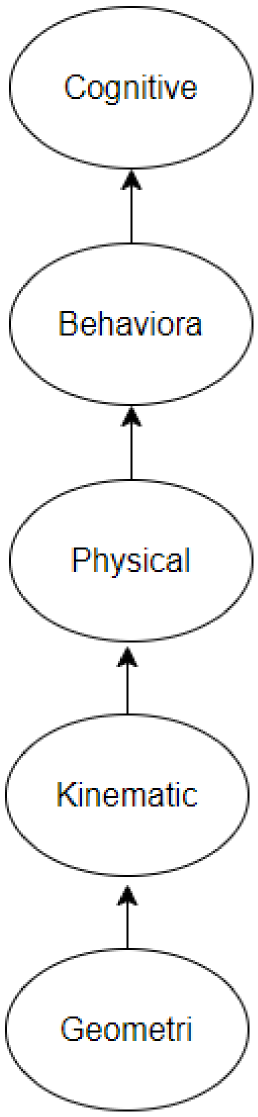
- The handwriting recognition software reads the text written on paper by a pen or on screen by a stylus. It can recognize the shapes of the letters and convert it into editable text.

8. **Intelligent Robots**

- Robots are able to perform the tasks given by a human. They have sensors to detect physical data from the real world such as light, heat, temperature, movement, sound, bump, and presure. They have efficient processors, multiple sensors and huge memory, to exhibit intelligence. In addition, they are capable of learning from their mistakes and the can adapt to the new environment.

1.7 Cognitive Modeling: Simulating Human Thinking Procedure

Cognitive modeling is basically the field of study within computer science that edals with the study and simulating the thinking process of human beings. The main task of AI is to make machine think like human. The most important feature of human thinking process is problem solving. That is why more or less cognitive modeling tries to undestand how humans can solve the problems. After that this model can be used for various AI applications such as machine learning, robotics, natural language processing, etc. Following is the diagram of different thinking levels of human brain:



Different thinking levels

1.8 Agent & Environment

In this section, we will focus on the agent and environment and how these help in Artificial Intelligence.

1.8.1 Agent

An agent is anything that can perceive its environment through sensors and act upon that environment through effectors.

- A **human agent** has sensory organs such as eyes, ears, nose, tongue and skin parallel to the sensors, and other organs such as hand, legs, mouth, for effectors.
- A **robotic agent** replaces cameras and intrared range finders for the sensors, and various motors and actuators for effectors.
- A **software agent** has encoded bit strings as its programs and actions.

1.8.2 Environment

Some programs operate in an entirely **artificial environment** confined to keyboard input, database, computer file systems and character output on a screen.

In contrast, some software agents (software robots or softbots) exist in rich, unlimited softbots domains. The simulator has a **very detailed, complex environment**. The software agent needs to choose from a long array of actions in real time. A softbot is designed to scan online preferences of the customer and shows interesting items to the customer works in the **real** as well as an **artificial** environment.

2 Getting Started

In this chapter, we will learn how to get started with Python. We will also undestand how Python helps for Artificial Intelligence.

2.1 Why Python for AI

Artificial intelligence is considered to be the trending technology of the future. Already there are a number of applications made on it. Due to this, many companies and researchers are taking interest in it. But the main question that arises here is that in which programming language can these AI applications be developed? There are various programming languages like Lisp, Prolog, C++, Java and Python, which can be used for developing applications of AI. Among them, Python programming language gains a huge popularity and the reasons are as follows:

5. **Simple syntax & less coding**

- Python involves very less coding and simple syntax among other programming languages which can be used for developing AI applications. Due to the feature, the testing can be easier and we can focus more on programming.

6. **Inbuilt libaries for AI projects**

- A major advantage for using Python for AI is that it comes with inbuilt libararies. Python has libaries for almost all kinds of AI projects. For example, **NumPy**, **SciPy**, **matplotlib**, **nltk**, **SimpleAI** are some the important inbuilt libraries of Python.
  - **Open Source** - Python is an open source programming language. This makes it widely popular in the community.
  - **Can be used for broad range of programming** - Python can be used for a broad range of programming tasks like small shell script to enterprise web applications. This is another reason Python is suitable for AI projects.

2.2 Features of Python

Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and it has fewer syntactical constructions than other languages. Python's features include the following:

- **Easy-to-learn** - Python was few keywords, simple structure, and a clearly defined syntax. This allows the student to pick up the language quickly.
- **Easy-to-read** - Python code is more clearly defined and visible to the eyes.
- **Easy-to-maintain** - Python's source code is fairly easy-to-maintain.
- **A broad standard libary** - Python's bulk of the libary is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.
- **Interactive Mode** - Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.
- **Portable** - Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
- **Extendable** - We can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.
- **Databases** - Python provides interfaces to all major commercial databases.
- **GUI Programming** - Python supports GUI applications that can be created and ported to many system calls, libaries and windows systems, such as Windows MFC, Macintosh, and the X Windows system of Unix.
- **Scalable** - Python provides a better structure and support for large programs than shell scripting.

2.2.1 Important features of Python

Let us now consider the following important features of Python:

- It supports functional and structured programming methods as well as OOP.
- It can be used as a scripting language or can be compiled to byte-code for building large applications.
- It provides very high-level dynamic data types and supports dynamic type checking.
- It supports automatic garbage collection.
- It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java.

2.3 Installing Python

Python distribution is available for a large number of platforms. You need to download only the binary code applicable for your platform and install Python.

If the binary code for your platform is not available, you need a C compiler to compile the source code manually. Compiling the source code offers more flexibility in term of choice of features that you require in your installation. Here is a quick overview of features that you require in your installation.

Here is quick overview of installing Python on various platforms:

6. **Unix and Linux installation**

- Follow these steps to install Python on Unix/Linux machine.
  - Open a Web browser and go to https://www.python.org/download
  - Follow the link to download zipped source code available for Unix/Linux.
  - Download and extract files.
  - Editing the *Modules/Setup* file if you want to customize some options.
  - run ./configure script
  - make
  - make install
- This installs Python at the standard location */usr/local/bin* and its libraries at */usr/local/lib/pythonXX* where XX is the version of Python.

7. **Windows Installation**

- Follow these steps to install Python on Windows machine.
- Follow the link for the Windows installer *python-XYZ.msi* file where XYZ is the version you need to install.
- To use this installer *python-XYZ.msi*, the Windows system must support Microsoft Installer 2.0. Save the installer file to your local machine and then run it to find out if your machine supports MSI..
- Run the downloaded file. This brings up the Python install wizard, which is really easy to use. Just accept the default settings and wait until the install is finished.

6. **Macintosh Installation**

- If you are on MAC OS X, it is recommended that you use Homebrew to install Python 3. It is a great package installer for Mac OS X and it is really easy to use. If you don't have Homebrew, you can install it using the following command:

```
$ ruby -e "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

We can update the package manager with the command below:

```
$ brew update
```

Now run the following command to install Python 3 on your system:

```
$ brew install python3
```

2.4 Setting up PATH

Programs and other executable files can be in many directories, so operating systems provide a search path that lists the directories that the OS searches for executables.

The path is stored in an environment variable, which is a named string maintained by the operating system. This variable contains information available to the command shell and other programs.

The path variable is named as PATH in Unix or Path in Windows (Unix is case-sensitive, Windows is not).

In Mac OS, the installer handles the path details. To invoke the Python interpreter from any particular directory, you must add the Python directory to your path.

2.4.1 Setting Path at Unix/Linux

To add the Python directory to the past for a particular session in Unix:

- In the csh shell

```
setenv PATH "$PATH:/usr/local/bin/python"
```

- In the bash shell (Linux)

```
export ATH = "$PATH:/usr/local/bin/python"
```

- In the sh or ksh shell

```
PATH = "$PATH:/usr/local/bin/python"
```

Note: /usr/local/bin/python is the path of the Python directory.

2.4.2 Setting a Path at Windows

To add the Python directory to the path for a particular session in Windows:

- At the command prompt

```
path %path%;C:\Python
```

Note: C:is the path of the Python directory.

2.5 Running Python

Let us now see the different ways to run Python. The ways are described below:

2.5.1 Interective Interpreter

We can start Python from Unix, DOS, or any other system that provides you a command-line interpreter or shell window.

- Enter **python** at the command line.
- Start coding right away in the interactive interpreter.

```
$python # Unix/Linux
```

Or

```
python% # Unix/Linux
```

Or

```
C:> python # Windows/DOS
```

Here is the list of all the available command line options:

S.No.	Option & Description
1	**-d** It provides debug output.
2	**-o** It generates optimized bytecode (resulting in .pyo files)
3	**-s** Do not run import site to look for Python paths on startup.
4	**-v** Verbose output (detailed trace on import statements).
5	**-x** Disables class-based built-in exceptions (just use strings), obsolete starting with version 1.6
6	**-c cmd** Runs Python script sent in as cmd string.
7	**File** Run Python script from given file.

2.6 Script from the Command-line

A Python script can be executed at the command line by invoking the interpreter on your application, as in the following:

```
$python script.py # Unix/Linux
```

or

```
python% script.py # Unix/Linux
```

or,

```
C:> python script.py # Windows/DOS
```

Note: Be sure the file permission mode allows execution.

2.7 Integrated Development Environment

You can run Python from a Graphical User Interface (GUI) environment as well, if you have a GUI application on your system that support Python.

- **Unix** - IDLE is the very first Unix IDE for Python.
- **Windows** - PythonWin is the first Windows interface for Python and is an IDE with a GUI.
- **Macintosh** - The Macintosh version of Python along with the IDLE IDE is available from the main website, downloadable as either MacBinary or BinHex'd files.

If you are not able to set up the environment properly, then you can take help from your system admin. Make sure the Python environment is properly set up and working perfectly fine.

We can also use another Python platform called Anaconda. It includes hundreds of popular data science packages and the conda package and virtual environment manager for Windows, Linux, and MacOS. You can download it as per your operating system from the link https://www.anaconda.com/download/.

For this tutorial we are using Python 3.6.3 version on MS Windows.

3 Machine Learning

Learning means the acquisition of knowledge or skills through study or experience. Based on this, we can define machine learning (ML) as follows:

It may be defined as the field of computer science, more specifically an application of artificial intelligence, which provides computer systems the ability to learn with data and improve from experience without being explicitly programmed.

Basically, the main focus of machine learning is to allow the computers learn automatically without human intervention. Now the question arises that how such learning can be started and done? It can be started with the observation of data. The data can be some examples, instruction or some direct experiences too. Then on the basis of this input, machine makes better decisions by looking for some patterns in data.

3.1 Types of Machine Learning (ML)

Machine Learning Algorithms helps computer system learn without being explicitly programmed. These algorithms are categorized into supervised or unsupervised. Let us now see a few algorithms:

3.1.1 Supervised machine learning algorithms

This is the most commonly used machine learning algorithm. It is called supervised because the process of algorithm learning from the training dataset can be thought of as a teacher supervising the learning process. In this kind of ML algorithm, the possibile outcomes are already known and training data is also labeled with correct answers. It can be understood as follows:

Suppose we have input variables **x** and an output variable **y** and we applied an algorithm to learn the mapping function from the input to output such as:

```
Y = f(x)
```

Now, the main goal is to approximate the mapping function so well that when we have new input data (x), we can predict the output variable (Y) for that data.

Mainly supervised learning problems can be divided into the following two kinds of problems:

- **Classification** - A problem is called classification problem when we have the categorized output such as “black”, “teaching”, “non-teaching”, etc.
- **Regression** - A problem is called regression problem when we have the real value output such as “distance”, “kilogram”, etc.

Decision tree, random forest, knn, logistic regression are the examples of supervised machine learning algorithms.

3.1.2 Unsupervised machine learning algorithms

As the name suggests, these kind of machine learning algorithms do not have any supervisor to provide any sort of guidance. That is why unsupervised machine learning algorithms are closely aligned with what some call true artificial intelligence. It can be understood as follows:

Suppose we have input variable x, then there will be no corresponding output variables as there is in supervised learning algorithms.

In simple words, we can say that in unsupervised learning there will be no correct answer and no teacher for the guidance. Algorithms help to discover interesting patterns in data.

Unsupervised learning problems can be divided into the following two kinds of problems:

- **Clustering** - In clustering problems, we need to discover the inherent groupings in the data. For example, grouping customers by their purchasing behavior.
- **Association** - A problem is called association problem because such kinds of problem require discovering the rules that describe large portions of our data. For example, finding the customers who buy both **x** and **y**.

K-means for clustering, Apriori algorith, for association are the examples of unsupervised machine learning algorithms.

3.1.2.1 Reinforcement machine learning algorithms

These kinds of machine learning algorithms are used very less. These algorithms train the system to make specific decisions. Basically, the machine is exposed to an environment where it trains itself continually using the trial and error method. These algorithms learn from past experience and tries to capture the best possible knowledge to make accurate decisions. Markov Decision Process is an example of reinforcement machine learning algorithms.

3.2 Most Common Machine Learning Algorithms

In this section, we will learn about the most common machine learning algorithms. The algorithms are described below:

3.2.1 Linear Regression

It is one of the most well-known algorithms in statistics and machine learning.

Basic concept - Mainly linear regression is a linear model that assumes a linear relationship between the input variables say x and the single output variable say y. In other words, we can say that y can be calculated from a linear combination of the input variables x. The relationship between variables can be established by fitting a best line.

3.2.1.1 Types of Linear Regression

Linear regression is of the following two types:

- **Simple linear regression**. A linear regression algorithm is called simple linear regression if it is having only one independent variable.
- **Multiple linear regression**. A linear regression algorithm is called multiple linear regression if it is having more than one independent variable.

Linear regression is mainly used to estimate the real values based on continuous variable(s). For example, the total sale of a shop in a day, based on real values, can be estimated by linear regression.

3.2.2 Logic Regression

It is a classification algorithm and also known as **logit** regression.

Mainly logistic regression is a classification algorithm that is used to estimate the discrete values like 0 or 1, true or false, yes or no based on given set of independent variable. Basically, it predicts the probability hence its output lies in between 0 and 1.

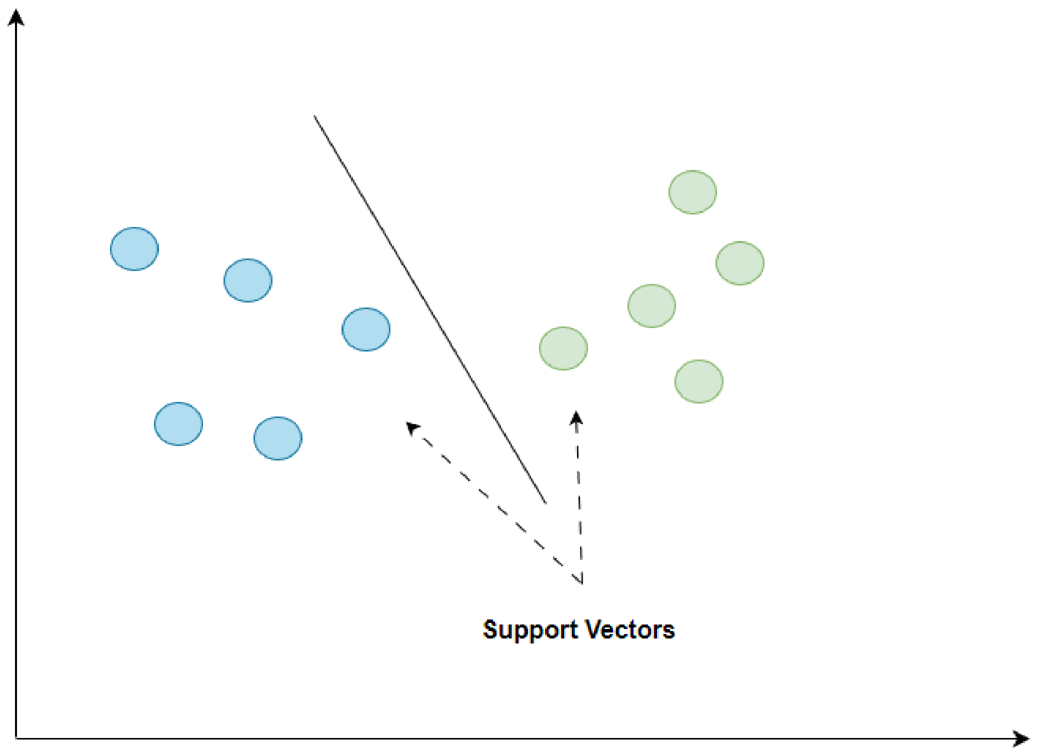
3.2.3 Decision Tree

Decision tree is a supervised learning algorithm that is mostly used for classification problems.

Basically it is a classifier expressed as recursive partition based on the independent variables. Decision tree has nodes which form the rooted tree. Rooted tree is a direccted tree with a node called “root”. Root does not have any incoming edges and all the other nodes have one incoming edge. These nodes are called leaves or decision nodes. For example, consider the following decision tree to see whether a person is fit or not.

3.2.4 Support Vector Machine (SVM)

It is used for both classification and regression problems. But mainly it is used for classification problems. The main concept of SVM is to plot each data item as a point in n-dimensional space with the value of each feature being the value of a particular coordinate. Here n would be the features we would have. Following is a simple graphical representation to undestand the concept of SVM.



Vectors

In the above diagram, we have two features hence we first need to plot these two variables in two dimensional space where each point has two coordinates, called support vectors. The line splits the data into two different classified groups. This line would be the classifier.

3.2.5 Naive Bayes

It is also a classification technique. The logic behind this classification technique is to use Bayes theorem for building classifiers. The assumption is that the predictors are independent. In simple words, it assumes that the presence of a particular feature in a class in unrelated to the presence of any other feature. Below is the equation for Bayes theorem:

```
$$P\left (\frac{A}{B}\right) = \frac{P\left (\frac{B}{A}\right)P\left (A
\right)}{P\left (B \right)}$$
```

The Naive Bayes model is easy to build and particularly useful for large data sets.

3.2.6 K-Nearest Neighbors (KNN)

It is used for both classification and regression of the problems. It is widely used to solve classification problems. The main concept of the algorithm is that it used to store all the available cases and classifies new cases by majority votes of its k neighbors. The case being then assigned to the class which is the most common amongst its K-nearest neighbors, measured by a distance function. The distance function can be Euclidean, Minkowski and Hamming distance. Consider the following to use KNN:

- Computationally KNN are expensive than other algorithms used for classification problems.
- The normalization of variables needed otherwise higher range variables can bias it.
- In KNN, we need to work on pre-processing stage like noise removal.

3.2.7 K-Means Clustering

As the name suggests, it is used to solve clustering problems. It is basically a type of unsupervised learning. The main logic of K-Means clustering algorithm is to classify the data set through a number of clusters. Follow these stpes to form clusters by K-means:

- K-means picks k number of points for each cluster known as centroids.
- Now each data point forms a cluster with the closest centroids, i.e., k clusters.
- Now, it will find the centroids each cluster based on the existing cluster members.
- We need to repeat these steps until convergence occurs.

3.2.8 Random Forest

It is a supervised classification algorithm. The advantage of random forest algorithm is that it can be used for both classification and regression kind of problems. Basically it is the collection of decision trees (i.e., forest) or you can say ensemble of the decision trees. The basic concept of random forest is that each tree gives a classification and the forest chooses the best classifications from them. Followings are the advantages of Random Forest algorithm:

- Random forest clasifier can be used for both classification and regression tasks.
- They can handle the missing values.
- It won't over fir the model even if we have more number of trees in the forest.

4 Data Preparation

We have already studied supervised as well as unsupervised machine learning algorithms. These algorithms require formatted data to start the training process. We must prepare or format data in a certain way so that it can be supplied as an input to ML algorithms.

This chapter focuses on data preparation for machine learning algorithms.

4.1 Preprocessing the Data

In our daily life, we deal with lots of data but this data is in raw form. To provide the data as the input of machine learning lagorithms, we need to convert it into a meaningful data. That is where data preprocessing comes into picture. In other simple words, we can say that before providing the dta to the machine learning algorithms w need to preprocess the data.

4.1.1 Data preprocessing steps

Follow these steps to preprocess the data in Python:

Step 1 - Importing the useful packages - If we are using Python then this would be the first step for converting the data into a certain format, i.e., preprocessing it can be done as follows:

```
import numpy as np
import sklearn.preprocessing
```

Have we have used the following two packages:

- **NumPy** - Basically NumPy is a general purpose array-processing package designed to efficiently manipulate large multi-dimentional arrays of arbitrary records without sacrificing to much speed for small multi-dimentional arrays.
- **Sklearn.preprocessing** - This package provides many common utility functions and transformer classes to change raw feature vectors into a representaion that is more suitable for machine learning algorithms.

Step 2 - Defining sample data - After importing the packages, we need to define some sample data so that we can apply preprocessing techniques on that data. We will now define the following sample data:

```
Input_data = np.array([2.1,  -1.9,  5.5],
                      [-1.5,  2.4   3.5],
                      [0.5,  -7.9,  5.6],
                      [5.9,   2.3, -5.8],)
```

Step3 - Applying preprocessing technique - In this step, we need to applyany of the preprocessing techniques.

The following section describes the data preprocessing techniques.

4.2 Techniques for Data Preprocessing

The techniques for data preprocessing are described below:

4.2.1 Binarization

This is the preprocessing technique which is used when we need to convert our numerical values into Boolean values. We can use an inbuilt method to binarize the input data say by using 0.5 as the threshold value in the following way:

```
data_binarized = preprocessing.Binarizer(threshold = 0.5).transform(imput)
print("\nBinarized data:\n", data_binarized)
```

Now after running the above code we will get the following output, all the values above 0.5(threshold values) would be converter to 1 and al the values below 0.5 would be converted to 0.

4.2.1.1 Binarized data

```
[[ 1.  0.  1.]
 [ 0.  1.  1.]
 [ 0.  0.  1.]
 [ 1.  1.  0.]]
```

4.2.2 Mean Removal

It is another very common preprocessing technique that is used in machine learning. Basically it is used to eliminate the mean from feature vector so that every feature is centered on zero. We can also remove the bias from the feature in the feature vector. For applying mean removal preprocessing technique on the sample data, we can write the Python code shown below. The code will display the Mean and Standard deviation of the input data:

```
print("Mean = ", input_data.mean(axis = 0))
print("Std deviation = ", input_)data.std(axis = 0)
```

We will get the following output after running the above line of code:

```
         Mean = [1.75      -1.275      2.2]
Std deviation = [2.71431391 4.20022321 4.69414529]
```

Now, the code below will remove the Mean and Standard deviation of the input data:

```
data_scaled = preprocessing.scale(input_data)
print("Mean =", data_scaled.mean(axis=0))
print("Std deviation =", data_scaled.std(axis = 0))
```

We will get the following output after running the above lines of code:

```
         Mean = [1.110022302e-16 0.0000000e+00 0.0000000e+00]
Std deviation = [1.             1.            1.]
```

4.2.3 Scaling

It is another data preprocessing technique that is used to scale the feature vectors. Scaling of feature vectors is needed because the values of every feature can vary between many random values. In other words we can say that scaling is important because we do not want any feature to be synthetically large or small. With the help of the following Python code, we can do the scalling of our input data, i.e. feature vector:

4.2.3.1 Min max scaling

```
data_scaler_minmax = preprocessing.MinMaxScaler(feature_range=(0,1))
data_scaled_minmax = data_scaler_minmax.fit_transform(input_data)
print ("\nMin max scaled data:\n", data_scaled_minmax)
```

We will get the following output after running the above lines of code:

4.2.3.2 Min max scaled data

```
[ [ 0.48648649  0.58252427  0.99122807]
  [ 0.          1.          0.81578947]
  [ 0.27027027  0.          1.        ]
  [ 1.          0.99029126  0.        ] ]
```

4.2.4 Normalization

It is another data preprocessing technique that is used to modify the feature vectors. Such kind of modification is necessary to measure the feature vectors on a common scale. Followings are two types of normalization which can be used in machine learning:

4.2.4.1 L1 Normalization

It is also referred to as **Least Absolute Deviations**. This kind of normalization modifies the values so that the sum of the abolute values is always up to 1 in each row it can be implemented on the input data with the help of the following Python code:

```
# Normalize data
data_normalized_l1 = preprocessing.normalize(input_data, norm ='l1')
print("\nL1 normalized data:\n", data_normalized_l1)
```

The above line of code generates the following output &miuns;

```
L1 normalized data:
[[  0.22105236   -0.2           0.57894737]
 [ -0.2027027     0.342432432   0.47297297]
 [  0.03571429   -0.54628571    0.4        ]
 [  0.42142857    0.16428571   -0.41428571]]
```

4.2.4.2 L2 Normalization

It is also referred to as **least squares**. This kind of normalization modifies the vlues so that the sum of the squares is always up to 1 in each row. It can be implemented on the input data with the help of the following Python code:

```
# Normalize data
data_normalized_l2 = preprocessing.normalize(input_data, norm = '12')
print("\nL2 normalized data :\n", data_normalized_l2)
```

The above line of code will generate the following output:

```
L2 normalized data:
[[ 0.33946114   -0.30713151     0.88906489]
 [-0.33325106    0.53320169     0.7775858 ]
 [ 0.05156558   -0.81473612     0.57753446]
 [ 0.68706914    0.26784051    -0.6754239 ]]
```

4.2.5 Labeling the data

We already know that data in acertain format is necessary for machine learning algorithms. Another important requirement is that the data must be labelled properly before sending it as the input of machine learning algorithms. For example, if we talk about classification, there are lot of labels on the data. Those labels are in the form of words, numbers, etc. Functions related to machine learning in **sklearn** expect that the data much have number labels. Hence, if the data is in other form then it must be converted to numbers. The process of transforming the word labels into numerical form is called lable encoding.

4.2.5.1 Label encoding steps

Follow these steps for encoding the data labels in Python.

Step 1 - Importing the useful packages If we are using Python then this would be first step for converting the data into certain format, i.e., preprocessing. It can be done as follows:

```
import numpy as np
from sklearn import preprocessing
```

Step 2- Defining sample labels After importing the packages, we need to define some sample labels so that we can create and train the label encoder. We will now define the following sample labels:

```
# Sample_input labels
input_labels = ['red', 'black', 'green', 'black', 'yellow', 'white']
```

**Step 3 - Creating & training label encoder object In this step, we need to create the label encoder and train it. The following Python code will help in doing this:

```
# Creating the label encoder
encoder = preprocessing.labelEncoder()
encoder.fit(input_labels)
```

Following would be the output after running the above Python code:

```
LabelEncoder()
```

Step 4 - Checking the performance by encoding random order list This step can be used to check the performance by encoding the random ordered list. Following Python code can be written to do the same:

```
# encoding a set of labels
text_labels = ['green', 'red', 'black']
encoded_values = encoder.transform(test_labels)
print("\nLabels = ", test_labels)
```

The labels would get printed as follows:

```
Labels = ['green', 'red', 'black']
```

Now, can we get the list of encoded values i.e. word labels converted to numbers as follows:

```
print("Encoded values =", list(envoded_values))
```

The encoded values would get printed as follows

```
Encoded values = [1, 2, 0]
```

Step 5 - Checking the performance by decoding a random set of numbers This step can be used to check the performance by decoding the random set of numbers. Following Python code can be written to do same:

```
# decoding a set of values
endoded_values = [3, 0, 4, 1]
decoded_list = encoder.inverse_transform(encoded_values)
print("\nEncoded values =", encoded values)
```

Now, Encoded values would get printed as follows:

```
Encoded values = [3, 0, 4, 1]
print("\nDecoded lavels =", list(decoded_list))
```

Now, decoded values would get printed as follows:

```
Decoded labels = ['white', 'black', 'yellow', 'green']
```

4.2.5.2 Laveled v/s Unlabeled Data

Unlabeled data mainly consists of the sample of natural or human-created object that can easily be obtained from the world. They include, audio, video, photos, news articles, etc.

On the other hand, labeled data takes a set of unlabeled data and augments each piece of that unlabeled data with some tag or label or class that is meaningful. For example, if we have a photo then the label can be put based on the content of the photo, i.e., it is photo of boy or a girl or animal or anthing else Labeling the data needs human expertise or judgement about a given piece ofunlabeled data.

There are many scenarios where unlabeled data is plentiful and easily obtained but labeled data often requires a human/expert to annotate. Semi-supervised learning attempts to combine labeled and unlabeled data to buld better moodels.

4.3 Classification

In this chapter, we will focus on implemeting supervised learning - clasiification.

The clasiffication technique or model attempts to get some conclusion from observed values. In classification problem, we have the categorized output such as “Black” or “white” or “Teaching” and “Non-Teaching”. While building the classification model, we need to have training dataset that contrains data points and the corresponding labels. For example, if we want to check whether the image is of a car or not. For cheking this, we will build a training dataset having the two classes related to “car” and “no car”. Then we need to train the model by using training samples. The classification models are mainly used in face recognition, spam identification, etc.

4.3.1 Steps for Building a Classifier in Python

For building a classifier in Python, we are going to use Python 3 and Scikit-learn which is a tool for machine learning. Follow these steps to build a classifier in Python:

Step 1 - Import Scikit-learn This would be very first step for building a classifier in Python. In this step, we will install a Pyhon package called Scikit-learn which is one of the best machine learning modules in Python. The following command will help us import he package:

```
Import Sklearn
```

Step 2 - Import Scikit-learn's dataset In this step, we can begin working with the dataset for our machine learning model. Here, we are going to use **the** Breast Cancer Wisconsin Diagnostic Database. The dataset includes various information about breast cancer tumors, as well as classification label of **malignat** or **benign**. The dataset has 569 instances, or data, on 569 tumors and includes information on 30 attributs, or features, such as the radius of the tumor, texture, smoothness, and are. With the help of the following command, we can import the Scikit-learn's breast cancer dataset:

```
from sklearn.datasets import load_breast_cancer
```

Now, the following command will lload the dataset.

```
data = load_breast_cancer()
```

Following is a list of important dictionary keys:

- Classification label names(target_names)
- The actual labels(target)
- The attribute/feature names(feature_names)
- The attribute(data)

Now, with the help of the following command, we can create new variables for each important set of information and assign the data in other words, we can organize the data with the following command:

```
label_names = data['target_names']
labels = data['target']
feature_names = data['feature_names']
features = data ['data']
```

Now, to make it clearer we can print the class labels, the first data instance's label, our feature names and the feature's value with the help of the following commands:

```
print(label_names)
```

The above command will print the class names which are malignant and benign respectively. It is shown as the output below:

```
['malignant' 'benign']
```

Now, the command below will show that they are mapped to binary values 0 and 1. Here 0 represents malignant cancer and 1 represents benign cancer. You will receive the following output:

```
print(labels[0])
0
```

The two commands given below will produce the feature names and fature values.

```
print(feature_names[0])
mean radius
print (features[0])
[ 1.79900000e+01 1.03800000e+01 1.22800000e+02 1.00100000e+03
  1.18400000e-01 2.77600000e-01 3.00100000e-01 1.47100000e-01
  2.41900000e-01 7.87100000e-02 1.09500000e+00 9.05300000e-02
  8.58900000e+00 1.53400000e+02 6.39900000e-03 4.90400000e-02
  5.37300000e-02 1.58700000e-02 3.00300000e-02 6.19300000e+03
  2.53800000e+01 1.73300000e+01 1.84600000e+02 2.01900000e+03
  1.22000000e-01 6.65600000e-01 7.11900000e-01 2.65400000e-01
  4.60100000e-01 1.18900000e-01]
```

From the above output, we can see that the first data instance is a malignant tumor the radius of which is 1.7990000e+01.

Step 3 - Organizing data into sets In this step, we will divide our data into two parts namely a traning aset and a test set. Spliting the data into these sets is very important because we have to test our model on the unseen data. To split the data into sets, sklearn has a function called the **train_test_split()** function. With the help of the following commands, we can split the data in thease sets:

```
from sklearn.model_selection import train_test_split
```

The above command will import the **train_test_split** function from sklearn and the command below will split the data into training and test data. In the example given below, we are using 40 % of the data for testing and remaining data would be used for training the model.

```
train, test, train_labels, test_labels = train_test_split(features)
```

Step 4 - Building the model In this step, we will building our model. We are going to use Naive Bayes algorithm for building the model. Following commands can be used to build the model:

```
from sklearn.naive_bayes import GaussianNB
```

The above command will import the GaussianNB module. Now, the following command will help you initialize the model.

```
gnb = GaussianNB()
```

We will train the model by filtering it to the data using gnb.fit()

```
model = gnb.fit(train, train_labels)
```

Step 5 - Evaluating the model and its accuracy In this steps, we are going to evaluate the model by making prediction on our test data. Then we will find out its accuracy also. For making predictions, we will use the predict() function. The following command will help you do this:

```
preds = gnb.predict(test)
print(preds)
[1 0 0 1 1 0 0 0 1 1 1 0 1 0 1 0 1 1 1 0 1 1 0 1 1 1 1 1 1
 0 1 1 1 1 1 1 1 0 1 0 1 1 0 1 1 1 1 1 1 1 1 0 0 1 1 1 1 0
 0 1 1 0 0 1 1 1 0 0 1 1 0 0 1 0 1 1 1 1 1 1 0 1 1 0 0 0 0
 0 1 1 1 1 1 1 1 1 0 0 1 0 0 1 0 0 1 1 1 0 1 1 0 1 1 0 0 0
 1 1 1 0 0 1 1 0 1 0 0 1 1 0 0 0 1 1 1 0 1 1 0 0 1 0 1 1 0
 1 0 0 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0
 1 1 0 1 1 1 1 1 1 0 0 0 1 1 0 1 0 1 1 1 1 0 1 1 0 1 1 1 0
 1 0 0 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 0 1 0 0 1 1 0 1]
```

The above series of 0s and 1s are the predicted values for the tumor classes - malignant and benign.

Now, by comparing the two arrays namely **test_labels** and **preds**, we can find out the accuracy of our model. We are going to use the **accuracy_score()** function to determine the accuracy. Consider the following command for this:

```
from sklearn.metrics import accuracy_score
print(accuracy_score(test_labels.preds))
0.951754385965
```

The result shows that the NaiveBayes classifier is 95.17% accurate.

In this way, with the help of the above steps we can build our classifier in Python.

4.3.2 Building Classifier in Python

In this section, we will learn how to build a classifier in Python.

4.3.2.1 Naive Bayes Classifier

Naive Bayes is a classification technique used to build classifier using the Bayes theorem. The assumption is that the predictors are independent. In simple words, it assumes that the presence of a particular feature in a class in unrelated to the presence of any other feature. For building Naive Bayes classifier we need to use the python libary called scikit learn. There are three types of Naive Bayes models named **Gaussian, Multinomial, and BBernoulli** under scikit learn package.

To build a Naive Bayes machine learning classifier model, we need the following & minus.

4.3.2.2 Dataset

We are going to use the dataset named Breast Cance Wisconsin Diagnostic Database. The dataset includes various information about breast cancer tumors, as well asa classification labels of **malignant** or **benign**. The dataset has 569 instances, or data, on 569 tumors and includes information on 30 attributes, or features, such as the radius of the tumor, texture, smoothness, and are. We can import this dataset from sklearn package.

4.3.2.3 Nalve Bayes Model

For building Naive Bayes classifier, we need Naive Bayes model. As told earlier, there are three types of Naive Bayes models named **Gaussian, Multinomial** and **Bernoulli** under scikit learn package. Here, in the following example we are going to use the Gaussian Naive Bayes model.

By using the above, we are going to build a Naive Bayes machine learning model to use the tumor information to predict whether or not a tumor is malignant or benign.

To begin with, we need to install the sklearn module. It can be done with thehelp of the following command:

```
Import Sklearn
```

Now, we need to import the dataset named Breast Cancer Wisconsin Diagnostic Database.

```
from sklearn.datasets import load_breast_cancer
```

Now, the following command will load the dataset.

```
data = load_breast_cancer()
```

The data can be organized as follows:

```
label_names = data ['target_names']
labels = data['target']
feature_names = data['faeture_names']
features = data['data']
```

Now, to make it cleare we can print the class labels, the first data instance label, our feature names and the features value with the help of the following command:

```
print(label_names)
```

The above command will print the class names which are malignant and benign respectively. It is show as the output below:

```
['malignant' 'benign']
```

Now, the command given below will show that they are mapped to binary values 0 and 1. Here 0 represents malignant cancer and 1 represents benign cancer. It is shown as the output below:

```
print(labels[0])
0
```

The following two commands will produce the feature names and feature values:

```
print(feature_names[0])
mean radius
print(features[0])
[ 1.79900000e+01 1.03800000e+01 1.22800000e+02 1.00100000e+03
  1.18400000e-01 2.77600000e-01 3.00100000e-01 1.47100000e-01
  2.41900000e-01 7.87100000e-02 1.09500000e+00 9.05300000e-02
  8.58900000e+00 1.53400000e+02 6.39900000e-03 4.90400000e-02
  5.37300000e-02 1.58700000e-02 3.00300000e-02 6.19300000e+03
  2.53800000e+01 1.73300000e+01 1.84600000e+02 2.01900000e+03
  1.22000000e-01 6.65600000e-01 7.11900000e-01 2.65400000e-01
  4.60100000e-01 1.18900000e-01
]
```

From the above output, we can see that the first data instance is a malignant tumor the main radius of which is 1.7990000e+01.

For testing our model on unseen data, we need to split our data into training and testing data. It can be done with the help of the following code:

```
from sklearn.model_selection import train_test_split
```

The above command will import the **train_test_split** function from sklearn and the command below will split the data into training and test data. In the below example, we are using 40 % of the data for testing and theremining data would be used for training the model.

```
train, test, train_labels, test_labels =
train_test_split (feature, labels, test_size = 0.40, random_state = 42)
```

Now, we are building the model with the following command:

```
from sklearn.naive_bayes import GaussianNB
```

The above command will import the GaussianNB module. Now, with the command given below we need to initialize the model.

```
gnb = GaussianNB()
```

We will train the model by fitting it to the data by using gnb.fit()

```
model = gnb.fit(train, train_labels)
```

Now, evaluate the model by making prediction on the test data and it can be done as follows:

```
preds = gnb.predict(test)
print(preds)
```

```
[1 0 0 1 1 0 0 0 1 1 1 0 1 0 1 0 1 1 1 0 1 1 0 1 1 1 1 1 1
 0 1 1 1 1 1 1 1 0 1 0 1 1 0 1 1 1 1 1 1 1 1 0 0 1 1 1 1 0
 0 1 1 0 0 1 1 1 0 0 1 1 0 0 1 0 1 1 1 1 1 1 0 1 1 0 0 0 0
 0 1 1 1 1 1 1 1 1 0 0 1 0 0 1 0 0 1 1 1 0 1 1 0 1 1 0 0 0
 1 1 1 0 0 1 1 0 1 0 0 1 1 0 0 0 1 1 1 0 1 1 0 0 1 0 1 1 0
 1 0 0 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0
 1 1 0 1 1 1 1 1 1 0 0 0 1 1 0 1 0 1 1 1 1 0 1 1 0 1 1 1 0
 1 0 0 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 0 1 0 0 1 1 0 1]
```

The above series of 0s and 1s are the predicted values for the tumor classes i.e. malignant and benign.

Now, by comparing the two arrays namely **test_labels** and **preds**, we can find out the accuracy of our model. We are going to use the **accuracy_score()** function to determine the accuracy. Consider the following command:

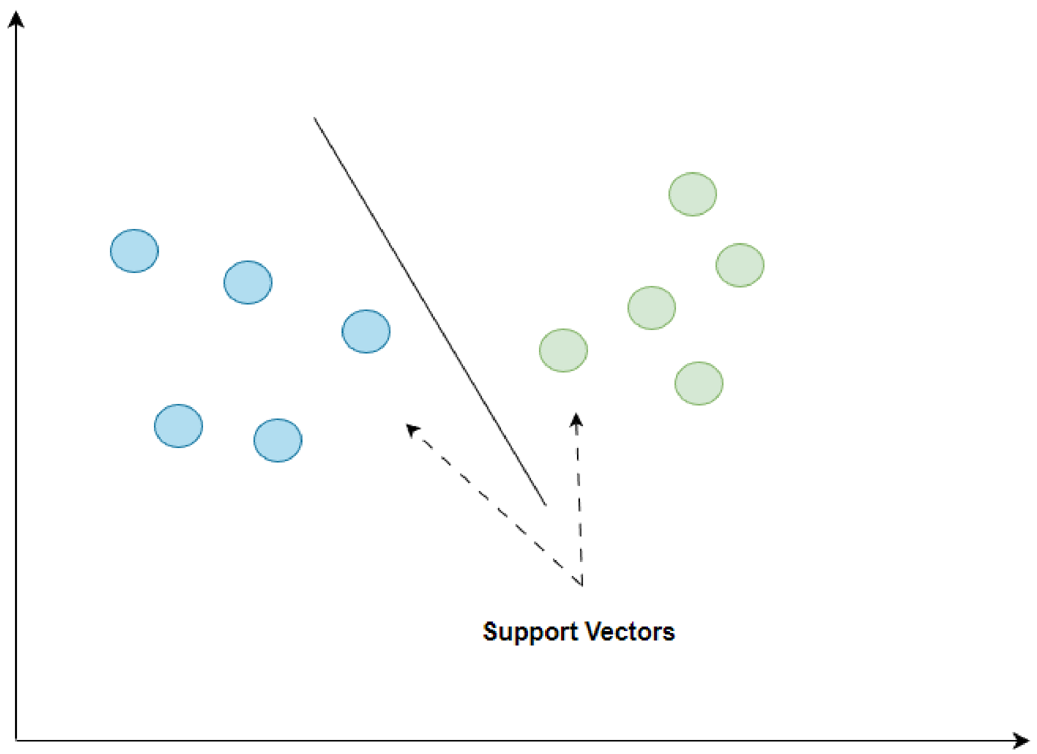
```
from sklearn.metrics import accuracy_score
print(accuracy_score(test_labels.preds))
0.951754385967
```

The result shows that NaiveBayes classifier is 95.17% accurate.

That was machine learning classifier based on Naive Bayse Gaussian model.

4.3.3 Support Vector Machine (SVM)

Basically, Support vector machine (SVM) is a supervised machine learning algorithm that can be used for both regression and classification. The main concept of SVM is to plot each data item as a point in n-dimensional space with the value of each feature being the value of a particular coordinate. Here n would be the features we would have. Following is a simple graphical representation to understand the concept of SVM:



Vectors

In the above diagram, we have two features. Hence, we first need to plot these two variables in two dimensional space where each point has two coordinates, called support vectors. The line splits the data into two different classified groups. This line would be the classifier.

Here, we are going to build an SVM classifier by using scikit-learn and iris dataset. Sciktlearn libary has the **sklearn.svm** module and provides sklearn.svm.svc for classification. The SVM classifier to predict the class of the iris plant based on 4 features are shown below.

4.3.4 Dataset

We will use the iris dataset which contains 3 classes of 50 instance each, where each class refers to a type of iris plant. Each instance has the four features namely sepal length, speal width, petal length and petal width. The SVM classifier to predict the class of the iris plant based on 4 features is shown below.

4.3.5 Dataset

We will use the iris dataset which contains 3 classes of 50 instances each, where each class refers to a type of iris plant. Each instance has the four features namely sepal length, sepal width, petal length and petal width. The SVM classifier to predict the class of the iris plant based on 4 features is shown below.

4.3.6 Kernel

It is a technique used by SVM. Basically these are the functions which take low-dimensional input space and transform it to a higher dimensional space. It converts non-separable problem to separable problem. The kernel function can be any on amoung linear, polynomial, rbf and sigmoid. In this example, we will use the linear kernel.

Let us now import the following packages:

```
import pandas as pd
import numpy as np
from sklearn import svm, datasets
import matplotlib.pyplot as plt
```

Now, load the input data:

```
iris = datasets.load_iris()
```

We are taking first two features:

```
x = iris.data [:, :2]
y = iris.target
```

We will plot support vector machine boundaries with original data. We are creating a mesh to plot.

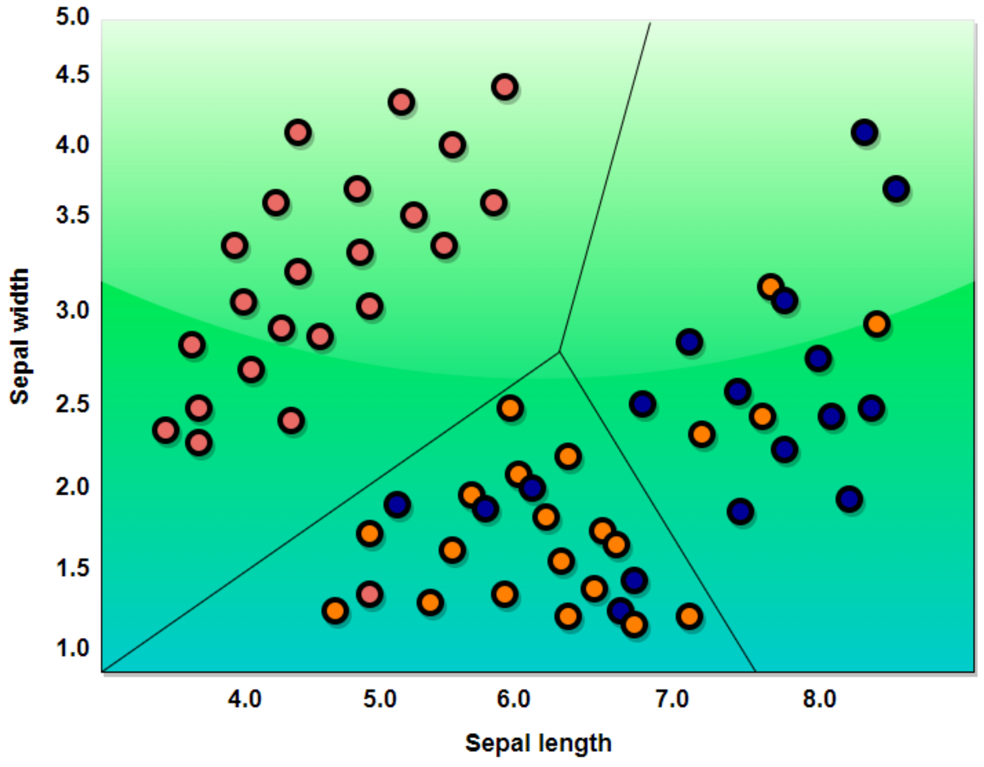
```
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1],max() + 1
h = (x_max / x_min)/100
xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
np.arange(y_min, y_max, h))
X_plot = np.c_[xx.ravel(), yy.ravel()]
```

We need to give the value of regularization parameter.

```
C = 1.0
```

We need to crate the SVM classifier object.

```
Svc_classifier = svm_classifier.SVC(kernel='liner',
C=C, decision_function_shape = 'ovr').fit(X, y)
Z = svc_classifier.predict(X_plot)
Z = Z.reshape(xx.shape)
plt.figure(figsize = (15, 5))
plt.subplot(121)
plt.contourf(xx, yy, Z, cmap = plt.cm.tab10, alpha = 0.3)
plt.scatter(X[:, 0], X[:, 1], c = y, cmap = plt.cm.Set1)
plt.xlabel['Sepal length']
plt.ylabel['Sepal width']
plt.xlim(xx.min(), xx.max())
plt.title('SVC with linear kernel')
```



SVC with linear

4.4 Logistic Regression

Basically, logistic regression model is one of the members of supervised classification algorithm famiily. Logistic regression measures the reletionship between dependent variables and independent vaiables by estimating the probabilities using a logistic function.

Here, if we talk about dependent and independent variables then dependent variable is the target class variable we are going to predict and on the other side the independent variables are the features we are going to use to predict the target class.

In logistic regression, estimating the probabilities means to predict the likelihood occurance of the event. For example, the shop owner would like to predict the customer who entered into the shop will buty the play station (for example) or not. There would be many features of customer - gender, age, etc. which would be observed by the shop keeper to predict the likelihood occurance, i.e., buying a play station or not. The logistic function is the sigmoid curve that is used to build the function with various parameters.

4.4.1 Prerequistes

Before building the classifier using logistic regression, we need to install the Tkinter package on our system, it can be installed from https://docs.python.org/2/libary/thinter.html

Now, with the help of the code given below, we can create a classifier using logistic regression:

First, we will import some packages:

```
import numpy as np
from sklearn import linear_model
import matplolib.pyplot as plt
```

Now, we need to define the sample data which can be done as follows:

```
x = np.array([[2, 4.8], [2.9, 4.7], [2.5, 5], [3.2, 5.5], [6, 5], [7.6, 4],
              [3.2, 0.9], [2.9, 1.9], [2.4, 3.5], [0.5, 3.4], [1, 4], [0.9,
5.9]])
y = np.array ([0, 0, 0, 1, 1, 1, 2, 2, 2, 3, 3, 3])
```

Next, we need to create the logistic regression classifier, which can be done as follows:

```
Classifier_LR = linear_model.logisticRegression(solver = 'liblinear', C = 75)
```

Last but not the least, we need to train this classifier:

```
Classifier_LR.fit(x, y)
```

Now, how we can visualize the output? It can be done by creating a function named Logistic_visualize() :

```
Def Logistic_visualize(Classifier_LR, X, y):
  min_x, max_x = X[:, 0].min() - 1.0, X[:, 0].max() + 1.0
  min_y, max_y = X[:, 1].min() - 1.0, X[:, 1].max() + 1.0
```

In the above line, we defined the minimum and maximum values X and Y to be used in mesh frid. In addition, we will define the step size for plotting the mesh grid.

```
mesh_step_size = 0.02
```

Let us define the mech grid of X and Y values as follows:

```
x_vals, y_vals = np.meshgrid(np.arrange(min_x), max_x, mesh_step_size),
                 np.arange(min_y, max_y, mesh_step_size)
```

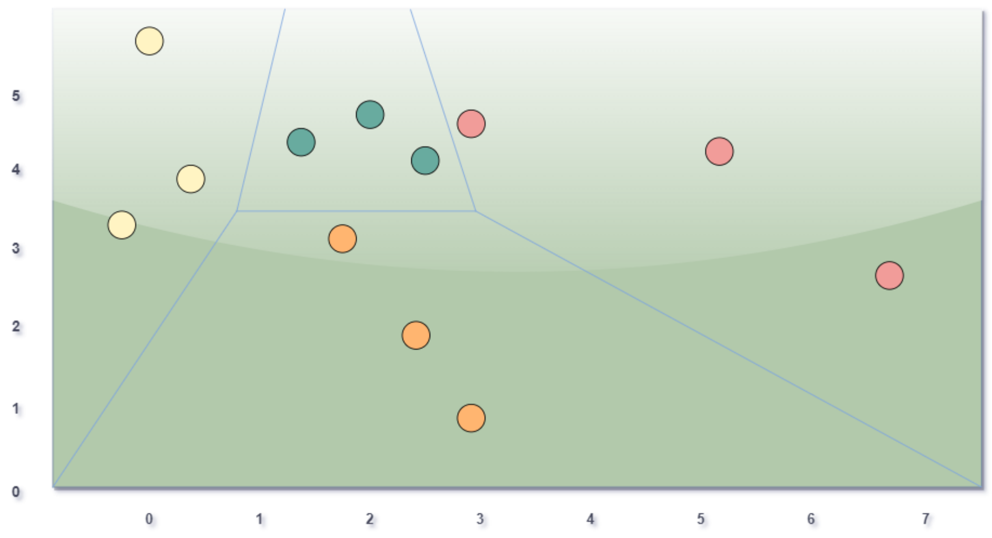
With the help of following code, we can run the classifier on the mesh grid:

```
output = classifier.predict(np.c_[x_vals.ravel(), y_vals.ravel()])
output = output.reshape(x_vals.shape)
plt.figure()
plt.pcolermesh(x_vals, y_vals, output, cmap = plt.cm.gray)
plt.scatter(x[:, 0], X[:, 1], c = y, s = y, s = 75, edgecolors = 'black',
linewidth=1, cmap = plt.cm.Paired)
```

The following line of code will specify the boundaries of the plot:

```
plt.xlim(x_vals.min(), x_vals.max())
plt.ylim(_vals.min(), y_vals.max())
plt.xticks((np.arrange(int(X[:, 0].min() - 1), int(X[:, 0].max() + 1), 1.0)))
plt.yticks((np.arrange(int(X[:, 1].min() - 1), int(X[:, 1].max() + 1), 1.0)))
plt.show()
```

Now, after running the code we will get the following output, logistic regression classifier:



Prerequisites

4.5 Decision Tree Classifier

A decision tree is basically a binary tree flowchart where each node splits a group of observations according to some feature variable.

Here, we are building a Decision Tree classifier for predicting male or female. We will take a very small data set having 19 samples. These samples would consist of two features - ‘height’ and ‘length of hair’.

4.5.1 Prerequisite

For building the following classifier, we need to install **pydotplus** and **graphviz**. Basically, graphviz is a tool for drawing graphics using dot files and **pydotplus** is a module to Graphviz’s Dot language. It can be installed with the package manager or pip.

Now, we can build the decision tree classifier with the help of the following Python code:

To begin with, let us import some important libaries as follows:

```
import pydotplus
from sklearn.import tree
from sklearn.datasets import load_iris
from sklearn.metrics import classification_report
from sklearn.import cross_validation
import collections
```

Now, we need to provide the dataset as follows:

```
X = [[165, 19], [175 ,32], [136, 35], [174, 65], [141, 28], [176, 15], [131, 
32],
[166, 6], [128, 32], [179,10], [136, 34], [186, 2], [126, 25], [176, 28], 
[112, 38],
[169, 9], [171, 36], [116, 25], [196, 25]]

Y = ['Man', 'Woman', 'Woman', 'Man', 'Woman', 'Man', 'Woman', 'Man', 'Woman', 
'Man', 'Woman', 'Man', 'Woman', 'Woman', 'Woman', 'Man', 'Woman', 'Woman', 
'Man']
data_feature_names = ['height', 'length of hair']

X_train, X_test, Y_train, Y_test = cross_validation.train_test_split
(X, Y, test_size=0.40, random_state=5)
```

After providing the dataset, we need to fit the model which can be done as follows:

```
clf = tree.DecisionTreeClassifier()
clf = clf.fit(X, Y)
```

Prediction can be made with the help of the following code:

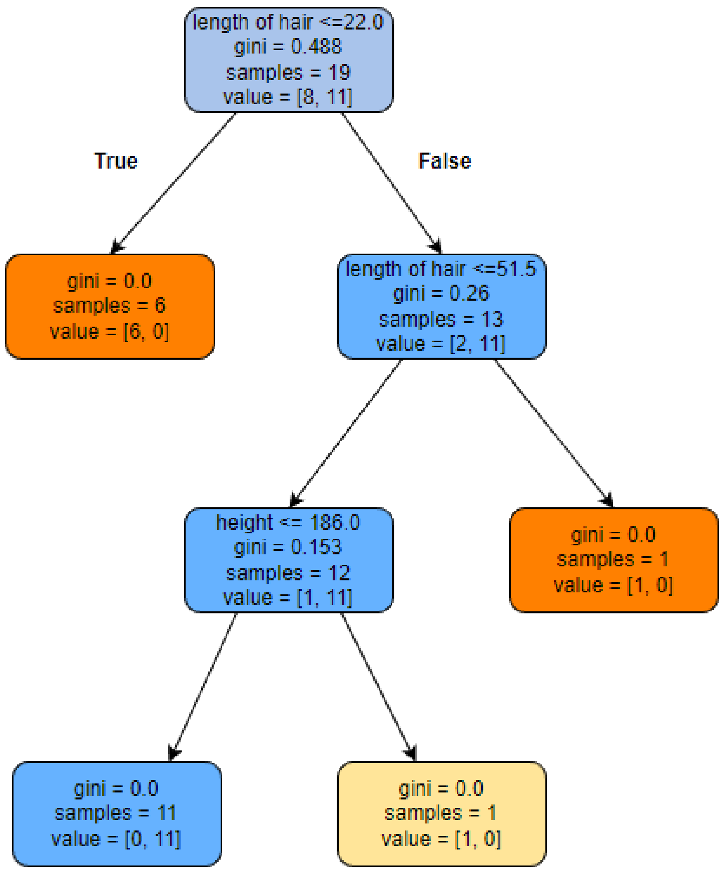
```
dot_data = tree.export_graphviz(clf, feature_names = data_feature_names, 
           out_file = None, filled = True, rounded = True)
graph = pydotplus.graph_from_dot_data(dot_data)
colors = ('orange', 'yellow')
edges = collections.defaultdict(list)

for edge in graph.get_edge_list():
edges [edge.get_source()].append(int(edge.get_destination()))

for edge in edges: edges[edge].sort()

for i in range(2):dest = graph.get_node(str(edges[edge][i]))[0]
dest.set_fillcolor(colors[i])
graph.write_png('Decisiontree16.png')
```

It will give the prediction for the above code as **['Woman']** and create the following decision tree:



Prediction based on model

We can change the values of features in prediction to test it.

4.6 Random Forest Classifier

As we know that ensemble methods are the methods which combine machine learning models into a more powerful machine learning model. Random Forest, a collection of decision trees, is one of them. It is better than single decision tree because while retraining the predictive powers it can reduce over-filtering by averaging the results. Here, we are going to implement the random forest model on scikit learn cancer dataset.

Import the necessary packages:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_breast_cancer
cancer = load_breast.cancer()
import matplotlib.pyplot as plt
import numpy as np
```

Now, we need to provide the dataset which can be done as follows&minus

```
cancer = load_breast_cancer()
X_train, X_test, y_train,
y_test = train_test_split(cancer.data, cancer.target, random_state = 0)
```

After providing the dataset, we need to fit the model which can be done as follows:

```
forest = RandomForestClassifier(n_estimators = 50, random_state = 0)
forest.fit(X_train, y_train)
```

Now, get the accuracy on training as well as testing subset: if we will increase the number of estimators then, the accuracy of testing subset would be increased.

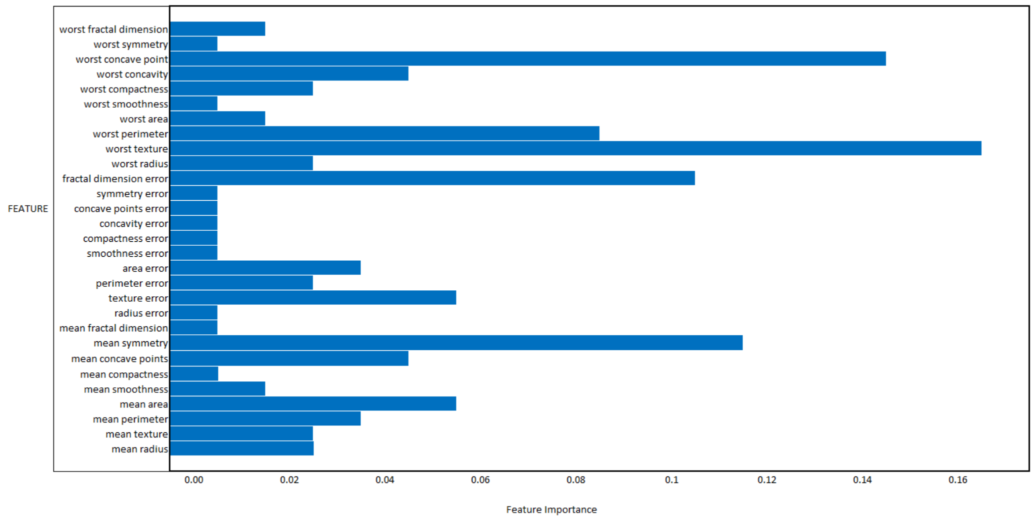
```
print('Accuracy on the training subset:(:.3f)', format(forest.score(X_train,
y_train)))
print('Accuracy on the training subset:(:.3f)', format(forest.score(X_test,
y_test)))
```

4.6.1 Output

```
Accuracy on the training subset:(:.3f) 1.0
Accuracy on the training subset:(:.3f) 0.965034965034965
```

Now, like the decision tree, random forest has the **feature_importance** module which will provide a better view of feature weight than decision tree. It can be plot and visualize as follows:

```
n_features = cancer.data.shape[1]
plt.barh(range(n_features),forest.feature_importances_, align='center')
plt.yticks(np.arrange(n_features), cancer.feature_names)
plt.xlabel('Feature Importance')
plt.ylabel('Feature')
plt.show()
```



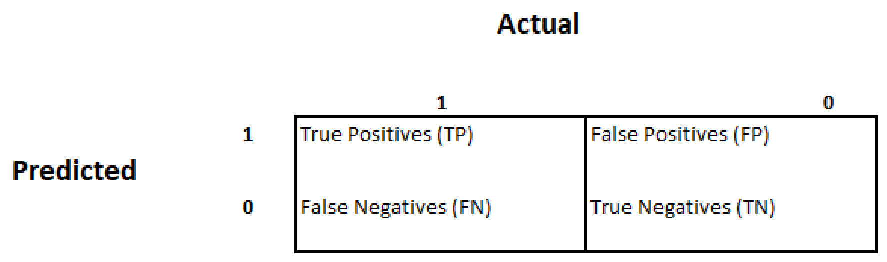
Output

4.7 Performance of a classifier

After implementing a machine learning algorithm, we need to find out how effective the model is. The criteria for measuring the effectivess may be based upon datasets and metric. For evaluating different machine learning algorithms, we can use different performance metrics. For example, suppose if a classifier is used to distinguish between images of different objects, we can use the classification performance metrics such as average accuracy, AUC, etc. In one or other sense, the metric we choose to evaluate our machine learning model is very important because the choice of metrics influences how the performance of a machine learning algorithm is measured and compared. Following are some the metrics:

4.7.1 Confusion Matrix

Basically it is used for classification problem where the output can be of two or more types of classes. It is the easiest way to measure the performance of a classifier. A confusion matrix is basically a table with two dimensions namely "Actual" and "Predicted", Both the dimensions have "True Positives (TP)", "True Negatives (TN)", "False Positives (FP)", "False Negatives (FN)".



		Actual	
		1	0
Predicted	1	True Positives (TP)	False Positives (FP)
	0	False Negatives (FN)	True Negatives (TN)

Confusion Matrix

Confusion Matrix

In the confusion matrix above, 1 is for positive class and 0 is for negative class.

Following are the terms associated with Confusion matrix:

- **True Positives** - TPs are the cases when the actual class of data point was 1 and the predicted is also 1.
- **True Negatives** - TNs are the cases when the actual class of the data point was 0 and the predicted is also 0.
- **False Positives** - FPs are the cases when the actual class of data point was 0 and the predicted is also 1.
- **False Negatives** - FNs are the cases when the actual class of the data point was 1 and the predicted is also 0.

4.7.2 Accuracy

The confusion matrix itself is not a performance measure as such but almost all the performance matrices are based on the confusion matrix. One of them is accuracy. In classification problems, it may be defined as the number of correct predictions made by the model over all kinds of predictions made. The formula for calculating the accuracy is as follows:

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

4.7.3 Precision

It is mostly used in document retrievals. It may be defined as how many of the returned docuemnts are correct. Following is the formula for calculating the precision:

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

4.7.4 Recall or Sensitivity

It may be defined as how many of the positives do the model return. Following is the formula for calculating the recall/sensitivity of the model:

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

4.7.5 Specificity

It may be defined as how many of the negatives do the model return. It is exactly opposite to recall. Following is the formula for calculating the specificity of the model:

```
$$Specificity = \frac{TN}{TN+FP}$$
```

4.8 Class Imbalance Problem

Class imbalance is the scenario where the number of observation belonging to one class is significantly lower than those belonging to the other classes. For example, the problem is prominent in the scenario where we need to identify the rare diseases, fraudulent transaction in bank etc.

4.8.1 Example of imbalanced classes

Let us consider an example of fraud detection data set to undestand the concept of imbalanced class:

```
Total observations = 5000
Fradulent Observations = 50
Non-Fradulent Observations = 4950
Event Rate = 1%
```

4.8.2 Solution

Balancing the classes' acts as a solution to imbalanced classes. The main objective of balancing the classes is to either increase the frequency of the minority class or decrease the frequency of the majority class. Following are the approaches to solve the issue of imbalances classes:

4.8.3 Re-Sampling

Re-sampling is a series of methods used to reconstruct the sample data sets - both training sets and testing sets. Re-sampling is done to improve the accuracy of model. Following are some re-sampling techniques:

- **Random Under-Sampling**: This technique aims to balance class distribution by randomly eliminating majority class examples. This is done until the majority and minority class instances are balanced out.

```
Total observations = 5000
Fradulent Observations = 50
Non-Fradulent Observations = 4950
Event Rate = 1%
```

In this case, we are taking 10% samples without replacement from non-fraud instance and then combine them with the fraud instances:

Non-fraudulent, observations after random under sampling = 10% of 4950 = 495

Total observations after combining them with fradulent observations = 50+495=545

Hence now, the event rate for new dataset after under sampling = 9%

The main advantage of this technique is that it can reduce run time and improve storage. But on the other side, it can discard useful information while reducing the number of training data samples.

- **Random Over-Sampling**: This technique aims to balance class distribution by increasing the number of instances in the minority class by replacing them.

```
Total observations = 5000
Fradulent Observations = 50
Non-Fradulent Observations = 4950
Event Rate = 1%
```

In case we are replacing 50 fraudulent observations 30 times then fraudulent observations after replicating the minority class observations would be 1500. And then total observation in the new data after oversampling would be 4950+1500 = 6400. Hence the event rate for the new data set would be 1500/6450 = 23%

The main advantage of this method is that there would be no loss of useful information. But on the other hand, it has the increased chances of over-filtting because it replicates the minority class events.

4.9 Ensemble Techniques

This methodology basically is used to modify existing classification algorithms to make them appropriate for imbalanced data sets. In this approach we construct several two stage classifier from the original dta and then aggregate their predictions. Random forest classifier is an example of ensemble based classifier.

5 Regression

Regression is one of the most important statistical and machine learning tools. We would not be wrong to say that the journey of machine learning starts from regression. It may be defined as the parametric technique that allows us to make decisions based upon data by learning the relationship between input and output variables. Here, the output variables dependent on the input variables, are continuous-valued real numbers. In regression, the relationship between input and output variables matters and it helps us in undestanding how the value of the output variable changes with the change of input variable. Regression is frequently used for prediction of prices, economics, variations, and so on.

5.1 Building Regressors in Python

In this section, we will learn how to build single as well as multivariable regressor.

5.1.1 Linear Regressor/Single Variable Regressor

Let us important a few required packages:

```
import numpy as np
from sklearn import linear_model
import sklearn.metrics as sm
import matplotlib.pyplot as plt
```

Now, we need to provide the input data and we have saved our data in the file named linear.txt

```
input = 'D:/ProgramData/linear.txt'
```

We need to load this data by using the **np.loadtxt** function.

```
input_data = np.loadtxt(input, delimiter=',')
X, y = input_data[:, :-1], input_data[:, -1]
```

The next step would be to train the model. Let us give training and testing samples.

```
training_samples = int(0.6 * len(X))
testing_samples = len(X) - num_training

X_train, y_train = X[:training_samples], y[:training_samples]

X_test, y_test = X[training_samples:], y[training_samples:]
```

Now, we need to create a linear regressor object.

```
reg_linear = linear_model.linearRegression()
```

Train the object with the training samples.

```
reg_linear.fit(X_train, y_train)
```

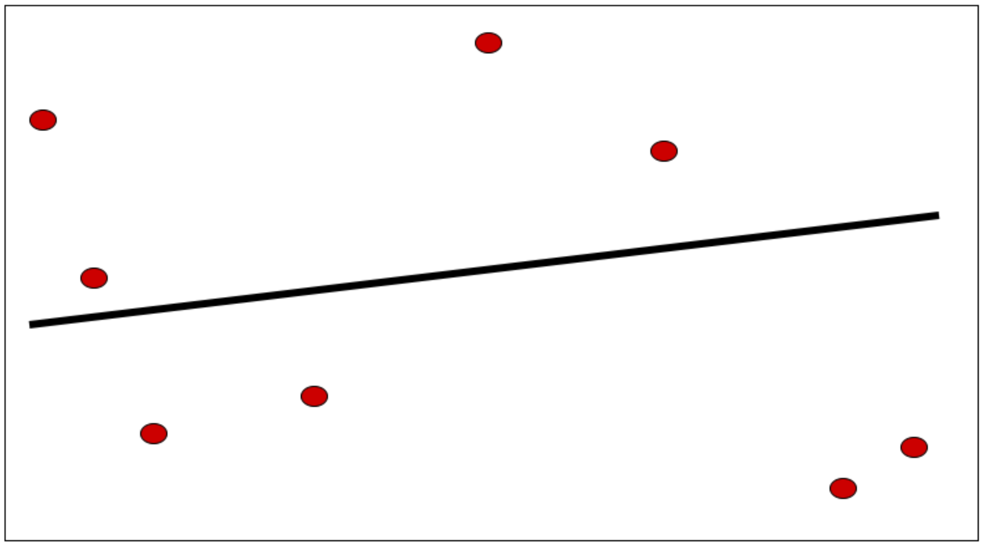
We need to do the prediction with the testing data.

```
y_test_pred = reg_linear.predict(X_test)
```

Now plot and visualize the data.

```
plt.scatter(X_test, y_test, color = 'red')
plt.plot(X_test, y_test_pred, color = 'black', linewidth = 2)
plt.xticks(())
plt.yticks(())
plt.show()
```

5.1.2 Output



Linear Regressor

Now, we can compute the performance of our liner regression as follows:

```
print("Performance of Linear regressor:")
print("Mean absolute error =", round(sm.mean_absolute_error(y_test,
y_test_pred), 2))
print("Mean squred error =", round(sm.mean_sqared_error(y_test, y_test_pred),
2))
print("Median absolute error =", round(sm.median_absolute_error(y_test,
y_test_pred), 2))
print("Explain variance score =", round(sm.explained_variance_score(y_test,
y_test_pred), 2))
print("R2 score =", round(sm.r2_score(y_test, y_test_pred), 2))
```

5.1.3 Output

Performance of Linear Regressor:

```
Mean absolute error = 1.78
Mean squared error = 3.89
Median absolute error = 2.01
Explain variance score = -0.09
R2 score = -0.09
```

In the above code, we have used this small data. If you want some big dataset then you can use sklearn.dataset to import bigger dataset.

```
2,4.82.9,4.72.5,53.2,5.56,57.6,43.2,0.92.9,1.92.4,
3.50.5,3.41,40.9,5.91.2,2.583.2,5.65.1,1.54.5,
1.22.3,6.32.1,2.8
```

5.1.4 Multivariable Regressor0

First, let us import a few required packages:

```
import numpy as np
from sklearn import linear_model
import sklearn.metrics as sm
import matplotlib.pyplot as plt
from sklearn.preprocessing import PolynomialFeatures
```

Now, we need to provide the input data and we have saved our data in the file named linear.txt

```
input = 'D:/ProgramData/Mul_linear.txt'
```

6 Logic Programming

In this chapter, we will focus logic programming and how it helps in Aritificial Intelligence.

We already know that logic is they study of principles of correct reasoning or simple words it is the strudy of what comes after what. For example, if two statements are tru then we can infer any third statement from it.

6.1.1 Concept

Logic Programming is the combination of two words, logic and programming Logic Programming is a programming paradigm in which the problems are expressed as facts and rules by program statements but within a system of formal logic. Just like other programming paradigms like object oriented, functional, declarative, and procedural, etc, it is also a particular way to approach programming.

6.2 How to Solve Problems with Logic Programming

Logic Programming uses facts and rules for solving the problem. That is why they are called the building blocks of Logic Programming. A goal need to bespecified for every program in logic programming. To understand how a problem can be solved in logic programming, we need to know about the building blocks - Facts and Rules.

6.2.1 Facts

Actaully, every logic program needs facts to work with so that it can achieve the given goal. Facts basically are true statements about the program and data. For example, Delhi is the capital of India.

6.2.2 Rules

Actually, rules are the constraints which allow us to make conclusions about the problem domain. Rules basically written as logical clauses to express various facts. For example, if we are building any game then all the rules must be defined.

Rules are very important to solve any problem in Logic Programming. Rules are basically logical conclusion which can express the facts. Following is the synatax of rule:

```
A:-B1,B2,...,B[].
```

Here, A is the head and B1, B2, ... Bn is the body.

For example - ancestor(X,Y):-father(X,Y)

ancestor(X,Z):-father(X,Y), ancestor(Y,Z).

This an be read as, for every X and Y, if X is the father of Y and Y is an ancestor Z, X is the ancestor of Z. For every X and Y, X is the ancestor of Z, if X is the father of Y and Y is an ancestor of Z.

6.3 Installing Useful Packages

For starting logic programming in Python, we need to install the fllowing two packages.

6.3.1 Kanren

It provides us a way to simplify the way we made code for business logic. It lets us express the logic in term of rules and facts. The following command will help you install kanren:

```
pip install kanren
```

6.3.2 SymPy

SymPy is a Python libary for symbolic mathematics. It aims to become a full-featured computer algebra system (CAS) while keeping the code as simple as possible in order to be comprehensible and easily extensible. The following command will help you install SymPy:

```
pip install sympy
```

6.4 Example of Logic Programming

Following are some examples which can be solved by logic programming.

6.4.1 Matching mathematical expressions

Actually we can find the unknown values by using logic programming in a very effective way. The following Python code will help you match a mathematical exprssion.

Consider importing the following packages first:

```
from kanren import run, var, fact
from kanren.assoccomm import eq_assoccomm as eq
from kanren.assoccomm import commutative, associative
```

We need to define the mathematical operations which we are going to use:

```
add = 'add'
mul = 'mul'
```

Both addition and multiplication are communicative processes. Hence, we need to specify it and this can be done as follows:

```
fact(commutative, mul)
fact(commutative, add)
fact(associative, mul)
fact(associative, add)
```

It is compulsory to define variables; this can be done as follows:

```
a, b = var('a'), var ('b')
```

We need to match the expression with the original pattern. We have the following original pattern, which is basically (5+a)*b

```
Original_pattern = (mul, (add, 5, a), b)
```

We have the following two expressions to match with the original pattern:

```
exp1 = (mul, 2, (add, 3, 1))
exp2 = (add, 5, (mul, 8,1))
```

Output can be printed with the following command:

```
print(run(0, (a,b), eq(original_pattern, exp1)))
print(run(0, (a,b), eq(original_pattern, exp2)))
```

After running this code, we will get the following output:

```
((3, 2))
()
```

The first output represents the values for **a** and **b**. The first expression matched the original pattern and returned the values for **a** and **b** but the second expression did not match the original pattern hence nothing has been returned.

6.5 Checking for Prime Numbers

With the help of logic programming, we can find the prime numbers from a list of numbers and can also generate prime numbers. The Python code given below will find the prime number from a list of numbers and will also generate the firs 10 prime numbers.

Let us first consider importing the following packages:

```
from kanren import isvar, run, membero
from kanren.core import success, fail, goaleval, condeseq, eq, var
from sympy.ntheory.generate import prime, isprime
import itertools as it
```

Now, we will define a function called prime_check which will check the prime numbers based on the given numers as data.

```
def prime_check(x):
if isvar(x):
   return condeseq([(eq, x, p)] for p in map(prime, it.count(1)))
else:
   return success if isprime(x) else fail
```

Now, we need to declare a variable which will be used

```
x = var()
print((set(run(0,x,(membero,x,(12,14,15,19,20,21,22,23,29,30,41,44,52,62,65,8
5)),(prime_check,x)))))
print((run(10,x,prime_check(X))))
```

The output of the code will be as follows:

```
{19, 23, 29, 41}
(2, 3, 5, 7, 11, 13, 17, 19, 23, 29)
```

6.6 Solving Puzzles

Logic programming can be used to solve many problems like 8-puzzles, Zebra puzzle, Sudoku, N-queen, etc. Here we are talking an example of a variant of Zebra puzzle which is as follows:

```
There are five houses.
The English man lives in the red house.
The Swede has a dog.
The Dane drinks tea.
The green house is immediately to the left of the white house.
They drink coffee in the green house.
The man who smokes Pall Mall has birds.
In the yellow house they smoke Dunhill.
In the middle house they drink milk.
The Norwegian lives in the first house.
The man who smokes Blend lives in the house next to the house with cats.
In a house next to the house where they have a horse, they smoke Dunhill.
The man who smokes Blue Master drinks beer.
The German smokes Prince.
The Norwegian lives next to the blue house.
They drink water in a house next to the whouse where they smoke Blend.
```

We are solving it for the question **who owns zebra** with help of Python.

Let us import the necessary packages.

```
from kanren import *
from kanren.core import lall
import time
```

Now, we need to define two functions - **left()** and **next()** to check whose house is left or next to who's house:

```
def left(q, p, list):
    return membero((q, p), zip(list, list[1:]))
    def next(q, , p, list):
    return conde([left(q, p, list)], [left(p,q, lists)])
```

Now, we will declare a variable house as follows:

```
houses = var()
```

We need to define the rules with the help of lall package as follows:

There are 5 houses:

```
rules_zebraproblem = lall(
  (eq, (var(), var(), var(), var(), var()), houses),

  (membero, ('Enhlishman', var(), var(), var(), 'red'), houses),
  (membero, ('Swedie', var(), var(), 'dog', var()), houses),
  (membero, ('Dane', var(), 'tea', var(), var()), 'houses),
  (left,(var(), var(), var(), var()), 'green'),
  (var(), var(), var(), var(), 'green'),
  (membero,(var(), var(), 'coffee', var(), 'green'), houses),
  (membero,(var(), 'Pall Mall', var(), 'birds', var()), houses),
  (eq,(var(), var(), (var(), var(), 'milk', var(), var()), var(), var()),
houses),
  (eq,(('Norwegian', var(), var(), var(), var()), var(), var(), var(),
var()), houses),
  (next,(var(), 'Blend', var(), var(), var()),
  (var(), var(), var(), 'cats', var()), houses),
  (next,(var(), 'Dunhill', var(), var(), var()),
  (var(), var(), var(), 'horse', var()), houses),
  (membero, (var(), 'Blue Master', 'beer', var(), var()), houses),
  (membero,('German', 'Prince', var(), var(), var()), houses),
  (next,('Norwegian', var(), var(), var(), var()),
  (var(), var(), var(), var(), 'blue'), houses),
  (next,(var(), 'Blend', var(), var(), var()),
  (var(), var(), 'water', var(), var()), houses),
  (membero,(var(), var(), var(), 'zebra', var()), houses)
)
```

Now, run the solver with the preceding constraints:

```
solutions = run(0, houses, rules_zebraproblem)
```

With the help of the following code, we can extract the output from the solver:

```
output_zebra = [house for house in solutions [0] if 'zebra' in house][0][0]
```

The followng code will help print the solution:

```
print ('\n' + output_zebra + 'owns zebra.')
```

The output of the above code would be as follows:

```
German owns zebra.
```

7 Unsupervised Learning Clustering

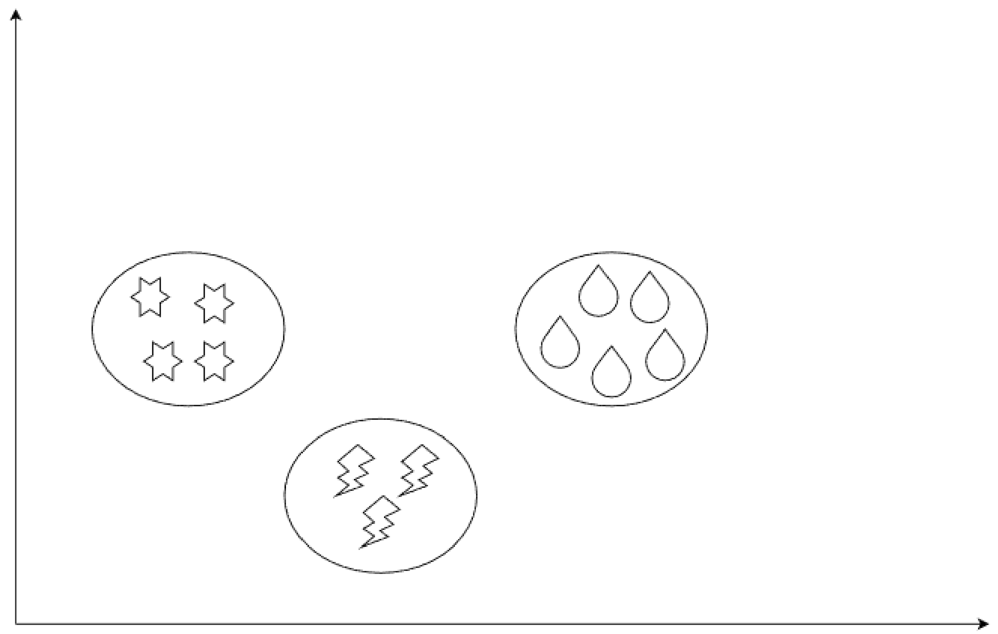
Unsupervised machine learning algorithms do not have any supervisor to provide any sort of guidance. That is why they are closely aligned with what some call true artificial intelligence.

In unsupervised learning, there would be no correct answer and no teacher for the guidance. ALgorithms need to discover the interesting pattern in data for learning.

7.1 What is Clustering?

Basically, it is a type of unsupervised learning method and a common technique for statistical data analysis used in many fields. Clustering mainly is a task of dividing the set of observation into subsets, called clusters, in such a way that observations in the same cluster are similar in one sense and they are dissimilar to the observations in other clusters. In simple owrds, we can say that the main goal of clustering is to group the data on the basis of similarity and dissimilarity.

For example, the following diagram shows similar kind of data in different clusters:



Clustering

7.2 Algorithms for Clustering the Data

Following are a few common algorithms for clustering the data.

7.2.1 K-Means algorithm

K-means clustering algorithm is one of the well-known algorithms for clustering the data. We need to assume that the numbers of clusters are already known. This is also called flat clustering. It is an iterative clustering algorithm. The steps given below need to be followed for this algorithm:

Step 1 - We need to specify the desired number of K subgroups.

Step 2 - Fix the number of clusters and randomly assign each data point to a cluster. Or in other words we need to classify our data based on the number of clusters.

In this step, cluster centroids should be computed.

As this is an iterative algorithm, we need to update the locations of K centroids with every iteration until we find the global optima or in other words the centroids reach at their optimal locations.

The following code will help in implementing K-means clustering algorithm in Python. We are going to use the Scikit-learn module.

Let us import the necessary packages:

```
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()
import numpy as np
from sklearn.cluster import KMeans
```

The following line of code will help in generating the two-dimensional dataset, containing four blobs, by using **make_blob** from the **sklearn.dataset** package.

```
from sklearn.datasets.samples_generator import make_blobs

X, y_true = make_blobs(n_samples = 500, centers = 4,
            cluster_std = 0.40, random_state = 0 )
```

We can visualize the dataset by using the following code:

```
plt.scatter(X[:, 0], X[:,1], s = 50);
plt.show()
```

We will load this data by using the **np.loadtxt** function.

```
input_data = np.loadtxt(input, delimiter=',')
X, y = input_data[:, :-1], input_data[:, -1]
```

Th next step would be to train the model, we will give training and testing samples.

```
training_samples = int(0.6 * len(X))
testing_samples = len(X) - num_training

X_train, y_train = X[:training_samples], y[:training_samples]

X_test, y_test = X[training_samples:], y[training_samples:]
```

Now, we need to create a linear regressor object.

```
reg_linear_mul = linear_model.linearRegression()
```

Train the object with the training samples.

```
reg_linear_mul.fit(X_train, y_train)
```

Now, at last we need to do the prediction with the testing data.

```
print("Performance of Linear regressor:")
print("Mean absolute error =", round(sm.mean_absolute_error(y_test,
y_test_pred), 2))
print("Mean squred error =", round(sm.mean_sqared_error(y_test, y_test_pred),
2))
print("Median absolute error =", round(sm.median_absolute_error(y_test,
y_test_pred), 2))
print("Explain variance score =", round(sm.explained_variance_score(y_test,
y_test_pred), 2))
print("R2 score =", round(sm.r2_score(y_test, y_test_pred), 2))
```

7.2.2 Output

Performance of Linear Regressor:

```
Mean absolute error = 0.6
Mean squared error = 0.65
Median absolute error = 0.41
Explain variance score = 0.34
R2 score = 0.33
```

Now, we will create a polynomial of defree 10 and train the regressor. We will provide the sample data point.

```
polynomial = PolynomialFeatures(defree = 10)
X_train_transformed = polynomial.fit_transform(X_train)
datapoint = [[2.23, 1.35, 1.12]]
poly_datapoint = polynomial.fit_transform(datapoint)

poly_linear_model = linear_model.linearRegression()
poly_linear_model.fit(X_train_transformed, y_train)
print("\nLinear regression :\n", reg_linear_mul.predict(datapoint))
print("\nPolynomial regression:\n",
poly_linear_model.predict(poly_datapoint))
```

7.2.3 Output

Linear regression:

```
[2.40170462]
```

Polynomial regression:

```
[1.8697225]
```

In the above code, we have used this small data. If you want a big dataset then, you can use sklearn dataset to import a bigger dataset.

```
2,4.8,1.2,3.22.9,4.7,1.5,3.62.5,5,2.8,23.2,5.5,3.5,2.16,5,
2,3.27.6,4,1.2,3.23.2,0.9,2.3,1.42.9,5.9,5.6,0.81.2,2.58,
3.45,1.233.2,5.6,2,3.25.1,1.5,1.2,1.34.5,1.2.4.1,2.32.3,
6.3,2.5,3.22.1,2.8,1.2,3.6
```

8 Logic Programming

In this chapter, we will focus logic programming and how it helps in Aritificial Intelligence.

We already know that logic is they study of principles of correct reasoning or simple words it is the strudy of what comes after what. For example, if two statements are tru then we can infer any third statement from it.

8.1.1 Concept

Logic Programming is the combination of two words, logic and programming Logic Programming is a programming paradigm in which the problems are expressed as facts and rules by program statements but within a system of formal logic. Just like other programming paradigms like object oriented, functional, declarative, and procedural, etc, it is also a particular way to approach programming.

8.2 How to Solve Problems with Logic Programming

Logic Programming uses facts and rules for solving the problem. That is why they are called the building blocks of Logic Programming. A goal need to bespecified for every program in logic programming. To understand how a problem can be solved in logic programming, we need to know about the building blocks - Facts and Rules.

8.2.1 Facts

Actaully, every logic program needs facts to work with so that it can achieve the given goal. Facts basically are true statements about the program and data. For example, Delhi is the capital of India.

8.2.2 Rules

Actually, rules are the constraints which allow us to make conclusions about the problem domain. Rules basically written as logical clauses to express various facts. For example, if we are building any game then all the rules must be defined.

Rules are very important to solve any problem in Logic Programming. Rules are basically logical conclusion which can express the facts. Following is the synatax of rule:

```
A:-B1,B2,...,B[].
```

Here, A is the head and B1, B2, ... Bn is the body.

For example - ancestor(X,Y):-father(X,Y)

ancestor(X,Z):-father(X,Y), ancestor(Y,Z).

This an be read as, for every X and Y, if X is the father of Y and Y is an ancestor Z, X is the ancestor of Z. For every X and Y, X is the ancestor of Z, if X is the father of Y and Y is an ancestor of Z.

8.3 Installing Useful Packages

For starting logic programming in Python, we need to install the fllowing two packages.

8.3.1 Kanren

It provides us a way to simplify the way we made code for business logic. It lets us express the logic in term of rules and facts. The following command will help you install kanren:

```
pip install kanren
```

8.3.2 SymPy

SymPy is a Python libary for symbolic mathematics. It aims to become a full-featured computer algebra system (CAS) while keeping the code as simple as possible in order to be comprehensible and easily extensible. The following command will help you install SymPy:

```
pip install sympy
```

8.4 Example of Logic Programming

Following are some examples which can be solved by logic programming.

8.4.1 Matching mathematical expressions

Actually we can find the unknown values by using logic programming in a very effective way. The following Python code will help you match a mathematical exprssion.

Consider importing the following packages first:

```
from kanren import run, var, fact
from kanren.assoccomm import eq_assoccomm as eq
from kanren.assoccomm import commutative, associative
```

We need to define the mathematical operations which we are going to use:

```
add = 'add'
mul = 'mul'
```

Both addition and multiplication are communicative processes. Hence, we need to specify it and this can be done as follows:

```
fact(commutative, mul)
fact(commutative, add)
fact(associative, mul)
fact(associative, add)
```

It is compulsory to define variables; this can be done as follows:

```
a, b = var('a'), var ('b')
```

We need to match the expression with the original pattern. We have the following original pattern, which is basically (5+a)*b

```
Original_pattern = (mul, (add, 5, a), b)
```

We have the following two expressions to match with the original pattern:

```
exp1 = (mul, 2, (add, 3, 1))
exp2 = (add, 5, (mul, 8,1))
```

Output can be printed with the following command:

```
print(run(0, (a,b), eq(original_pattern, exp1)))
print(run(0, (a,b), eq(original_pattern, exp2)))
```

After running this code, we will get the following output:

```
((3, 2))
()
```

The first output represents the values for **a** and **b**. The first expression matched the original pattern and returned the values for **a** and **b** but the second expression did not match the original pattern hence nothing has been returned.

8.5 Checking for Prime Numbers

With the help of logic programming, we can find the prime numbers from a list of numbers and can also generate prime numbers. The Python code given below will find the prime number from a list of numbers and will also generate the firs 10 prime numbers.

Let us first consider importing the following packages:

```
from kanren import isvar, run, membero
from kanren.core import success, fail, goaleval, condeseq, eq, var
from sympy.ntheory.generate import prime, isprime
import itertools as it
```

Now, we will define a function called prime_check which will check the prime numbers based on the given numers as data.

```
def prime_check(x):
if isvar(x):
   return condeseq([(eq, x, p)] for p in map(prime, it.count(1)))
else:
   return success if isprime(x) else fail
```

Now, we need to declare a variable which will be used

```
x = var()
print((set(run(0,x,(membero,x,(12,14,15,19,20,21,22,23,29,30,41,44,52,62,65,8
5)),(prime_check,x)))))
print((run(10,x,prime_check(X))))
```

The output of the code will be as follows:

```
{19, 23, 29, 41}
(2, 3, 5, 7, 11, 13, 17, 19, 23, 29)
```

8.6 Solving Puzzles

Logic programming can be used to solve many problems like 8-puzzles, Zebra puzzle, Sudoku, N-queen, etc. Here we are talking an example of a variant of Zebra puzzle which is as follows:

```
There are five houses.
The English man lives in the red house.
The Swede has a dog.
The Dane drinks tea.
The green house is immediately to the left of the white house.
They drink coffee in the green house.
The man who smokes Pall Mall has birds.
In the yellow house they smoke Dunhill.
In the middle house they drink milk.
The Norwegian lives in the first house.
The man who smokes Blend lives in the house next to the house with cats.
In a house next to the house where they have a horse, they smoke Dunhill.
The man who smokes Blue Master drinks beer.
The German smokes Prince.
The Norwegian lives next to the blue house.
They drink water in a house next to the whouse where they smoke Blend.
```

We are solving it for the question **who owns zebra** with help of Python.

Let us import the necessary packages.

```
from kanren import *
from kanren.core import lall
import time
```

Now, we need to define two functions - **left()** and **next()** to check whose house is left or next to who's house:

```
def left(q, p, list):
    return membero((q, p), zip(list, list[1:]))
    def next(q, , p, list):
    return conde([left(q, p, list)], [left(p,q, lists)])
```

Now, we will declare a variable house as follows:

```
houses = var()
```

We need to define the rules with the help of lall package as follows:

There are 5 houses:

```
rules_zebraproblem = lall(
  (eq, (var(), var(), var(), var(), var()), houses),

  (membero, ('Enhlishman', var(), var(), var(), 'red'), houses),
  (membero, ('Swedie', var(), var(), 'dog', var()), houses),
  (membero, ('Dane', var(), 'tea', var(), var()), 'houses),
  (left,(var(), var(), var(), var()), 'green'),
  (var(), var(), var(), var(), 'green'),
  (membero,(var(), var(), 'coffee', var(), 'green'), houses),
  (membero,(var(), 'Pall Mall', var(), 'birds', var()), houses),
  (eq,(var(), var(), (var(), var(), 'milk', var(), var()), var(), var()),
houses),
  (eq,(('Norwegian', var(), var(), var(), var()), var(), var(), var(),
var()), houses),
  (next,(var(), 'Blend', var(), var(), var()),
  (var(), var(), var(), 'cats', var()), houses),
  (next,(var(), 'Dunhill', var(), var(), var()),
  (var(), var(), var(), 'horse', var()), houses),
  (membero, (var(), 'Blue Master', 'beer', var(), var()), houses),
  (membero,('German', 'Prince', var(), var(), var()), houses),
  (next,('Norwegian', var(), var(), var(), var()),
  (var(), var(), var(), var(), 'blue'), houses),
  (next,(var(), 'Blend', var(), var(), var()),
  (var(), var(), 'water', var(), var()), houses),
  (membero,(var(), var(), var(), 'zebra', var()), houses)
)
```

Now, run the solver with the preceding constraints:

```
solutions = run(0, houses, rules_zebraproblem)
```

With the help of the following code, we can extract the output from the solver:

```
output_zebra = [house for house in solutions [0] if 'zebra' in house][0][0]
```

The followng code will help print the solution:

```
print ('\n' + output_zebra + 'owns zebra.')
```

The output of the above code would be as follows:

```
German owns zebra.
```

9 Unsupervised Learning Clustering

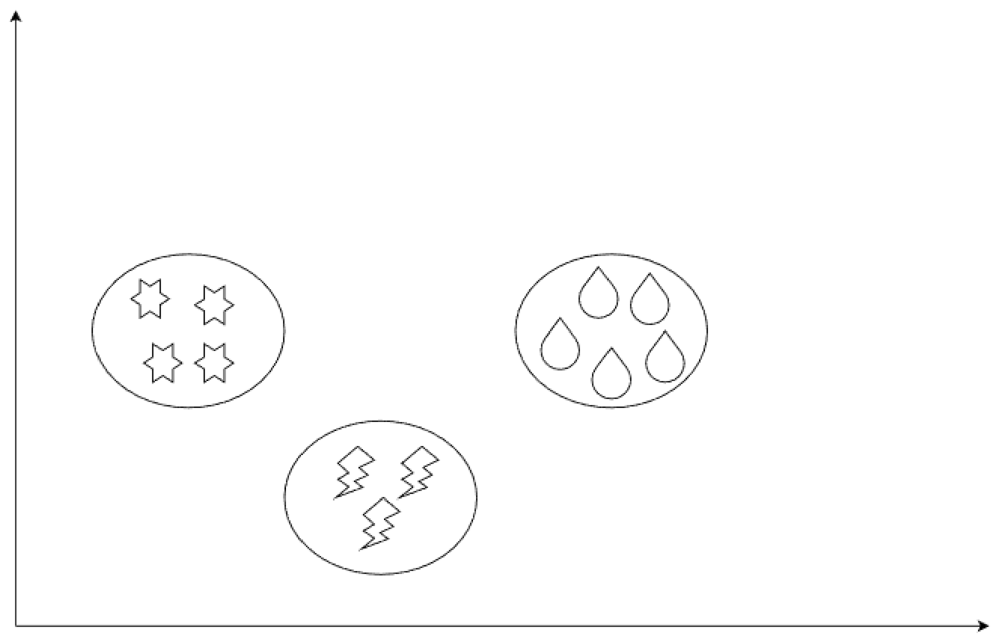
Unsupervised machine learning algorithms do not have any supervisor to provide any sort of guidance. That is why they are closely aligned with what some call true artificial intelligence.

In unsupervised learning, there would be no correct answer and no teacher for the guidance. ALgorithms need to discover the interesting pattern in data for learning.

9.1 What is Clustering?

Basically, it is a type of unsupervised learning method and a common technique for statistical data analysis used in many fields. Clustering mainly is a task of dividing the set of observation into subsets, called clusters, in such a way that observations in the same cluster are similar in one sense and they are dissimilar to the observations in other clusters. In simple owrds, we can say that the main goal of clustering is to group the data on the basis of similarity and dissimilarity.

For example, the following diagram shows similar kind of data in different clusters:



Clustering

9.2 Algorithms for Clustering the Data

Following are a few common algorithms for clustering the data.

9.2.1 K-Means algorithm

K-means clustering algorithm is one of the well-known algorithms for clustering the data. We need to assume that the numbers of clusters are already known. This is also called flat clustering. It is an iterative clustering algorithm. The steps given below need to be followed for this algorithm:

Step 1 - We need to specify the desired number of K subgroups.

Step 2 - Fix the number of clusters and randomly assign each data point to a cluster. Or in other words we need to classify our data based on the number of clusters.

In this step, cluster centroids should be computed.

As this is an iterative algorithm, we need to update the locations of K centroids with every iteration until we find the global optima or in other words the centroids reach at their optimal locations.

The following code will help in implementing K-means clustering algorithm in Python. We are going to use the Scikit-learn module.

Let us import the necessary packages:

```
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()
import numpy as np
from sklearn.cluster import KMeans
```

The following line of code will help in generating the two-dimensional dataset, containing four blobs, by using **make_blob** from the **sklearn.dataset** package.

```
from sklearn.datasets.samples_generator import make_blobs

X, y_true = make_blobs(n_samples = 500, centers = 4,
            cluster_std = 0.40, random_state = 0 )
```

We can visualize the dataset by using the following code:

```
plt.scatter(X[:, 0], X[:,1], s = 50);
plt.show()
```