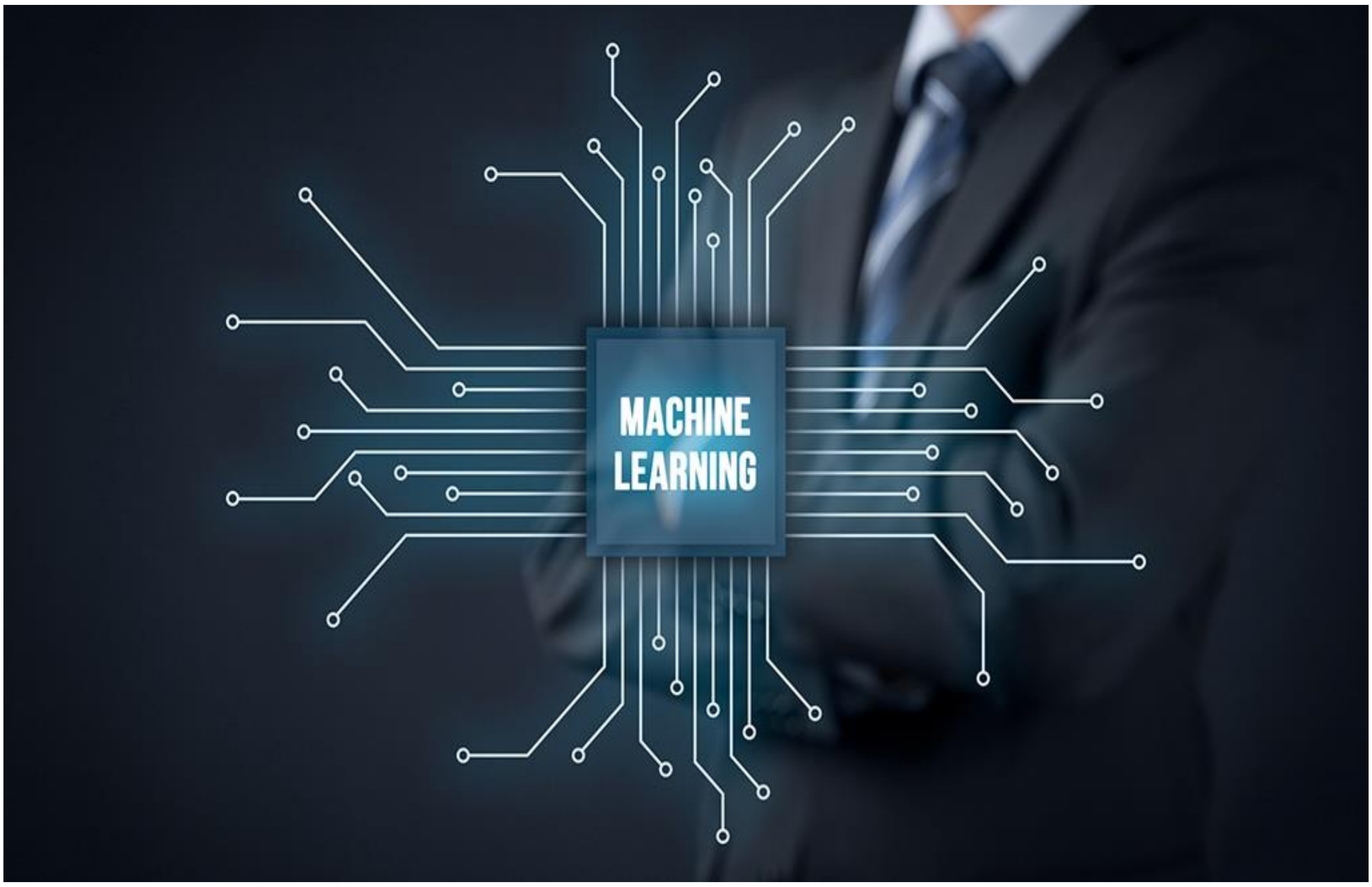


Machine Learning – Hand Book

Part: 3

Edited by: Susmoy Barman

Source: GeeksForGeeks

Multilayer perceptron

Introduction to Artificial Neutral Networks | Set 1

ANN learning is robust to errors in the training data and has been successfully applied for learning real-valued, discrete-valued, and vector-valued functions containing problems such as interpreting visual scenes, speech recognition, and learning robot control strategies. The study of artificial neural networks (ANNs) has been inspired in part by the observation that biological learning systems are built of very complex webs of interconnected neurons in brains. The human brain contain a densely inter- connected network of approximately 10^11-10^12 neurons, each connected neuron, on average connected , to l0^4-10^5 others neurons. So on an average human brain take approximate 10^-1 to make surprisingly complex decisions . ANN systems is movitated to capture this kind of highly parallel computation based on distributed representations. Generally ANNs are built out of a densely interconnected set of simple units, where each unit takes a number of real-valued inputs and produces a single real-valued output .
But ANNs are less motivated by biological neural systems, there are many complexities to biological neural systems that are not modeled by ANNs. Some of them are shown in figures.

Difference between Biological Neurons and Artificial Neurons

Biological Neurons	Artificial Neurons
Major components: Axions, Dendrites, Synapse	Major Components: Nodes, Inputs, Outputs, Weights, Bias
Information from other neurons, in the form of electrical impulses, enters the dendrites at connection points called synapses. The information flows from the dendrites to the cell where it is processed. The output signal, a train of impulses, is then send down the axon to the synapse of other neurons.	The arrangements and connections of the neurons made up the network and have three layers. The first layer is called the input layer and is the only layer exposed to external signals. The input layer transmits signals to the neurons in the next layer, which is called a hidden layer. The hidden layer extracts relavent features or patterns from the received signals. Those features or patterns that are considered important are then directed to the output layer, which is the final layer of the network.
A synapse is able to increase or decrease the strength of the connection. This is where information is stored.	The artificial signals can be changed by weights in a manner similar to the physical changes that occur in the synapses.
Approx 10^{11} neurons.	10^2– 10^4 neurons with current technology

Difference between the human brain and computers in terms of how information is processed.

Human Brain(Biological Neuron Network)	Computers(Artificial Neuron Network)
The human brain works asynchronously	Computers(ANN) work synchronously.

Biological Neurons compute slowly (several ms per computation)	Artificial Neurons compute fast (<1 nanosecond per computation)
The brain represents information in a distributed way because neurons are unreliable and could die any time.	In computer programs every bit has to function as intended otherwise these programs would crash.
Our brain changes their connectivity over time to represents new information and requirements imposed on us.	The connectivity between the electronic components in a computer never change unless we replace its components.
Biological neural networks have complicated topologies.	ANNs are often in a tree structure.
Researchers are still to find out how the brain actually learns.	ANNs use Gradient Descent for learning.

Advantage of Using Artificial Neural Networks:

- Problem in ANNs can have instances that are represented by many attribute-value pairs.
- ANNs used for problems having the target function output may be discrete-valued, real-valued, or a vector of several real- or discrete-valued attributes.
- ANN learning methods are quite robust to noise in the training data.The training examples may contain errors,which do not affect the final output.
- It is used generally used where fast evaluation of the learned target function may be required.
- ANNs can bear long training times depending on factors such as the number of weights in the network, the number of training examples considered, and the settings of various learning algorithm parameters.

The McCulloch-Pitts Model of Neuron:
The early model of an artificial neuron is introduced by **Warren McCulloch** and Walter Pitts in 1943. The McCulloch-Pitts neural model is also known as linear threshold gate. It is a neuron of a set of inputs I1, I2,…,Im and one output y. The linear threshold gate simply classifies the set of inputs into two different classes. Thus the output y is binary. Such a function can be described mathematically using these equations:

$$Sum = \sum_{i=1}^{N} I_i W_i,$$
$$y = f(Sum)$$

W1,W2,W3….Wn are weight values normalized in the range of either (0,1)or (-1,1) and associated with each input line, Sum is the weighted sum, and is a threshold constant. The function f is a linear step function at threshold

Single-layer Neural Networks (Perceptrons)
Input is multi-dimensional (i.e. input can be a vector):
input x = (I1, I2, .., In)
Input nodes (or units) are connected (typically fully) to a node (or multiple nodes) in the next layer. A node in the next layer takes a weighted sum of all its inputs:

$$SummedInput = \sum_i w_i I_i$$

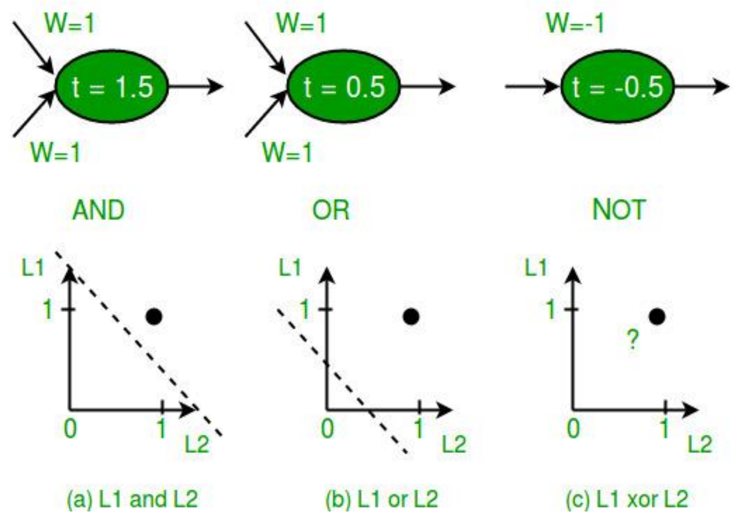
The rule:

The output node has a “threshold” t.

Rule: If summed input ? t, then it “fires” (output y = 1). Else (summed input < t) it doesn't fire (output y = 0).

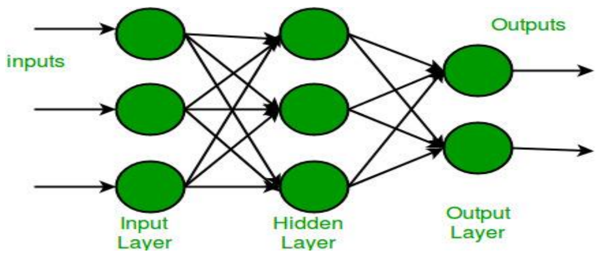
$$\begin{array}{l} if \sum_i w_i I_i \geqslant t \\ \quad \text{then y=1} \\ \text{else (if } \sum_i w_i I_i < t) \\ \quad \text{then y=0} \end{array}$$

Boolean Functions and Perceptrons



Limitations of Perceptrons:
(i) The output values of a perceptron can take on only one of two values (0 or 1) due to the hard-limit transfer function.
(ii) Perceptrons can only classify linearly separable sets of vectors. If a straight line or a plane can be drawn to separate the input vectors into their correct categories, the input vectors are linearly separable. If the vectors are not linearly separable, learning will never reach a point where all vectors are classified properly The Boolean function XOR is not linearly separable (Its positive and negative instances cannot be separated by a line or hyperplane). Hence a single layer perceptron can never compute the XOR function. This is a big drawback which once resulted in the stagnation of the field of neural networks. But this has been solved by multi-layer.

Multi-layer Neural Networks
A Multi Layer Perceptron (MLP) or Multi-Layer Neural Network contains one or more hidden layers (apart from one input and one output layer). While a single layer perceptron can only learn linear functions, a multi layer perceptron can also learn non – linear functions.



This neuron takes as input x1,x2,….,x3 (and a +1 bias term), and outputs f(summed inputs+bias), where f(.) called the activation function. The main function of Bias is to provide every node with a trainable constant value (in addition to the normal inputs that the node receives). Every activation function (or non-linearity) takes a single number and performs a certain fixed mathematical operation on it. There are several activation functions you may encounter in practice:

Sigmoid:takes real-valued input and squashes it to range between 0 and 1.

$$\sigma(x) = \frac{1}{(1+exp(-x))}$$

tanh:takes real-valued input and squashes it to the range [-1, 1].

$$tanh(x) = 2\sigma(2x) - 1$$

ReLu: ReLu stands for Rectified Linear Units. It takes real-valued input and thresholds it to 0 (replaces negative values to 0).

$$f(x) = max(0, x)$$

Introduction to Artificial Neural Network | Set 2

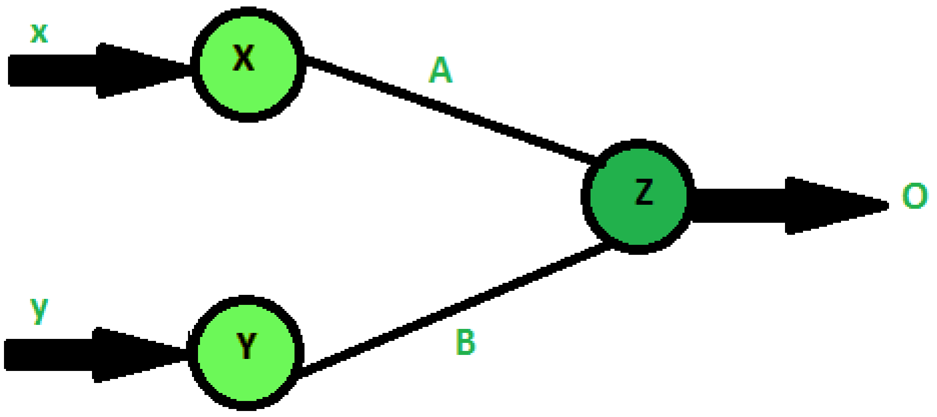
Prerequisite : Introduction to Artificial Neural Network
This article provides the outline for understanding the Artificial Neural Network.
Characteristics of Artificial Neural Network

- It is neurally implemented mathematical model
- It contains huge number of interconnected processing elements called neurons to do all operations
- Information stored in the neurons are basically the weighted linkage of neurons
- The input signals arrive at the processing elements through connections and connecting weights.

- It has the ability to learn , recall and generalize from the given data by suitable assignment and adjustment of weights.
- The collective behavior of the neurons describes its computational power, and no single neuron carries specific information .

How simple neuron works ?

Let there are two neurons **X** and **Y** which is transmitting signal to another neuron **Z** . Then , **X** and **Y** are input neurons for transmitting signals and **Z** is output neuron for receiving signal . The input neurons are connected to the output neuron , over a interconnection links (**A** and **B**) as shown in figure .



Architecture of a Simple Artificial Neuron Net

For above neuron architecture , the net input has to be calculated in the way .
I = xA + yB
where x and y are the activations of the input neurons X and Y . The output z of the output neuron Z can be obtained by applying activations over the net input .
O = f(I)
Output = Function (net input calculated)
The function to be applied over the net input is called *activation function* . There are various activation function possible for this.

Application of Neural Network

1. Every new technology need assistance from previous one i.e. data from previous ones and these data are analyzed so that every pros and cons should be studied correctly . All of these things are possible only through the help of neural network.

2. Neural network is suitable for the research on *Animal behavior, predator/prey relationships and population cycles* .

3. It would be easier to do *proper valuation* of property, buildings, automobiles, machinery etc. with the help of neural network.

4. Neural Network can be used in betting on horse races, sporting events and most importantly in stock market .

5. It can be used to predict the correct judgement for any crime by using a large data of crime details as input and the resulting sentences as output.

6. By analyzing data and determining which of the data has any fault (files diverging from peers) called as *Data mining, cleaning and validation* can be achieved through neural network.

7. Neural Network can be used to predict targets with the help of echo patterns we get from sonar, radar, seismic and magnetic instruments .

8. It can be used efficiently in *Employee hiring* so that any company can hire right employee depending upon the skills the employee has and what should be it's productivity in future .

9. It has a large application in *Medical Research* .

10. It can be used to for *Fraud Detection* regarding credit cards , insurance or taxes by analyzing the past records .

Introduction to ANN (Artificial Neural Networks) | Set 3 (Hybrid Systems)

Prerequisites: Genetic algorithms, Artificial Neural Networks, Fuzzy Logic

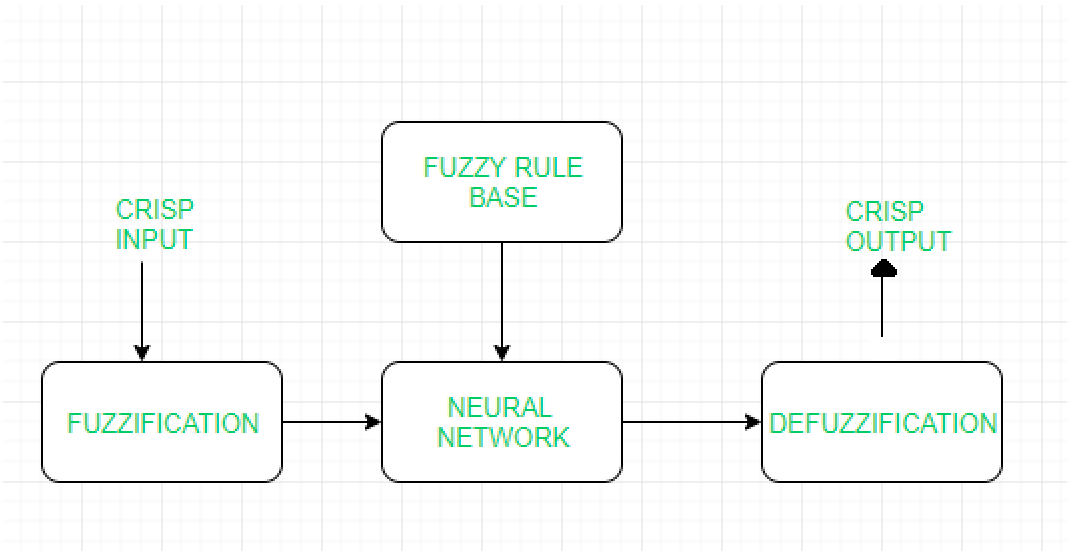
Hybrid systems: A Hybrid system is an intelligent system which is framed by combining atleast two intelligent technologies like Fuzzy Logic, Neural networks, Genetic algorithm, reinforcement Learning, etc. The combination of different techniques in one computational model make these systems possess an extended range of capabilities. These systems are capable of reasoning and learning in an uncertain and imprecise environment. These systems can provide human-like expertise like domain knowledge, adaptation in noisy environment etc.

Types of Hybrid Systems:

- Neuro Fuzzy Hybrid systems
- Neuro Genetic Hybrid systems
- Fuzzy Genetic Hybrid systems

(A) Neuro Fuzzy Hybrid systems:

Neuro fuzzy system is based on fuzzy system which is trained on the basis of working of neural network theory. The learning process operates only on the local information and causes only local changes in the underlying fuzzy system. A neuro-fuzzy system can be seen as a 3-layer feedforward neural network. The first layer represents input variables, the middle (hidden) layer represents fuzzy rules and the third layer represents output variables. Fuzzy sets are encoded as connection weights within the layers of the network, which provides functionality in processing and training the model.



Working flow:

- In input layer, each neuron transmits external crisp signals directly to the next layer.
- Each fuzzification neuron receives a crisp input and determines the degree to which the input belongs to input fuzzy set.
- Fuzzy rule layer receives neurons that represent fuzzy sets.
- An output neuron, combines all inputs using fuzzy operation UNION.
- Each defuzzification neuron represents single output of neuro-fuzzy system.

Advantages:

- It can handle numeric, linguistic, logic, etc kind of information.
- It can manage imprecise, partial, vague or imperfect information.
- It can resolve conflicts by collaboration and aggregation.
- It has self-learning, self-organizing and self-tuning capabilities.
- It can mimic human decision-making process.

Disadvantages:

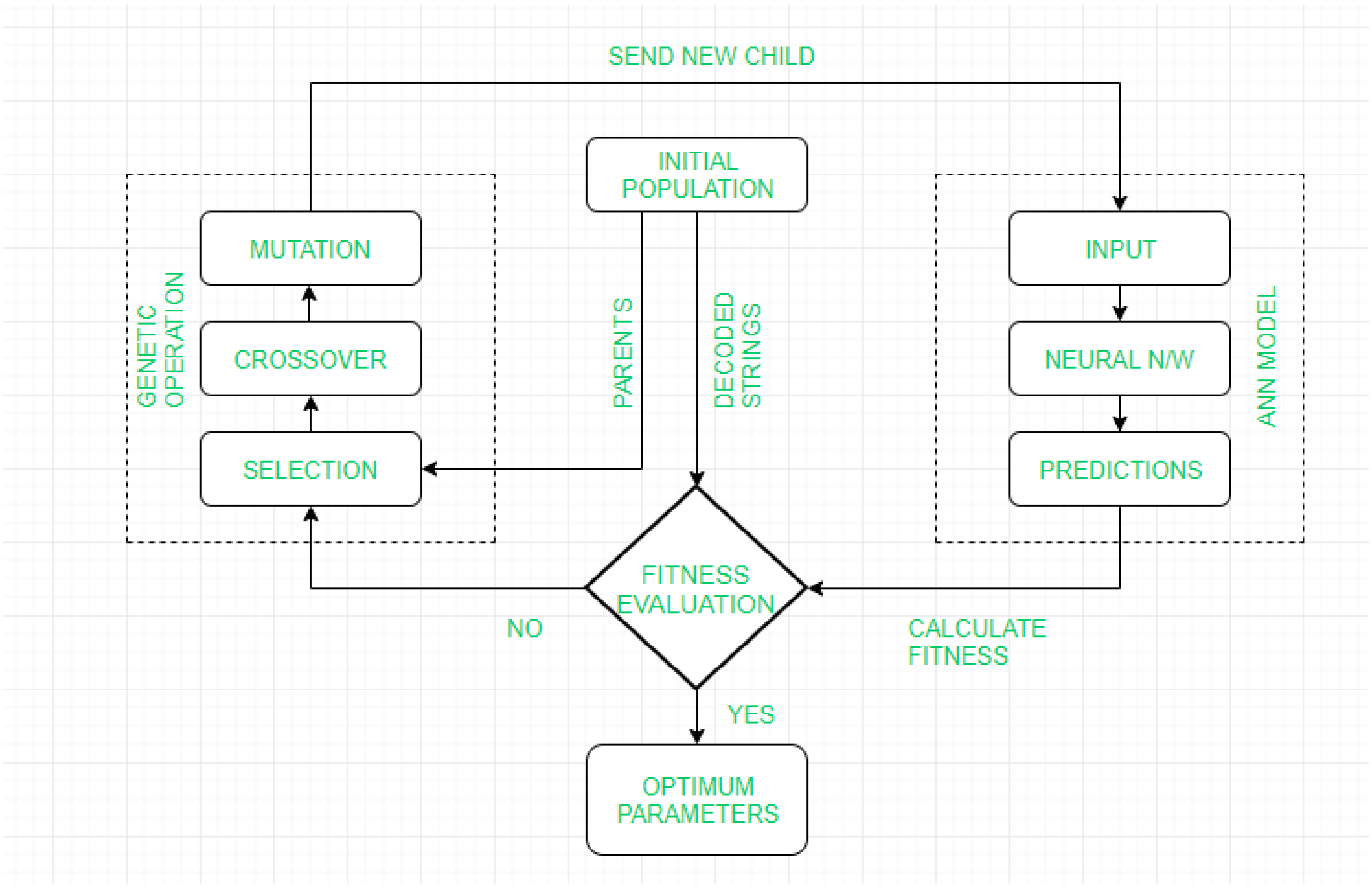
- Hard to develop a model from a fuzzy system
- Problems of finding suitable membership values for fuzzy systems
- Neural networks cannot be used if training data is not available.

Applications:

- Student Modelling
- Medical systems
- Traffic control systems
- Forecasting and predictions

(B) Neuro Genetic Hybrid systems:

A Neuro Genetic hybrid system is a system that combines **Neural networks**: which are capable to learn various tasks from examples, classify objects and establish relation between them and **Genetic algorithm**: which serves important search and optimization techniques. Genetic algorithms can be used to improve the performance of Neural Networks and they can be used to decide the connection weights of the inputs. These algorithms can also be used for topology selection and training network.



Working Flow:

- GA repeatedly modifies a population of individual solutions. GA uses three main types of rules at each step to create the next generation from the current population:
    1. **Selection** to select the individuals, called parents, that contribute to the population at the next generation
    2. **Crossover** to combine two parents to form children for the next generation
    3. **Mutation** to apply random changes to individual parents in order to form children
- GA then sends the new child generation to ANN model as new input parameter.
- Finally, calculating of the fitness by developed ANN model is performed.

Advantages:

- GA is used for topology optimization i.e to select number of hidden layers, number of hidden nodes and interconnection pattern for ANN.
- In GAs, the learning of ANN is formulated as a weight optimization problem, usually using the inverse mean squared error as a fitness measure.

- Control parameters such as learning rate, momentum rate, tolerance level, etc are also optimized using GA.
- It can mimic human decision-making process.

Disadvantages:

- Highly complex system.
- Accuracy of the system is dependent on the initial population.
- Maintaintainance costs are very high.

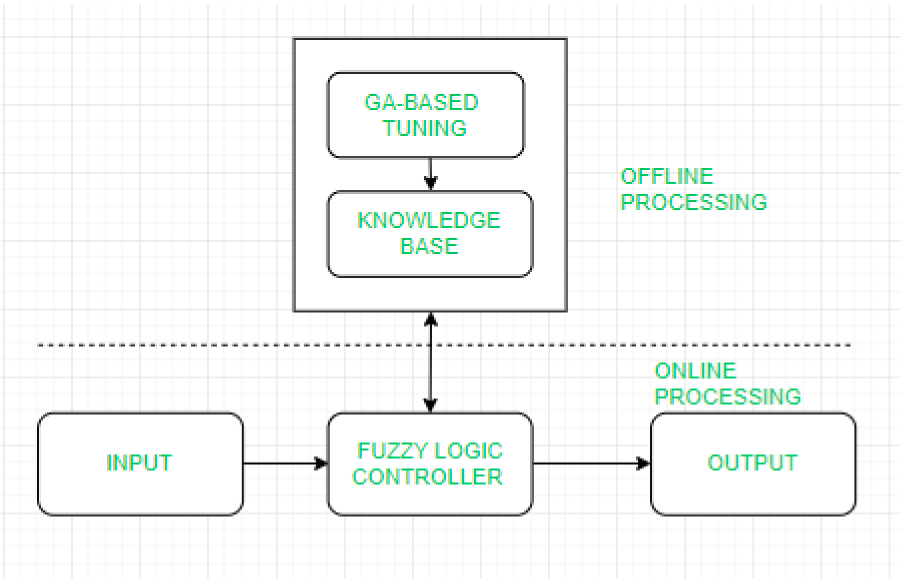
Applications:

- Face recognition
- DNA matching
- Animal and human research
- Behavioral system

(C) Fuzzy Genetic Hybrid systems:

A Fuzzy Genetic Hybrid System is developed to use fuzzy logic based techniques for improving and modelling Genetic algorithms and vice-versa. Genetic algorithm has proved to be a robust and efficient tool to perform tasks like generation of fuzzy rule base, generation of membership function etc.
Three approaches that can be used to develop such system are:

- Michigan Approach
- Pittsburgh Approach
- IRL Approach



Working Flow:

- Start with an initial population of solutions that represent first generation.
- Feed each chromosome from the population into the Fuzzy logic controller and compute performance index.
- Create new generation using evolution operators till some condition is met.

Advantages:

- GAs are used to develop the best set of rules to be used by a fuzzy inference engine
- GAs are used to optimize the choice of membership functions.
- A Fuzzy GA is a directed random search over all discrete fuzzy subsets.
- It can mimic human decision-making process.

Disadvantages:

- Interpretation of results is difficult.
- Difficult to build membership values and rules.
- Takes lots of time to converge.

Applications:

- Mechanical Engineering
- Electrical Engine
- Artificial Intelligence
- Economics

Image Classifier using CNN

The article is about creating an Image classifier for identifying cat-vs-dogs using TFLearn in Python. The problem is here hosted on kaggle.

Machine Learning is now one of the most hot topics around the world. Well it can even be said as the new electricity in today's world. But to be precise what is Machine Learning, well it's just one way of teaching the machine by feeding the large amount of data. To know more about Machine learning and its algorithms you can refer to some links that is provided in the Reference sections of this article.

Today, we will create a Image Classifier of our own which can distinguish whether a **given pic is of a dog or cat** or something else depending upon your fed data. To achieve our goal, we will use one of the famous machine learning algorithms out there which is used for Image Classification i.e. Convolutional Neural Network(or CNN).
So basically what is CNN – as we know its an machine learning algorithm for machines to understand the features of image with a foresight and remember the features to guess whether the name of the new image feeded to the machine. Since its not an article explaining the CNN so i'll add some links in the end if you guys are interested how CNN works and behaves.

So after going through all those links let us see how to create our very own cat-vs-dog image classifier. For the dataset we will use the kaggle dataset of cat-vs-dog:

- train dataset- link
- test dataset- link

Now after getting the data set we need to preprocess the data a bit and provide labels to each of the image given there during training the data set. To do so we can see that name of each image of training data set is either start with “cat” or “dog” so we will use that to our advantage then we use one hot encoder for machine to understand the labels(cat[1, 0] or dog[0, 1]).

```
def label_img(img):
    word_label = img.split('.')[-3]

 # DIY One hot encoder
    if word_label == 'cat': return [1, 0]
    elif word_label == 'dog': return [0, 1]
```

Libraries Required :

- **TFLearn** – Deep learning library featuring a higher-level API for TensorFlow used to create layers of our CNN
- **tqdm** – Instantly make your loops show a smart progress meter, just for simple designing sake
- **numpy** – To process the image matrices
- **open-cv** – To process the image like converting them to grayscale and etc.
- **os** – To access the file system to read the image from the train and test directory from our machines
- **random** – To shuffle the data to overcome the biasing
- **matplotlib** – To display the result of our predictive outcome.
- **tensorflow** – Just to use the tensorboard to compare the loss and adam curve our result data or obtained log.

TRAIN_DIR and TEST_DIR should be set according to the user convenience and play with the basic hyperparameters like epoch, learning rate, etc to improve the accuracy. I have converted the image to grayscale so that we will only have to deal with 2-d matrix otherwise 3-d matrix is tough to directly apply CNN to, especially not recommended for beginners. Below here is the code which is heavily commented or otherwise you can find the code here in my github account from this link.

```
# Python program to create
# Image Classifier using CNN

# Importing the required libraries
import cv2
import os
import numpy as np
from random import shuffle
from tqdm import tqdm

'''Setting up the env'''

TRAIN_DIR = 'E:/dataset / Cats_vs_Dogs / train'
TEST_DIR = 'E:/dataset / Cats_vs_Dogs / test1'
IMG_SIZE = 50
LR = 1e-3



'''Setting up the model which will help with tensorflow models'''
MODEL_NAME = 'dogsvscats-{}-{}.model'.format(LR, '6conv-basic')
```

```
'''Labelling the dataset'''
def label_img(img):
    word_label = img.split('.')[-3]
    # DIY One hot encoder
    if word_label == 'cat': return [1, 0]
    elif word_label == 'dog': return [0, 1]

'''Creating the training data'''
def create_train_data():
    # Creating an empty list where we should the store the training data
    # after a little preprocessing of the data
    training_data = []

    # tqdm is only used for interactive loading
    # loading the training data
    for img in tqdm(os.listdir(TRAIN_DIR)):

        # labeling the images
        label = label_img(img)

        path = os.path.join(TRAIN_DIR, img)

        # loading the image from the path and then converting them into
        # greyscale for easier covnet prob
        img = cv2.imread(path, cv2.IMREAD_GRAYSCALE)

        # resizing the image for processing them in the covnet
        img = cv2.resize(img, (IMG_SIZE, IMG_SIZE))

        # final step-forming the training data list with numpy array of
the images
        training_data.append([np.array(img), np.array(label)])

    # shuffling of the training data to preserve the random state of our
data
    shuffle(training_data)

    # saving our trained data for further uses if required
    np.save('train_data.npy', training_data)
    return training_data

'''Processing the given test data'''
# Almost same as processing the traning data but
# we dont have to label it.
def process_test_data():
    testing_data = []
    for img in tqdm(os.listdir(TEST_DIR)):
        path = os.path.join(TEST_DIR, img)
        img_num = img.split('.')[0]
        img = cv2.imread(path, cv2.IMREAD_GRAYSCALE)
        img = cv2.resize(img, (IMG_SIZE, IMG_SIZE))
        testing_data.append([np.array(img), img_num])

    shuffle(testing_data)
    np.save('test_data.npy', testing_data)
    return testing_data

'''Running the training and the testing in the dataset for our model'''
train_data = create_train_data()
test_data = process_test_data()

# train_data = np.load('train_data.npy')
# test_data = np.load('test_data.npy')
'''Creating the neural network using tensorflow'''
# Importing the required libraries
import tflearn
from tflearn.layers.conv import conv_2d, max_pool_2d
from tflearn.layers.core import input_data, dropout, fully_connected
from tflearn.layers.estimator import regression

import tensorflow as tf
tf.reset_default_graph()
convnet = input_data(shape =[None, IMG_SIZE, IMG_SIZE, 1], name ='input')

convnet = conv_2d(convnet, 32, 5, activation ='relu')
convnet = max_pool_2d(convnet, 5)

convnet = conv_2d(convnet, 64, 5, activation ='relu')
convnet = max_pool_2d(convnet, 5)

convnet = conv_2d(convnet, 128, 5, activation ='relu')
convnet = max_pool_2d(convnet, 5)

convnet = conv_2d(convnet, 64, 5, activation ='relu')
convnet = max_pool_2d(convnet, 5)

convnet = conv_2d(convnet, 32, 5, activation ='relu')
convnet = max_pool_2d(convnet, 5)

convnet = fully_connected(convnet, 1024, activation ='relu')
convnet = dropout(convnet, 0.8)

convnet = fully_connected(convnet, 2, activation ='softmax')
convnet = regression(convnet, optimizer ='adam', learning_rate = LR,
      loss ='categorical_crossentropy', name ='targets')

model = tflearn.DNN(convnet, tensorboard_dir ='log')

# Splitting the testing data and training data
```

```
train = train_data[:-500]
test = train_data[-500:]

'''Setting up the features and lables'''
# X-Features & Y-Labels

X = np.array([i[0] for i in train]).reshape(-1, IMG_SIZE, IMG_SIZE, 1)
Y = [i[1] for i in train]
test_x = np.array([i[0] for i in test]).reshape(-1, IMG_SIZE, IMG_SIZE, 1)
test_y = [i[1] for i in test]

'''Fitting the data into our model'''
# epoch = 5 taken
model.fit({'input': X}, {'targets': Y}, n_epoch = 5,
    validation_set =({'input': test_x}, {'targets': test_y}),
    snapshot_step = 500, show_metric = True, run_id = MODEL_NAME)
model.save(MODEL_NAME)

'''Testing the data'''
import matplotlib.pyplot as plt
# if you need to create the data:
# test_data = process_test_data()
# if you already have some saved:
test_data = np.load('test_data.npy')

fig = plt.figure()

for num, data in enumerate(test_data[:20]):
    # cat: [1, 0]
    # dog: [0, 1]

    img_num = data[1]
    img_data = data[0]

    y = fig.add_subplot(4, 5, num + 1)
    orig = img_data
    data = img_data.reshape(IMG_SIZE, IMG_SIZE, 1)

    # model_out = model.predict([data])[0]
    model_out = model.predict([data])[0]

    if np.argmax(model_out) == 1: str_label ='Dog'
    else: str_label ='Cat'

    y.imshow(orig, cmap ='gray')
    plt.title(str_label)
    y.axes.get_xaxis().set_visible(False)
    y.axes.get_yaxis().set_visible(False)
plt.show()
```

The output image will not be very clear since all the image are reduce to 50X50 for machine to process fast though the tradeoff between speed and loss.
And to access the tensorboard use the following command in your cmd(Windows user)

```
tensorboard --logdir=foo:C:\Users\knapseck\Desktop\Dev\Cov_Net\log
```

Output:

https://www.geeksforgeeks.org/image-classifier-using-cnn/

Hidden Markov Model

Markov Decision Process

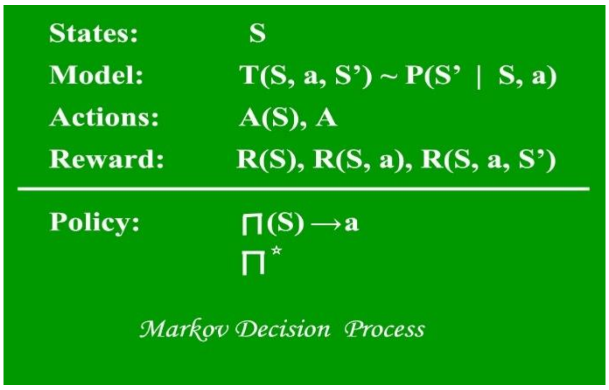
Reinforcement Learning :

Reinforcement Learning is a type of Machine Learning. It allows machines and software agents to automatically determine the ideal behavior within a specific context, in order to maximize its performance. Simple reward feedback is required for the agent to learn its behavior; this is known as the reinforcement signal.

There are many different algorithms that tackle this issue. As a matter of fact, Reinforcement Learning is defined by a specific type of problem, and all its solutions are classed as Reinforcement Learning algorithms. In the problem, an agent is supposed to decide the best action to select based on his current state. When this step is repeated, the problem is known as a **Markov Decision Process**.

A **Markov Decision Process (MDP)** model contains:

- A set of possible world states S.
- A set of Models.
- A set of possible actions A.
- A real valued reward function R(s,a).
- A policy the solution of **Markov Decision Process**.



States:	S
Model:	$T(S, a, S') \sim P(S' \mid S, a)$
Actions:	$A(S), A$
Reward:	$R(S), R(S, a), R(S, a, S')$
Policy:	$\Pi(S) \rightarrow a$ Π^*

Markov Decision Process

What is a State?

A **State** is a set of tokens that represent every state that the agent can be in.

What is a Model?

A **Model** (sometimes called Transition Model) gives an action's effect in a state. In particular, T(S, a, S') defines a transition T where being in state S and taking an action 'a' takes us to state S' (S and S' may be same). For stochastic actions (noisy, non-deterministic) we also define a probability P(S'|S,a) which represents the probability of reaching a state S' if action 'a' is taken in state S. Note Markov property states that the effects of an action taken in a state depend only on that state and not on the prior history.

What is Actions?

An **Action** A is set of all possible actions. A(s) defines the set of actions that can be taken being in state S.

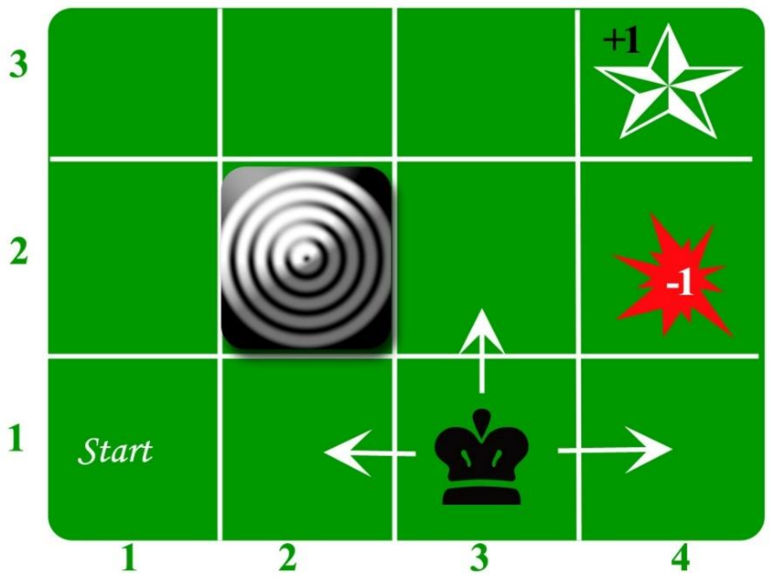
What is a Reward?

A **Reward** is a real-valued reward function. R(s) indicates the reward for simply being in the state S. R(S,a) indicates the reward for being in a state S and taking an action 'a'. R(S,a,S') indicates the reward for being in a state S, taking an action 'a' and ending up in a state S'.

What is a Policy?

A **Policy** is a solution to the Markov Decision Process. A policy is a mapping from S to a. It indicates the action 'a' to be taken while in state S.

Let us take the example of a grid world:



An agent lives in the grid. The above example is a 3*4 grid. The grid has a START state(grid no 1,1). The purpose of the agent is to wander around the grid to finally reach the Blue Diamond (grid no 4,3). Under all circumstances, the agent should avoid the Fire grid (orange color, grid no 4,2). Also the grid no 2,2 is a blocked grid, it acts like a wall hence the agent cannot enter it.

The agent can take any one of these actions: **UP, DOWN, LEFT, RIGHT**

Walls block the agent path, i.e., if there is a wall in the direction the agent would have taken, the agent stays in the same place. So for example, if the agent says LEFT in the START grid he would stay put in the START grid.

First Aim: To find the shortest sequence getting from START to the Diamond. Two such sequences can be found:

- **RIGHT RIGHT UP UP RIGHT**
- **UP UP RIGHT RIGHT RIGHT**

Let us take the second one (UP UP RIGHT RIGHT RIGHT) for the subsequent discussion.
The move is now noisy. 80% of the time the intended action works correctly. 20% of the time the action agent takes causes it to move at right angles. For example, if the agent says UP the probability of going UP is 0.8 whereas the probability of going LEFT is 0.1 and probability of going RIGHT is 0.1 (since LEFT and RIGHT is right angles to UP).

The agent receives rewards each time step:-

- Small reward each step (can be negative when can also be term as punishment, in the above example entering the Fire can have a reward of -1).
- Big rewards come at the end (good or bad).
- The goal is to Maximize sum of rewards.

Chinese Room Argument in Artificial Intelligence

When we ask, '***Is artificial intelligence (AI) possible***?' We really ask *'Can we create consciousness in computers'* ?

The Chinese room argument holds that a program cannot give a computer a "mind", "understanding" or "consciousness"" regardless of how intelligently or human-like the program may make the computer behave/ [Source Wiki]

In 1980, John Searle argued that Turing Test could not be used to determine "*whether or not a machine is considered as intelligent like humans*". He argued that any machine like ELIZA and PARRY could easily pass Turing Test simply by manipulating symbols of which they had no understanding. Without understanding, they could not be described as "**thinking**" in the same sense people do.

Configuration

John imagines himself (instead of machine) as non-Chinese person sitting inside the room isolated from another Chinese person who is outside the room tries to communicate. He is provided a list of Chinese characters and an instruction book explaining in detail the rules according to which strings (sequences) of characters may be formed, but without giving the meaning of the characters. That means he has a book with an English version of the computer program, along with sufficient paper, pencils, erasers, and filing cabinets.

In this thought experiment, a person in the "Chinese room" is passed questions from outside the room, and consults a library of books to formulate an answer

Now he recieve all the messages posted through a slot in the door written in Chinese language. He will process all the symbols according to program instructions and produces the chinese characters as output like:

- If he finds Chinese symbol like , he returns symbol
- If he finds Chinese symbol like ДКЯЕ, he returns symbol Љbdъ

Actually instruction book contains so many rules that contains input symbols and their respective output symbol. He just need to locate the input Chinese symbol and return the corresponding Chinese symbol as a output.
Now, the argument goes on, a computer(machine), is just like this man, in that it does nothing more than follow the rules given in an instruction book (the program). It does not understand the meaning of the questions given to it nor its own answers, and thus cannot be said to be thinking. *The fact is that inside person has no understanding of Chinese language but still he manage to communicate with outside person in Chinese language perfectly.*

Compare the **John** with machine in Turing Test, the machine may have huge collection of database containing questions and answers. When a interrogator ask the question, the machine is simply locating the question in the database and returning the corresponding answer to the interrogator. The whole scenario would seems like that human is returning the answer unlike machine.

Hence the machine in configuration has no understanding of those questions and answers, without "**understanding**" (or "**intentionality**"), we cannot describe what the machine is doing as "**thinking**" and, since it does not think, it does not have a "**mind**" in anything like the normal sense of the word. Therefore we can't consider machine as intelligent.

Data Processing

Getting started with Classification

Introduction

As the name suggests, Classification is the task of "classifying things" into sub-categories.But, by a machine! If that doesn't sound like much, imagine your computer being able to differentiate between you and a stranger. Between a potato and a tomato. Between an A grade and a F- .

Yeah. It sounds interesting now!

In Machine Learning and Statistics, Classification is the problem of identifying to which of a set of categories (sub populations), a new observation belongs to, on the basis of a training set of data containing observations and whose categories membership is known.

Types of Classification

Classification is of two types:

- **Binary Classification** : When we have to categorize given data into 2 distinct classes. Example – On the basis of given health conditions of a person, we have to determine whether the person has a certain disease or not.
- **Multiclass Classification** : The number of classes is more than 2. For Example – On the basis of data about different species of flowers, we have to determine which specie does our observation belong to.

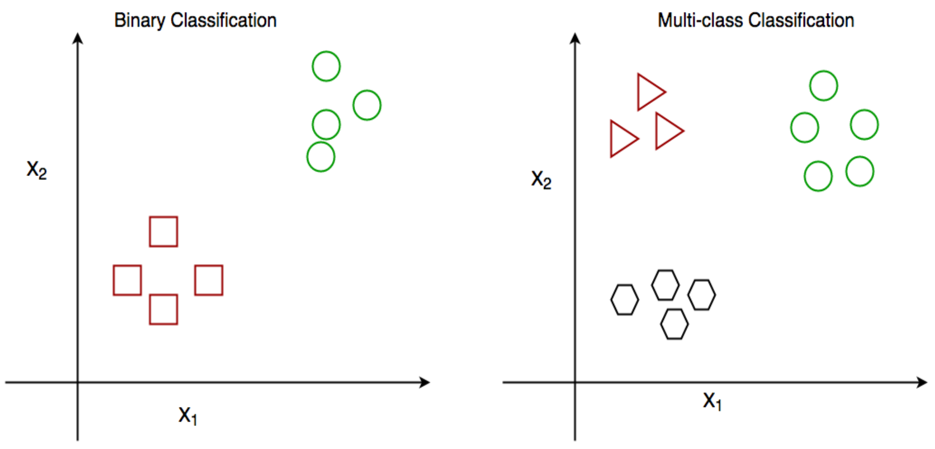


Fig : Binary and Multiclass Classification. Here x1 and x2 are our variables upon which the class is predicted.

How does classification works?

Suppose we have to predict whether a given patient has a certain disease or not, on the basis of 3 variables, called features.

Which means there are two possible outcomes:

1. The patient has the said disease. Basically a result labelled "Yes" or "True".
2. The patient is disease free. A result labelled "No" or "False".

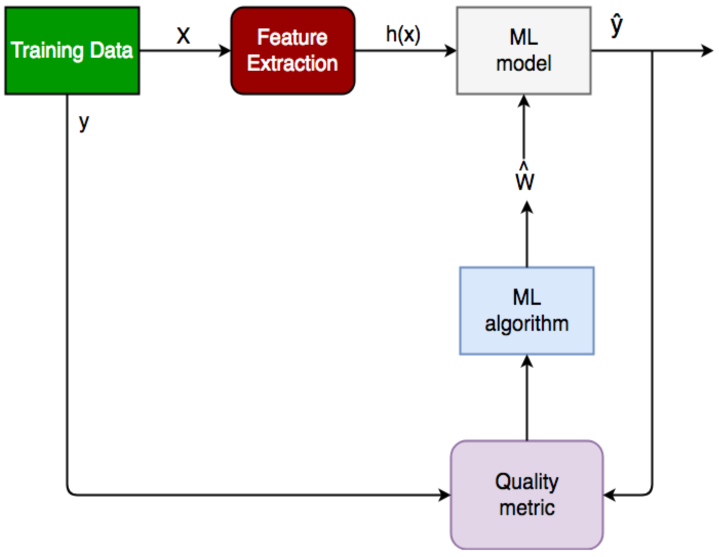
This is a binary classification problem.

We have a set of observations called training data set, which comprises of sample data with actual classification results. We train a model, called Classifier on this data set, and use that model to predict whether a certain patient will have the disease or not.

The outcome, thus now depends upon :

1. How well these features are able to "map" to the outcome.
2. The quality of our data set. By quality I refer to statistical and Mathematical qualities.
3. How well our Classifier generalizes this relationship between the features and the outcome.
4. The values of the x1 and x2.

Following is the generalized block diagram of the classification task.



Generalized Classification Block Diagram.

1. X : pre-classified data, in the form of a N*M matrix. N is the no. of observations and M is the number of features
2. y : An N-d vector corresponding to predicted classes for each of the N observations.
3. Feature Extraction : Extracting valuable information from input X using a series of transforms.
4. ML Model : The "Classifier" we'll train.
5. y' : Labels predicted by the Classifier.
6. Quality Metric : Metric used for measuring the performance of the model.
7. ML Algorithm : The algorithm that is used to update weights w', which update the model and "learns" iteratively.

Types of Classifiers (algorithms)

There are various types of classifiers. Some of them are :

- Linear Classifiers : Logistic Regression
- Tree Based Classifiers : Decision Tree Classifier
- Support Vector Machines
- Artificial Neural Networks
- Bayesian Regression
- Gaussian Naive Bayes Classifiers
- Stochastic Gradient Descent (SGD) Classifier
- Ensemble Methods : Random Forests, AdaBoost, Bagging Classifier, Voting Classifier, ExtraTrees Classifier

Detailed description of these methodologies is beyond an article!

Practical Applications of Classification

1. Google's self driving car uses deep learning enabled classification techniques which enables it to detect and classify obstacles.
2. Spam E-mail filtering is one of the most widespread and well recognized uses of Classification techniques.
3. Detecting Health Problems, Facial Recognition, Speech Recognition, Object Detection, Sentiment Analysis all use Classification at their core.

Implementation

Let's get a hands on experience at how Classification works.We are going to study about various Classifiers and see a rather simple analytical comparison of their performance on a well known, standard data set, the Iris data set.

Requirements for running the given script

1. Python 2.7
2. Scipy and Numpy
3. Matplotlib for data visualization
4. Pandas for data i/o
5. Scikit-learn Provides all the classifiers

Python Implementation- Github link to the Project

Conclusion

Classification is a very vast field of study. Even though it comprises of a small part of Machine Learning as a whole, it is one of the most important ones.

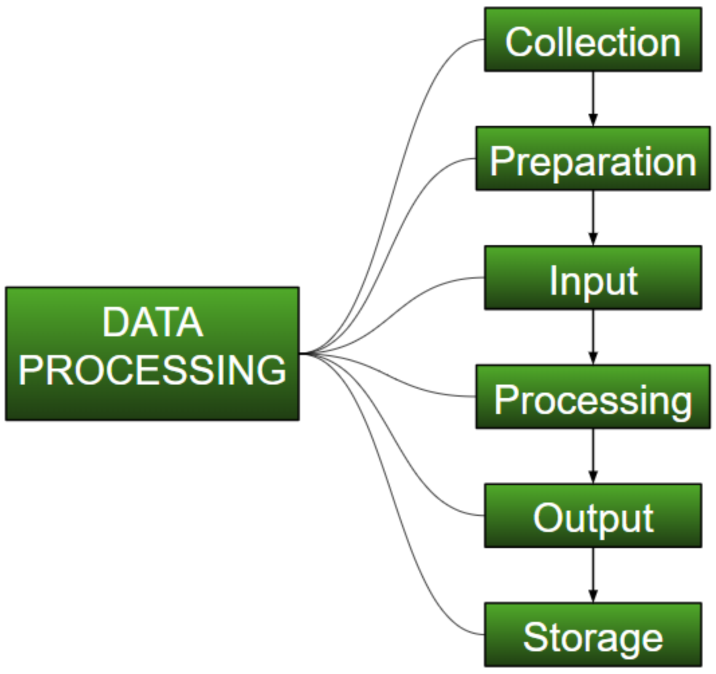
That's all for now. In the next article, we will see how Classification works in practice and get our hands dirty with Python Code.

This article is contributed by **Sarthak Yadav**. If you like GeeksforGeeks and would like to contribute, you can also write an article and mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

ML | Understanding Data Processing

Data Processing is a task of converting data from a given form to a much more usable and desired form i.e. making it more meaningful and informative. Using Machine Learning algorithms, mathematical modelling and statistical knowledge, this entire process can be automated. The output of this complete process can be in any desired form like graphs, videos, charts, tables, images and many more, depending on the task we are performing and the requirements of the machine. This might seem to be simple but when it comes to really big organizations like Twitter, Facebook, Administrative bodies like Paliament, UNESCO and health sector organisations, this entire process needs to be performed in a very structured manner. So, the steps to perform are as follows:



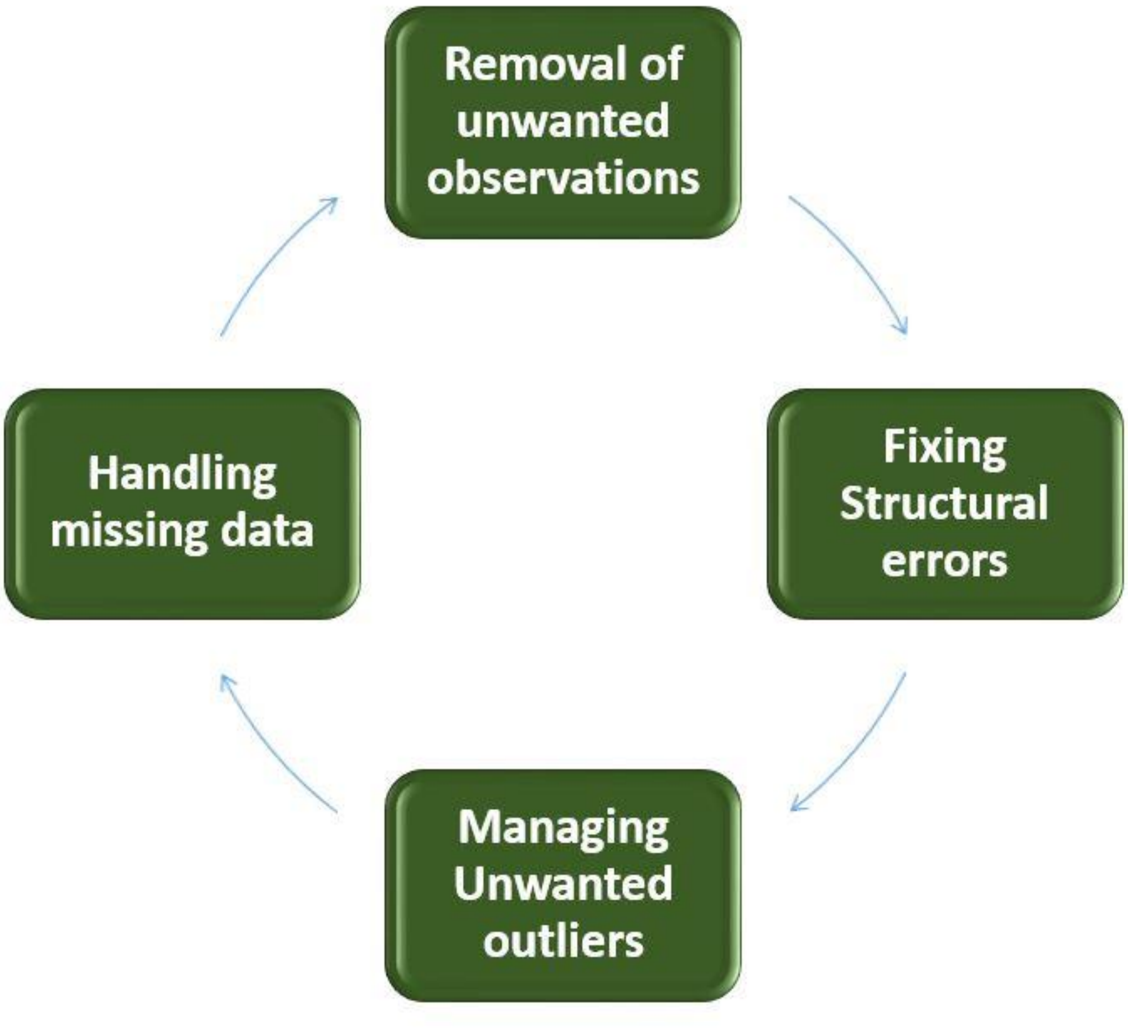
- **Collection :**
  The most crucial step when starting with ML is to have data of good quality and accuracy. Data can be collected from any authenticated source like data.gov.in, Kaggle or UCI dataset repository.For example, while preparing for a competitive exam, students study from the best study material that they can access so that they learn the best to obtain the best results. In the same way, high-quality and accurate data will make the learning process of the model easier and better and at the time of testing, the model would yield state of the art results.
  A huge amount of capital, time and resources are consumed in collecting data. Organizations or researchers have to decide what kind of data they need to execute their tasks or research.
  Example: Working on the Facial Expression Recognizer, needs a large number of images having a variety of human expressions. Good data ensures that the results of the model are valid and can be trusted upon.
- **Preparation :**
  The collected data can be in a raw form which can't be directly fed to the machine. So, this is a process of collecting datasets from different sources, analyzing these datasets and then constructing a new dataset for further processing and exploration. This preparation can be performed either manually or from the automatic approach. Data can also be prepared in numeric forms also which would fasten the model's learning.
  Example: An image can be converted to a matrix of N X N dimensions, the value of each cell will indicate image pixel.
- **Input :**
  Now the prepared data can be in the form that may not be machine-readable, so to convert this data to readable form, some conversion algorithms are needed. For this task to be executed, high computation and accuracy is needed. Example: Data can be collected through the sources like MNIST Digit data(images), twitter comments, audio files, video clips.
- **Processing :**
  This is the stage where algorithms and ML techniques are required to perform the instructions provided over a large volume of data with accuracy and optimal computation.
- **Output :**
  In this stage, results are procured by the machine in a meaningful manner which can be inferred easily by the user. Output can be in the form of reports, graphs, videos, etc
- **Storage :**
  This is the final step in which the obtained output and the data model data and all the useful information are saved for the future use.

Data Cleansing | Introduction

Introduction:
Data cleaning is one of the important parts of machine learning. It plays a significant part in building a model. Data Cleaning is one of those things that everyone does but no one really talks about. It surely isn't the fanciest part of machine learning and at the same time, there aren't any hidden tricks or secrets to uncover. However, proper data cleaning can make or break your project. Professional data scientists usually spend a very large portion of their time on this step.
Because of the belief that, **"Better data beats fancier algorithms"**.
If we have a well-cleaned dataset, we can get desired results even with a very simple algorithm, which can prove very beneficial at times.

Obviously, different types of data will require different types of cleaning. However, this systematic approach can always serve as a good starting point.

Steps involved in Data Cleaning



1. **Removal of unwanted observations**
   This includes deleting duplicate/ redundant or irrelevant values from your dataset. Duplicate observations most frequently arise during data collection and Irrelevant observations are those that don't actually fit the specific problem that you're trying to solve.

   - Redundant observations alter the efficiency by a great extent as the data repeats and may add towards the correct side or towards the incorrect side, thereby producing unfaithful results.
   - Irrelevant observations are any type of data that is of no use to us and can be removed directly.
2. **Fixing Structural errors**
   The errors that arise during measurement, transfer of data or other similar situations are called structural errors. Structural errors include typos in the name of features, same attribute with different name, mislabeled classes, i.e. separate classes that should really be the same or inconsistent capitalization.

   - For example, the model will treat America and america as different classes or values, though they represent the same value or red, yellow and red-yellow as different classes or attributes, though one class can be included in other two classes. So, these are some structural errors that make our model inefficient and gives poor quality results.
3. **Managing Unwanted outliers**
   Outliers can cause problems with certain types of models. For example, linear regression models are less robust to outliers than decision tree models. Generally, we should not remove outliers until we have a legitimate reason to remove them. Sometimes, removing them improves performance, sometimes not. So, one must have a good reason to remove the outlier, such as suspicious measurements that are unlikely to be the part of real data.
4. **Handling missing data**
   Missing data is a deceptively tricky issue in machine learning. We cannot just ignore or remove the missing observation. They must be handled carefully as they can be an indication of something important. The two most common ways to deal with missing data are:
   1. Dropping observations with missing values.

      Dropping missing values is sub-optimal because when you drop observations, you drop information.

      - The fact that the value was missing may be informative in itself.
      - Plus, in the real world, you often need to make predictions on new data even if some of the features are missing!
   2. Imputing the missing values from past observations.

      Imputing missing values is sub-optimal because the value was originally missing but you filled it in, which always leads to a loss in information, no matter how sophisticated your imputation method is.

      - Again, “missingness” is almost always informative in itself, and you should tell your algorithm if a value was missing.
      - Even if you build a model to impute your values, you’re not adding any real information. You’re just reinforcing the patterns already provided by other features.

   Both of these approaches are sub-optimal because dropping an observation means dropping information, thereby reducing data and imputing values also is sub-optimal as we fil the values that were not present in the actual dataset, which leads to a loss of information.

Missing data is like missing a puzzle piece. If you drop it, that’s like pretending the puzzle slot isn’t there. If you impute it, that’s like trying to squeeze in a piece from somewhere else in the puzzle.
So, missing data is always informative and indication of something important. And we must aware our algorithm of missing data by flagging it. By using this technique of flagging and filling, you are essentially allowing the algorithm to estimate the optimal constant for missingness, instead of just filling it in with the mean.

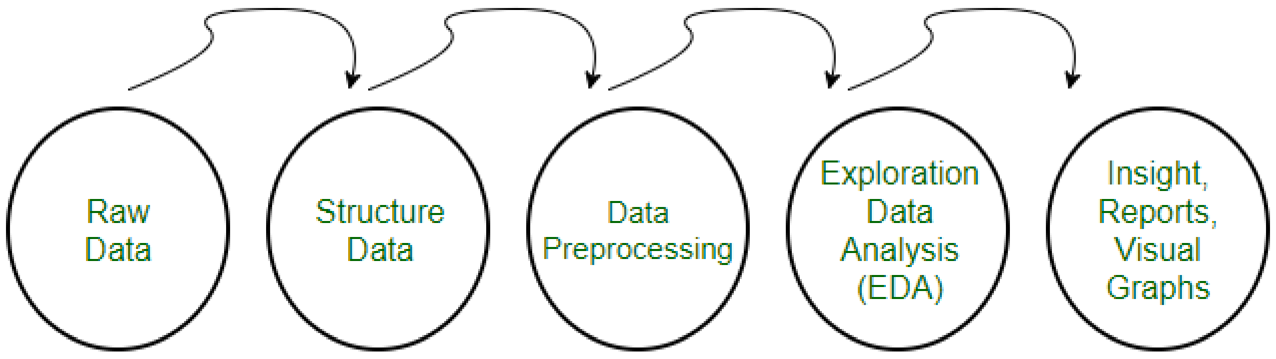
Some data cleansing tools

- Openrefine
- Trifacta Wrangler
- TIBCO Clarity
- Cloudingo
- IBM Infosphere Quality Stage

Conclusion
So, we have discussed four different steps in data cleaning to make the data more reliable and to produce good results. After properly completing the Data Cleaning steps, we'll have a robust dataset that avoids many of the most common pitfalls. This step should not be rushed as it proves very beneficial in the further process.

Data Preprocessing for Machine learning in Python

• Pre-processing refers to the transformations applied to our data before feeding it to the algorithm.
• Data Preprocessing is a technique that is used to convert the raw data into a clean data set. In other words, whenever the data is gathered from different sources it is collected in raw format which is not feasible for the analysis.


Raw Data
Structure Data
Data Preprocessing
Exploration Data Analysis (EDA)
Insight, Reports, Visual Graphs

Need of Data Preprocessing
• For achieving better results from the applied model in Machine Learning projects the format of the data has to be in a proper manner. Some specified Machine Learning model needs information in a specified format, for example, Random Forest algorithm does not support null values, therefore to execute random forest algorithm null values have to be managed from the original raw data set.
• Another aspect is that data set should be formatted in such a way that more than one Machine Learning and Deep Learning algorithms are executed in one data set, and best out of them is chosen.

This article contains 3 different data preprocessing techniques for machine learning.

The Pima Indian diabetes dataset is used in each technique.
This is a binary classification problem where all of the attributes are numeric and have different scales.
It is a great example of a dataset that can benefit from pre-processing.
You can find this dataset on the UCI Machine Learning Repository webpage.
Note that the program might not run on Geeksforgeeks IDE, but it can run easily on your local python interpreter, provided, you have installed the required libraries.

1. Rescale Data
• When our data is comprised of attributes with varying scales, many machine learning algorithms can benefit from rescaling the attributes to all have the same scale.
• This is useful for optimization algorithms in used in the core of machine learning algorithms like gradient descent.
• It is also useful for algorithms that weight inputs like regression and neural networks and algorithms that use distance measures like K-Nearest Neighbors.
• We can rescale your data using scikit-learn using the MinMaxScaler class.

```
# Python code to Rescale data (between 0 and 1)
import pandas
import scipy
import numpy
from sklearn.preprocessing import MinMaxScaler
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/pima-
indians-diabetes/pima-indians-diabetes.data"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age',
'class']
dataframe = pandas.read_csv(url, names=names)
array = dataframe.values

# separate array into input and output components
X = array[:,0:8]
Y = array[:,8]
scaler = MinMaxScaler(feature_range=(0, 1))
rescaledX = scaler.fit_transform(X)

# summarize transformed data
numpy.set_printoptions(precision=3)
print(rescaledX[0:5,:])
```

After rescaling see that all of the values are in the range between 0 and 1.

```
Output

[[ 0.353  0.744  0.59   0.354  0.0    0.501  0.234  0.483]
 [ 0.059  0.427  0.541  0.293  0.0    0.396  0.117  0.167]
 [ 0.471  0.92   0.525  0.     0.0    0.347  0.254  0.183]
 [ 0.059  0.447  0.541  0.232  0.111  0.419  0.038  0.0  ]
 [ 0.0    0.688  0.328  0.354  0.199  0.642  0.944  0.2  ]]
```

2. Binarize Data (Make Binary)

• We can transform our data using a binary threshold. All values above the threshold are marked 1 and all equal to or below are marked as 0.
• This is called binarizing your data or threshold your data. It can be useful when you have probabilities that you want to make crisp values. It is also useful when feature engineering and you want to add new features that indicate something meaningful.
• We can create new binary attributes in Python using scikit-learn with the Binarizer class.

```
# Python code for binarization
from sklearn.preprocessing import Binarizer
import pandas
import numpy
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/pima-indians-diabetes/pima-indians-diabetes.data"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = pandas.read_csv(url, names=names)
array = dataframe.values

# separate array into input and output components
X = array[:,0:8]
Y = array[:,8]
binarizer = Binarizer(threshold=0.0).fit(X)
binaryX = binarizer.transform(X)

# summarize transformed data
numpy.set_printoptions(precision=3)
print(binaryX[0:5,:])
```

We can see that all values equal or less than 0 are marked 0 and all of those above 0 are marked 1.

Output

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

3. Standardize Data

• Standardization is a useful technique to transform attributes with a Gaussian distribution and differing means and standard deviations to a standard Gaussian distribution with a mean of 0 and a standard deviation of 1.
• We can standardize data using scikit-learn with the StandardScaler class.

```
# Python code to Standardize data (0 mean, 1 stdev)
from sklearn.preprocessing import StandardScaler
import pandas
import numpy
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/pima-indians-diabetes/pima-indians-diabetes.data"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age',
'class']
dataframe = pandas.read_csv(url, names=names)
array = dataframe.values

# separate array into input and output components
X = array[:,0:8]
Y = array[:,8]
scaler = StandardScaler().fit(X)
rescaledX = scaler.transform(X)

# summarize transformed data
numpy.set_printoptions(precision=3)
print(rescaledX[0:5,:])
```

The values for each attribute now have a mean value of 0 and a standard deviation of 1.

Output

```
[[ 0.64   0.848  0.15   0.907 -0.693  0.204  0.468  1.426]
 [-0.845 -1.123 -0.161  0.531 -0.693 -0.684 -0.365 -0.191]
 [ 1.234  1.944 -0.264 -1.288 -0.693 -1.103  0.604 -0.106]
 [-0.845 -0.998 -0.161  0.155  0.123 -0.494 -0.921 -1.042]
 [-1.142  0.504 -1.505  0.907  0.766  1.41   5.485 -0.02 ]]
```

Misc

Pattern Recognition | Introduction

Pattern is everything around in this digital world. A pattern can either be seen physically or it can be observed mathematically by applying algorithms.
Example: The colours on the clothes, speech pattern etc. In computer science, a pattern is represented using vector features values.

What is Pattern Recognition ?

Pattern recognition is the process of recognizing patterns by using machine learning algorithm. Pattern recognition can be defined as the classification of data based on knowledge already gained or on statistical information extracted from patterns and/or their representation. One of the important aspects of the pattern recognition is its application potential.
Examples: Speech recognition, speaker identification, multimedia document recognition (MDR), automatic medical diagnosis.

In a typical pattern recognition application, the raw data is processed and converted into a form that is amenable for a machine to use. Pattern recognition involves classification and cluster of patterns.

- In classification, an appropriate class label is assigned to a pattern based on an abstraction that is generated using a set of training patterns or domain knowledge. Classification is used in supervised learning.
- Clustering generated a partition of the data which helps decision making, the specific decision making activity of interest to us. Clustering is used in an unsupervised learning.

Features may be represented as continuous, discrete or discrete binary variables. A feature is a function of one or more measurements, computed so that it quantifies some significant characteristics of the object.
Example: consider our face then eyes, ears, nose etc are features of the face.

A set of features that are taken together, forms the **features vector**.
Example: In the above example of face, if all the features (eyes, ears, nose etc) taken together then the sequence is feature vector([eyes, ears, nose]). Feature vector is the sequence of a features represented as a d-dimensional column vector. In case of speech, MFCC (Melfrequency Cepstral Coefficent) is the spectral features of the speech. Sequence of first 13 features forms a feature vector.

Pattern recognition possesses the following features:

- Pattern recognition system should recognise familiar pattern quickly and accurate
- Recognize and classify unfamiliar objects
- Accurately recognize shapes and objects from different angles
- Identify patterns and objects even when partly hidden
- Recognise patterns quickly with ease, and with automaticity.

Training and Learning in Pattern Recognition

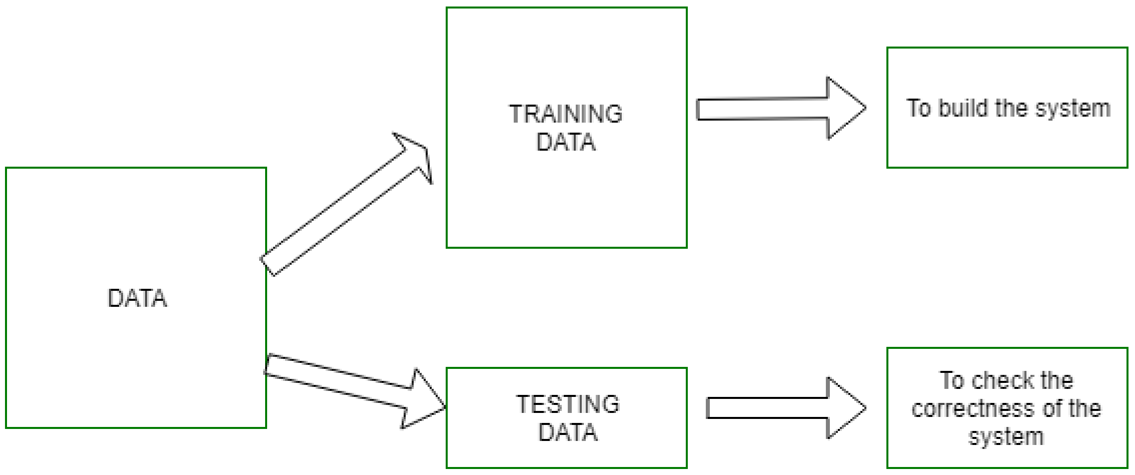
Learning is a phenomena through which a system gets trained and becomes adaptable to give result in an accurate manner. Learning is the most important phase as how well the system performs on the data provided to the system depends on which algorithms used on the data. Entire dataset is divided into two categories, one which is used in training the model i.e. Training set and the other that is used in testing the model after training, i.e. Testing set.

- **Training set:**

Training set is used to build a model. It consists of the set of images which are used to train the system. Training rules and algorithms used give relevant information on how to associate input data with output decision. The system is trained by applying these algorithms on the dataset, all the relevant information is extracted from the data and results are obtained. Generally, 80% of the data of the dataset is taken for training data.

- **Testing set:**

Testing data is used to test the system. It is the set of data which is used to verify whether the system is producing the correct output after being trained or not. Generally, 20% of the data of the dataset is used for testing. Testing data is used to measure the accuracy of the system. Example: a system which identifies which category a particular flower belongs to, is able to identify seven category of flowers correctly out of ten and rest others wrong, then the accuracy is 70 %



Real-time Examples and Explanations:
A pattern is a physical object or an abstract notion. While talking about the classes of animals, a description of an animal would be a pattern. While talking about various types of balls, then a description of a ball is a pattern. In the case balls considered as pattern, the classes could be football, cricket ball, table tennis ball etc. Given a new pattern, the class of the pattern is to be determined. The choice of attributes and representation of patterns is a very important step in pattern classification. A good representation is one which makes use of discriminating attributes and also reduces the computational burden in pattern classification.
An obvious representation of a pattern will be a **vector**. Each element of the vector can represent one attribute of the pattern. The first element of the vector will contain the value of the first attribute for the pattern being considered.
Example: While representing spherical objects, (25, 1) may be represented as an spherical object with 25 units of weight and 1 unit diameter. The class label can form a part of the vector. If spherical objects belong to class 1, the vector would be (25, 1, 1), where the first element represents the weight of the object, the second element, the diameter of the object and the third element represents the class of the object.

Advantages:

- Pattern recognition solves classification problems
- Pattern recognition solves the problem of fake bio metric detection.
- It is useful for cloth pattern recognition for visually impaired blind people.
- It helps in speaker diarization.
- We can recognise particular object from different angle.

Disadvantages:

- Syntactic Pattern recognition approach is complex to implement and it is very slow process.
- Sometime to get better accuracy, larger dataset is required.
- It cannot explain why a particular object is recognized.
  Example: my face vs my friend's face.

Applications:

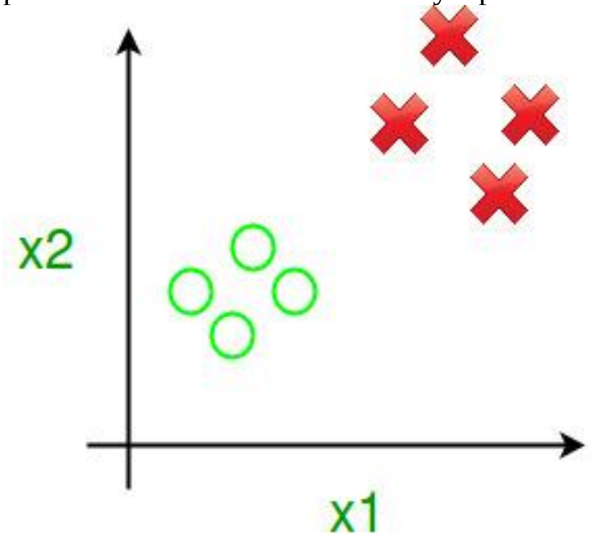
- **Image processing, segmentation and analysis**
  Pattern recognition is used to give human recognition intelligence to machine which is required in image processing.
- **Computer vision**
  Pattern recognition is used to extract meaningful features from given image/video samples and is used in computer vision for various applications like biological and biomedical imaging.

- **Seismic analysis**
  Pattern recognition approach is used for the discovery, imaging and interpretation of temporal patterns in seismic array recordings. Statistical pattern recognition is implemented and used in different types of seismic analysis models.
- **Radar signal classification/analysis**
  Pattern recognition and Signal processing methods are used in various applications of radar signal classifications like AP mine detection and identification.
- **Speech recognition**
  The greatest success in speech recognition has been obtained using pattern recognition paradigms. It is used in various algorithms of speech recognition which tries to avoid the problems of using a phoneme level of description and treats larger units such as words as pattern
- **Finger print identification**
  The fingerprint recognition technique is a dominant technology in the biometric market. A number of recognition methods have been used to perform fingerprint matching out of which pattern recognition approaches is widely used.

Calculate Efficiency Of Binary Classifier

Prerequisite: Getting started with Classification

In this article, we will discuss a method to calculate efficiency of **Binary classifier**. Let's assume there is problem where we have to classify a product which belongs to either class A or class B.



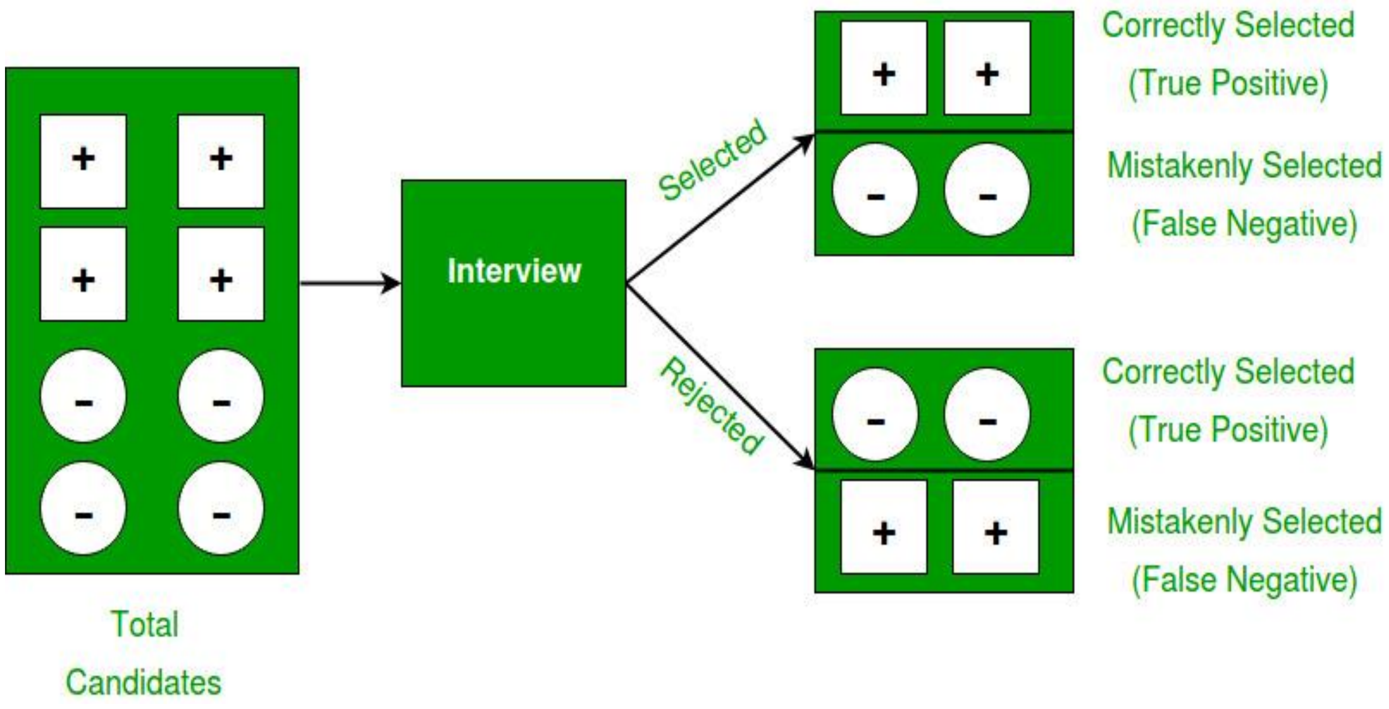
Let us define few statistical parameters :

TP (True Positive) = number of Class A products, which are classified as Class A products.
FN (False Negative) = number of Class A products, which are classified as Class B products.
TN (True Negative) = number of Class B products, which are classified as Class B products.
FP (False Positive) = number of Class B products, which are classified as Class A products.

```
FP = N-TP;       // where number N is the number of class A type products
FN = M-TN;      // where number M is the number of class B type products
```

We shall look at this example, to understand these parameters well.

If **(+)** denotes fit candidates for Job and **(-)** denotes unfit candidates for Job.



To calculate Efficiency of classifier we need to compute values of **Sensitivity, Specificity** and **Accuracy**.

Sensitivity measures the proportion of positives that are correctly identified as such.
Also known as **True positive rate**(TPR).

Specificity measures the proportion of negatives that are correctly identified as such.
Also known as **True negative rate**(TNR).

Accuracy measures how well the test predicts both TPR and TNR.

```
Sensitivity = ( TP / (TP+FN) ) * 100;
Specificity = ( TN/(TN+FP) ) * 100;
Accuracy = ( (TP+TN) / (TP+TN+FP+FN) ) * 100;
Efficiency = ( Sensitivity + Specificity + Accuracy ) / 3;
```

Let's take above example and compute efficiency of selection :

Say fit candidates belong to class A and unfit candidates belong to class B.

```
Before Interview :  N = 4 and M = 4

After Interview :
TP = 2
TN = 2
FP = N - TP = 2
FN = M - TN = 2

Sensitivity = 2/(2+2)*100 = 50
Specificity = 2/(2+2)*100 = 50
Accuracy    = (2+2)/(2+2+2+2)*100 = 50
Efficiency  = (50+50+50)/3 = 50
```

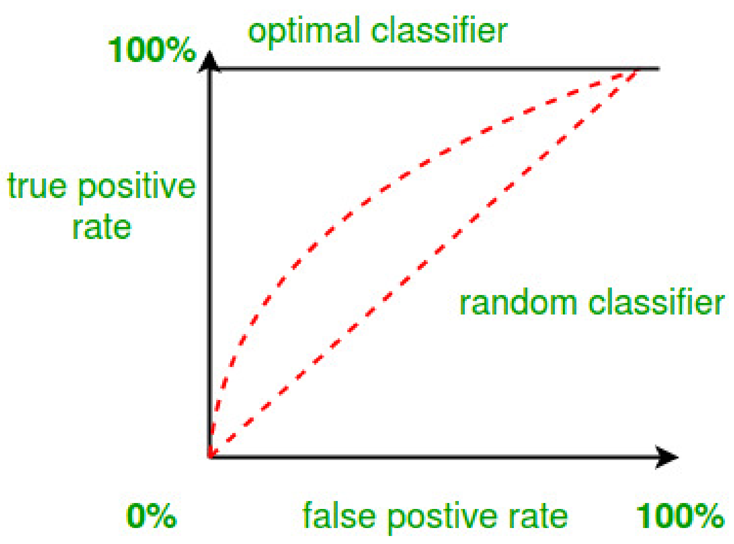
So,Efficiency of selection of candidates is **50% accurate.**

Other performance measures:

- **Error rate** = (FP + FN) / (TP + TN + FP + FN)
- **Precision** = TP / (TP + FP)
- **Recall** = TP / (TP + FN)
- **BCR (Balanced Classification Rate)** = 1/2* (TP / (TP + FN) + TN / (TN + FP))
- **AUC** = Area under ROC curve

Receiver Operating Characteristic Curve:

- Receiver operating characteristic(ROC) curve : 2-D curve parametrized by one parameter of the classification algorithm.
- AUC is always between 0 and 1.
- ROC curve can be obtained plotting TPR on y-axis and TNR on x-axis.
- AUC gives accuracy of the proposed model.



Cross Validation in Machine Learning

In machine learning, we couldn't fit the model on the training data and can't say that the model will work accurately for the real data. For this, we must assure that our model got the correct patterns from the data, and it is not getting up too much noise. For this purpose, we use the cross-validation technique.

Cross-Validation

Cross-validation is a technique in which we train our model using the subset of the data-set and then evaluate using the complementary subset of the data-set.

The three steps involved in cross-validation are as follows :

1. Reserve some portion of sample data-set.
2. Using the rest data-set train the model.
3. Test the model using the reserve portion of the data-set.

Methods of Cross Validation

Validation

In this method, we perform training on the 50% of the given data-set and rest 50% is used for the training purpose. The major drawback of this method is that we perform training on the 50% of the dataset, it may possible that the remaining 50% of the data contains some important information which we are leaving while training our model i.e higer bias.

LOOCV (Leave One Out Cross Validation)

In this method, we perform training on the whole data-set but leaves only one data-point of the available data-set and then iterates for each data-point. It has some advantages as well as disadvantages also.
An advantage of using this method is that we make use of all data points and hence it is low bias.
The major drawback of this method is that it leads to higher variation in the testing model as we are testing against one data point. If the data point is an outlier it can lead to higher variation. Another drawback is it takes a lot of execution time as it iterates over ‘the number of data points’ times.

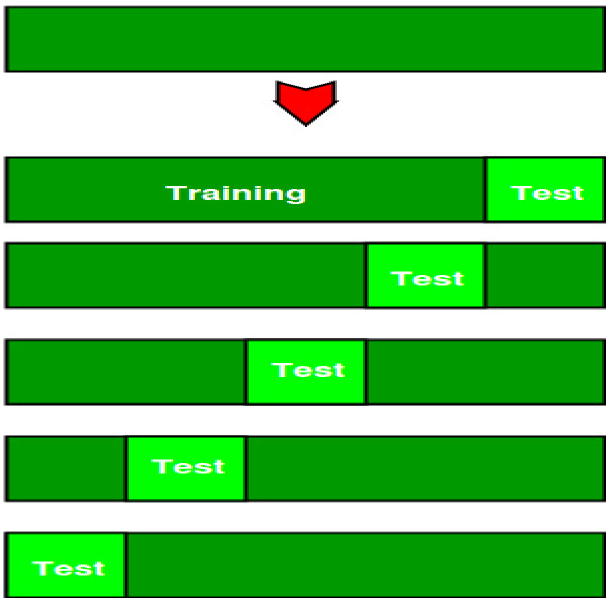
K-Fold Cross Validation

In this method, we split the data-set into k number of subsets(known as folds) then we perform training on the all the subsets but leave one(k-1) subset for the evaluation of the trained model. In this method, we iterate k times with a different subset reserved for testing purpose each time.

```
Note:
It is always suggested that the value of k should be 10 as the lower value
of k is takes towards validation and higher value of k leads to LOOCV method.
```

Example

The diagram below shows an example of the training subsets and evaluation subsets generated in k-fold cross-validation. Here, we have total 25 instances. In first iteration we use the first 20 percent of data for evaluation, and the remaining 80 percent for training([1-5] testing and [5-25] training) while in the second iteration we use the second subset of 20 percent for evaluation, and the remaining three subsets of the data for training([5-10] testing and [1-5 and 10-25] training), and so on.



```
Total instances: 25
Value of k      : 5

No. Iteration                  Training set observations                        Testing set
observations
 1        [ 5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24]    [0  1  2  3  4]
 2        [ 0   1   2   3   4  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24]    [5  6  7  8  9]
 3        [ 0   1   2   3   4   5   6   7   8   9  15  16  17  18  19  20  21  22  23  24]    [10  11  12  13
14]
 4        [ 0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  20  21  22  23  24]    [15  16  17  18
19]
 5        [ 0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19]    [20  21  22  23
24]
```

Comparision of train/test split to cross-validation

Advantages of train/test split:

1. This runs K times faster than Leave One Out cross-validation because K-fold cross-validation repeats the train/test split K-times.
2. Simpler to examine the detailed results of the testing process.

Advantages of cross-validation:

1. More accurate estimate of out-of-sample accuracy.
2. More “efficient” use of data as every observation is used for both training and testing.

Python code for k fold cross-validation.

```
# This code may not be run on GFG IDE
# as required packages are not found.

# importing cross-validation from sklearn package.
from sklearn import cross_validation

# value of K is 10.
data = cross_validation.KFold(len(train_set), n_folds=10, indices=False)
```

R vs Python in Datascience

Data science deals with identifying, representing and extracting meaningful information from data sources to be used to perform some business logics.The data scientist uses machine learning, statistics, probability, linear and logistic regression and more in order to make out some meaningful data. Finding patterns and similar combinations and cracking the best possible path way according to the business logic is the biggest job of analysis.

R, Python, SQL, SAS, Tableau, MATLAB, etc. are of the most useful tools for data science, R and Python being the most used ones. But still, it becomes confusing for any newbie to choose the better or the most suitable one among the two, R and Python. Let's try to visualize the difference.

Overview :

R	Python
R is a programming language and free software environment for statistical computing and graphics, supported by the R Foundation for Statistical Computing. It was designed by Ross Ihaka and Robert Gentleman and first released in August, 1993. It is widely used among statisticians and data miners for developing statistical software and data analysis.	Python is an Interpreted high-level programming language for general purpose programming. It was created by Guido Van Rossum and was first released in 1991. Python has a very clean and simple code syntax. It emphasizes code readability and thus debugging is also far more simpler and easier in Python.

Specialities for datascience :

R	Python
R packages cover advanced techniques which very useful for statistical work. The CRAN text view provides you with many useful R packages. R packages cover everything from Psychometrics to Genetics to Finance. On the other hand, Python, with the help of libraries like SciPy and packages like statsmodels, covers only the most common techniques.	R and Python are equally good for finding outliers in a data set, but for developing a web service to enable other people to upload datasets and find outliers, Python is better. People have built modules to create websites, interact with a variety of databases, and manage users in Python. In general, to create a tool or service that uses data analysis, Python is a better choice.

Functionalities :

R	Python
R has inbuilt functionalities for data analysis. R was built by eminent statisticians with statistics and data analysis in mind, so many tools that have been externally added to Python through packages are built in R by default.	Python is a general purpose programming language. So most of the data analysis functionalities are not inbuilt and are available through packages like Numpy and Pandas, which are available in PyPi(Python Package Index).

Key domains of application :

R	Python
Data visualization is a key aspect of analysis, as visual data is best understood. R packages like ggplot2, ggvis, lattice, etc. make data visualization easier in R. Python is catching up with packages like Bokeh, Matplotlib, etc. but is still far behind in this regard.	Python is better for deep learning. Packages like Lasagne, Caffe, Keras, Mxnet, OpenNN, Tensor flow, etc. allows development of deep neural networks far more simple in Python. Although some of these, like tensor flow, are being ported to R(packages like deepnet, H2O, etc.) but it is still better in Python.

Availability of Packages :

R	Python
R has hundreds of packages and ways to accomplish needful data science tasks. Although it allows to have desired perfection in completing the task, it makes it difficult for inexperienced developers to achieve certain goals.	Python relies on a few main packages, viz., Scikit learn and Pandas are the packages for machine learning data analysis respectively. It makes easier to accomplish required tasks but consequently it becomes difficult to achieve specialization.

Ultimately it's the job of data scientist itself to choose the most suitable language as needed. For statistical background, R might be a better option. But for the CS background or even a beginner, Python is most suitable option. But, it's better to have sound knowledge of both cause both might be useful at times in data science career.

ML using Python

Introduction To Machine Learning using Python

Machine learning is a type of artificial intelligence (AI) that provides computers with the ability to learn without being explicitly programmed. Machine learning focuses on the development of Computer Programs that can change when exposed to new data. In this article, we'll see basics of Machine Learning, and implementation of a simple machine learning algorithm using python.

Setting up the environment

Python community has developed many modules to help programmers implement machine learning. In this article, we will be using numpy, scipy and scikit-learn modules. We can install them using cmd command:

```
pip install numpy scipy scikit-learn
```

A better option would be downloading miniconda or anaconda packages for python, which come prebundled with these packages. Follow the instructions given here to use anaconda.

Machine Learning overview

Machine learning involves computer to get trained using a given data set, and use this training to predict the properties of a given new data. For example, we can train computer by feeding it 1000 images of cats and 1000 more images which are not of a cat, and tell each time to computer whether a picture is cat or not. Then if we show the computer a new image, then from the above training, computer should be able to tell whether this new image is cat or not.
Process of training and prediction involves use of specialized algorithms. We feed the training data to an algorithm, and the algorithm uses this training data to give predictions on a new test data. One such algorithm is K-Nearest-Neighbor classification (KNN classification). It takes a test data, and finds k nearest data values to this data from test data set. Then it selects the neighbor of maximum frequency and gives its properties as the prediction result. For example if the training set is:

petal_size	flower_type
1	a
2	b
1	a
2	b
3	c
4	d
3	c
2	b
5	a

Now we want to predict flower type for petal of size 2.5 cm. So if we decide no. of neighbors (K)=3, we see that the 3 nearest neighbors of 2.5 are 1, 2 and 3. Their frequencies are 2, 3 and 2 respectively. Therefore the neighbor of maximum frequency is 2 and flower type corresponding to it is b. So for a petal of size 2.5, the prediction will be flower type b.

Implementing KNN- classification algorithm using Python on IRIS dataset

Here is a python script which demonstrates knn classification algorithm. Here we use the famous iris flower dataset to train the computer, and then give a new value to the computer to make predictions about it. The data set consists of 50 samples from each of three species of Iris (Iris setosa, Iris virginica and Iris versicolor). Four features are measured from each sample: The length and Width of Sepals & Petals, in centimeters.
We train our program using this dataset, and then use this training to predict species of a iris flower with given measurements.

Note that this program might not run on Geeksforgeeks IDE, but it can run easily on your local python interpreter, provided, you have installed the required libraries.

```
# Python program to demonstrate
# KNN classification algorithm
# on IRIS dataser

from sklearn.datasets import load_iris
from sklearn.neighbors import KNeighborsClassifier
import numpy as np
from sklearn.model_selection import train_test_split

iris_dataset=load_iris()

X_train, X_test, y_train, y_test = train_test_split(iris_dataset["data"],
iris_dataset["target"], random_state=0)

kn = KNeighborsClassifier(n_neighbors=1)
kn.fit(X_train, y_train)

x_new = np.array([[5, 2.9, 1, 0.2]])
prediction = kn.predict(x_new)

print("Predicted target value: {}\n".format(prediction))
print("Predicted feature name: {}\n".format
    (iris_dataset["target_names"][prediction]))
print("Test score: {:.2f}".format(kn.score(X_test, y_test)))
```

Output:

```
Predicted target name: [0]
Predicted feature name: ['setosa']
Test score: 0.97
```

Explanation of the program:

Training the Dataset

- The first line imports iris data set which is already predefined in sklearn module. Iris data set is basically a table which contains information about various varieties of iris flowers.
- We import kNeighborsClassifier algorithm and train_test_split class from sklearn and numpy module for use in this program.
- Then we encapsulate load_iris() method in iris_dataset variable. Further we divide the dataset into training data and test data using train_test_split method. The X prefix in variable denotes the feature values (eg. petal length etc) and y prefix denotes target values (eg. 0 for setosa, 1 for virginica and 2 for versicolor).
- This method divides dataset into training and test data randomly in ratio of 75:25. Then we encapsulate KNeighborsClassifier method in kn variable while keeping value of k=1. This method contains K Nearest Neighbor algorithm in it.

- In the next line, we fit our training data into this algorithm so that computer can get trained using this data. Now the training part is complete.

Testing the Dataset

- Now we have dimensions of a new flower in a numpy array called x_new and we want to predict the species of this flower. We do this using the predict method which takes this array as input and spits out predicted target value as output.
- So the predicted target value comes out to be 0 which stands for setosa. So this flower has good chances to be of setosa species.
- Finally we find the test score which is the ratio of no. of predictions found correct and total predictions made. We do this using the score method which basically compares the actual values of the test set with the predicted values.

Thus, we saw how machine learning works and developed a basic program to implement it using scikit-learn module in python.

Learning Model Building in Scikit-learn : A Python Machine Learning Library

Pre-requisite: Getting started with machine learning
scikit-learn is an open source Python library that implements a range of machine learning, pre-processing, cross-validation and visualization algorithms using a unified interface.

Important features of scikit-learn:

- Simple and efficient tools for data mining and data analysis. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means, etc.
- Accessible to everybody and reusable in various contexts.
- Built on the top of NumPy, SciPy, and matplotlib.
- Open source, commercially usable – BSD license.

In this article, we are going to see how we can easily build a machine learning model using scikit-learn.

Installation:

Scikit-learn requires:

- NumPy
- SciPy as its dependencies.

Before installing scikit-learn, ensure that you have NumPy and SciPy installed. Once you have a working installation of NumPy and SciPy, the easiest way to install scikit-learn is using pip:

```
pip install -U scikit-learn
```

Let us get started with the modelling process now.

Step 1: Load a dataset

A dataset is nothing but a collection of data. A dataset generally has two main components:

- **Features**: (also known as predictors, inputs, or attributes) they are simply the variables of our data. They can be more than one and hence represented by a **feature matrix** ('X' is a common notation to represent feature matrix). A list of all the feature names is termed as **feature names**.
- **Response**: (also known as the target, label, or output) This is the output variable depending on the feature variables. We generally have a single response column and it is represented by a **response vector** ('y' is a common notation to represent response vector). All the possible values taken by a response vector is termed as **target names**.

Loading exemplar dataset: scikit-learn comes loaded with a few example datasets like the iris and digits datasets for classification and the boston house prices dataset for regression.
Given below is an example of how one can load an exemplar dataset:

```
# load the iris dataset as an example
from sklearn.datasets import load_iris
iris = load_iris()

# store the feature matrix (X) and response vector (y)
X = iris.data
y = iris.target

# store the feature and target names
feature_names = iris.feature_names
target_names = iris.target_names

# printing features and target names of our dataset
print("Feature names:", feature_names)
print("Target names:", target_names)

# X and y are numpy arrays
print("\nType of X is:", type(X))

# printing first 5 input rows
print("\nFirst 5 rows of X:\n", X[:5])
```

Output:

```
Feature names: ['sepal length (cm)','sepal width (cm)',
                'petal length (cm)','petal width (cm)']
Target names: ['setosa' 'versicolor' 'virginica']

Type of X is:

First 5 rows of X:
 [[ 5.1  3.5  1.4  0.2]
 [ 4.9  3.   1.4  0.2]
 [ 4.7  3.2  1.3  0.2]
 [ 4.6  3.1  1.5  0.2]
 [ 5.   3.6  1.4  0.2]]
```

Loading external dataset: Now, consider the case when we want to load an external dataset. For this purpose, we can use **pandas library** for easily loading and manipulating dataset.

To install pandas, use the following pip command:

```
pip install pandas
```

In pandas, important data types are:

Series: Series is a one-dimensional labeled array capable of holding any data type.

DataFrame: It is a 2-dimensional labeled data structure with columns of potentially different types. You can think of it like a spreadsheet or SQL table, or a dict of Series objects. It is generally the most commonly used pandas object.

Note: The CSV file used in example below can be downloaded from here: weather.csv

```
import pandas as pd

# reading csv file
data = pd.read_csv('weather.csv')

# shape of dataset
print("Shape:", data.shape)

# column names
print("\nFeatures:", data.columns)

# storing the feature matrix (X) and response vector (y)
X = data[data.columns[:-1]]
y = data[data.columns[-1]]

# printing first 5 rows of feature matrix
print("\nFeature matrix:\n", X.head())

# printing first 5 values of response vector
print("\nResponse vector:\n", y.head())
```

Output:

```
Shape: (14, 5)

Features: Index([u'Outlook', u'Temperature', u'Humidity',
                u'Windy', u'Play'], dtype='object')

Feature matrix:
     Outlook Temperature Humidity  Windy
0   overcast         hot     high  False
1   overcast        cool   normal   True
2   overcast        mild     high   True
3   overcast         hot   normal  False
4      rainy        mild     high  False
```

```
Response vector:
0    yes
1    yes
2    yes
3    yes
4    yes
Name: Play, dtype: object
```

Step 2: Splitting the dataset

One important aspect of all machine learning models is to determine their accuracy. Now, in order to determine their accuracy, one can train the model using the given dataset and then predict the response values for same dataset using that model and hence, find the accuracy of model.
But this method has several flaws in it, like:

- Goal is to estimate likely performance of a model on an **out-of-sample** data.
- Maximizing training accuracy rewards overly complex models that won't necessarily generalize our model.
- Unnecessarily complex models may over-fit the training data.

A better option is to split our data into two parts: first one for training our machine learning model, and second one for testing our model.
To summarize:

- Split the dataset into two pieces: a training set and a testing set.
- Train the model on the training set.
- Test the model on the testing set, and evaluate how well our model did.

Advantages of train/test split:

- Model can be trained and tested on different data than the one used for training.
- Response values are known for the test dataset, hence predictions can be evaluated
- Testing accuracy is a better estimate than training accuracy of out-of-sample performance.

Consider the example below:

```
# load the iris dataset as an example
from sklearn.datasets import load_iris
iris = load_iris()


# store the feature matrix (X) and response vector (y)
X = iris.data
y = iris.target


# splitting X and y into training and testing sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4,
random_state=1)

# printing the shapes of the new X objects
print(X_train.shape)
print(X_test.shape)


# printing the shapes of the new y objects
```

```
print(y_train.shape)
print(y_test.shape)
```

Output:

```
(90L, 4L)
(60L, 4L)
(90L,)
(60L,)
```

The **train_test_split** function takes several arguments which are explained below:

- **X, y**: These are the feature matrix and response vector which need to be splitted.
- **test_size**: It is the ratio of test data to the given data. For example, setting test_size = 0.4 for 150 rows of X produces test data of 150 x 0.4 = 60 rows.
- **random_state**: If you use random_state = some_number, then you can guarantee that your split will be always the same. This is useful if you want reproducible results, for example in testing for consistency in the documentation (so that everybody can see the same numbers).

Step 3: Training the model

Now, its time to train some prediction-model using our dataset. Scikit-learn provides a wide range of machine learning algorithms which have a unified/consistent interface for fitting, predicting accuracy, etc.

The example given below uses KNN (K nearest neighbors) classifier.

Note: We will not go into the details of how the algorithm works as we are interested in understanding its implementation only.

Now, consider the example below:

```
# load the iris dataset as an example
from sklearn.datasets import load_iris
iris = load_iris()

# store the feature matrix (X) and response vector (y)
X = iris.data
y = iris.target

# splitting X and y into training and testing sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4,
random_state=1)

# training the model on training set
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)

# making predictions on the testing set
y_pred = knn.predict(X_test)
```

```
# comparing actual response values (y_test) with predicted response values
(y_pred)
from sklearn import metrics
print("kNN model accuracy:", metrics.accuracy_score(y_test, y_pred))

# making prediction for out of sample data
sample = [[3, 5, 4, 2], [2, 3, 5, 4]]
preds = knn.predict(sample)
pred_species = [iris.target_names[p] for p in preds]
print("Predictions:", pred_species)

# saving the model
from sklearn.externals import joblib
joblib.dump(knn, 'iris_knn.pkl')
```

Output:

```
kNN model accuracy: 0.983333333333
Predictions: ['versicolor', 'virginica']
```

Important points to note from above code:

- We create a knn classifier object using:
- `knn = KNeighborsClassifier(n_neighbors=3)`
- The classifier is trained using X_train data. The process is termed as **fitting**. We pass the feature matrix and the corresponding response vector.
- `knn.fit(X_train, y_train)`
- Now, we need to test our classifier on the X_test data. **knn.predict** method is used for this purpose. It returns the predicted response vector, **y_pred**.
- `y_pred = knn.predict(X_test)`
- Now, we are interested in finding the accuracy of our model by comparing **y_test** and **y_pred**. This is done using metrics module's method **accuracy_score**:
- `print(metrics.accuracy_score(y_test, y_pred))`
- Consider the case when you want your model to make prediction on **out of sample** data. Then, the sample input can simply pe passed in the same way as we pass any feature matrix.
- `sample = [[3, 5, 4, 2], [2, 3, 5, 4]]`
- `preds = knn.predict(sample)`
- If you are not interested in training your classifier again and again and use the pre-trained classifier, one can save their classifier using **joblib**. All you need to do is:
- `joblib.dump(knn, 'iris_knn.pkl')`
- In case you want to load an already saved classifier, use the following method:

  ```
  knn = joblib.load('iris_knn.pkl')
  ```

As we approach to end of this article, here are some benefits of using scikit-learn over some other machine learning libraries(like R):

- **Consistent interface** to machine learning models
- Provides many **tuning parameters** but with **sensible defaults**
- Exceptional **documentation**
- Rich set of functionality for **companion tasks**.
- **Active community** for development and support.

Multiclass classification using scikit-learn

Multiclass classification is a popular problem in supervised machine learning.

Problem – Given a dataset of **m** training examples, each of which contains information in the form of various features and a label. Each label corresponds to a class, to which the training example belongs to. In multiclass classification, we have a finite set of classes. Each training example also has **n** features.

For example, in the case of identification of different types of fruits, “Shape”, “Color”, “Radius” can be features and “Apple”, “Orange”, “Banana” can be different class labels.

In a multiclass classification, we train a classifier using our training data, and use this classifier for classifying new examples.

Aim of this article – We will use different multiclass classification methods such as, KNN, Decision trees, SVM, etc. We will compare their accuracy on test data. We will perform all this with sci-kit learn (Python). For information on how to install and use sci-kit learn, visit http://scikit-learn.org/stable/

Approach –

1. Load dataset from source.
2. Split the dataset into “training” and “test” data.
3. Train Decision tree, SVM, and KNN classifiers on the training data.
4. Use the above classifiers to predict labels for the test data.
5. Measure accuracy and visualise classification.

Decision tree classifier – Decision tree classifier is a systematic approach for multiclass classification. It poses a set of questions to the dataset (related to its attributes/features). The decision tree classification algorithm can be visualized on a binary tree. On the root and each of the internal nodes, a question is posed and the data on that node is further split into separate records that have different characteristics. The leaves of the tree refer to the classes in which the dataset is split. In the following code snippet, we train a decision tree classifier in scikit-learn.

```
# importing necessary libraries
from sklearn import datasets
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split

# loading the iris dataset
iris = datasets.load_iris()

# X -> features, y -> label
X = iris.data
y = iris.target

# dividing X, y into train and test data
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state =
0)
```

```
# training a DescisionTreeClassifier
from sklearn.tree import DecisionTreeClassifier
dtree_model = DecisionTreeClassifier(max_depth = 2).fit(X_train, y_train)
dtree_predictions = dtree_model.predict(X_test)

# creating a confusion matrix
cm = confusion_matrix(y_test, dtree_predictions)
```

SVM (Support vector machine) classifier –

SVM (Support vector machine) is an efficient classification method when the feature vector is high dimensional. In sci-kit learn, we can specify the the kernel function (here, linear). To know more about kernel functions and SVM refer – Kernel function | sci-kit learn and SVM.

```
# importing necessary libraries
from sklearn import datasets
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split

# loading the iris dataset
iris = datasets.load_iris()

# X -> features, y -> label
X = iris.data
y = iris.target

# dividing X, y into train and test data
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state =
0)

# training a linear SVM classifier
from sklearn.svm import SVC
svm_model_linear = SVC(kernel = 'linear', C = 1).fit(X_train, y_train)
svm_predictions = svm_model_linear.predict(X_test)

# model accuracy for X_test
accuracy = svm_model_linear.score(X_test, y_test)

# creating a confusion matrix
cm = confusion_matrix(y_test, svm_predictions)
```

KNN (k-nearest neighbours) classifier – KNN or k-nearest neighbours is the simplest classification algorithm. This classification algorithm does not depend on the structure of the data. Whenever a new example is encountered, its k nearest neighbours from the training data are examined. Distance between two examples can be the euclidean distance between their feature vectors. The majority class among the k nearest neighbours is taken to be the class for the encountered example.

```
# importing necessary libraries
from sklearn import datasets
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
```

```
# loading the iris dataset
iris = datasets.load_iris()

# X -> features, y -> label
X = iris.data
y = iris.target

# dividing X, y into train and test data
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state =
0)

# training a KNN classifier
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors = 7).fit(X_train, y_train)

# accuracy on X_test
accuracy = knn.score(X_test, y_test)
print accuracy

# creating a confusion matrix
knn_predictions = knn.predict(X_test)
cm = confusion_matrix(y_test, knn_predictions)
```

Naive Bayes classifier – Naive Bayes classification method is based on Bayes' theorem. It is termed as 'Naive' because it assumes independence between every pair of feature in the data. Let $(x_1, x_2, \ldots, x_n)$ be a feature vector and **y** be the class label corresponding to this feature vector.

Applying Bayes' theorem,

$$P(y|x_1, ..., x_n) = \frac{P(y)P(x_1, ..., x_n|y)}{P(x_1, ..., x_n)}$$

Since, **x_1, x_2, ..., x_n** are independent of each other,

$$P(y|x_1, ..., x_n) = \frac{P(y)\prod_{i=1}^{n}(P(x_i|y)}{P(x_1, ..., x_n))}$$

Inserting proportionality by removing the **P(x_1, ..., x_n)** (since, it is constant).

$$P(y|x_1, ..., x_n)\alpha(P(y)\prod_{i=1}^{n}(P(x_i|y))$$

Therefore, the class label is decided by,

$$\hat{y} = arg\ \underset{y}{max}P(y)\prod_{i=1}^{n}(P(x_i|y)$$

P(y) is the relative frequency of class label **y** in the training dataset.

In case of Gaussian Naive Bayes classifier, **P(x_i | y)** is calculated as,

$$P(x_i|y) = \frac{1}{\sqrt{2\pi\sigma_y^2}}\ exp\left(-\frac{(x_i-\mu_y)^2}{2\sigma_y^2}\right)$$

```
# importing necessary libraries
from sklearn import datasets
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split

# loading the iris dataset
iris = datasets.load_iris()

# X -> features, y -> label
X = iris.data
y = iris.target

# dividing X, y into train and test data
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state =
0)

# training a Naive Bayes classifier
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB().fit(X_train, y_train)
gnb_predictions = gnb.predict(X_test)

# accuracy on X_test
accuracy = gnb.score(X_test, y_test)
print accuracy

# creating a confusion matrix
cm = confusion_matrix(y_test, gnb_predictions)
```

Classifying data using Support Vector Machines(SVMs) in Python

Introduction to SVMs:
In machine learning, support vector machines (SVMs, also support vector networks) are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis.
A Support Vector Machine (SVM) is a discriminative classifier formally defined by a separating hyperplane.
In other words, given labeled training data (supervised learning), the algorithm outputs an optimal hyperplane which categorizes new examples.

What is Support Vector Machine?

An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible.
In addition to performing linear classification, SVMs can efficiently perform a non-linear classification, implicitly mapping their inputs into high-dimensional feature spaces.

What does SVM do?

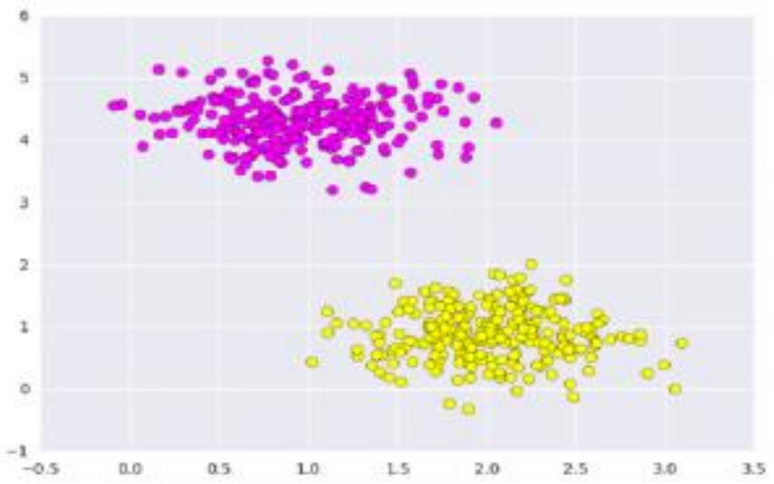
Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier.

Let you have basic understandings from this article before you proceed further. Here I'll discuss an example about SVM classification of cancer UCI datasets using machine learning tools i.e. scikit-learn compatible with Python.
Pre-requisites: Numpy, Pandas, matplot-lib, scikit-learn
Let's have a quick example of support vector classification. First we need to create a dataset:

```
# importing scikit learn with make_blobs
from sklearn.datasets.samples_generator import make_blobs
```

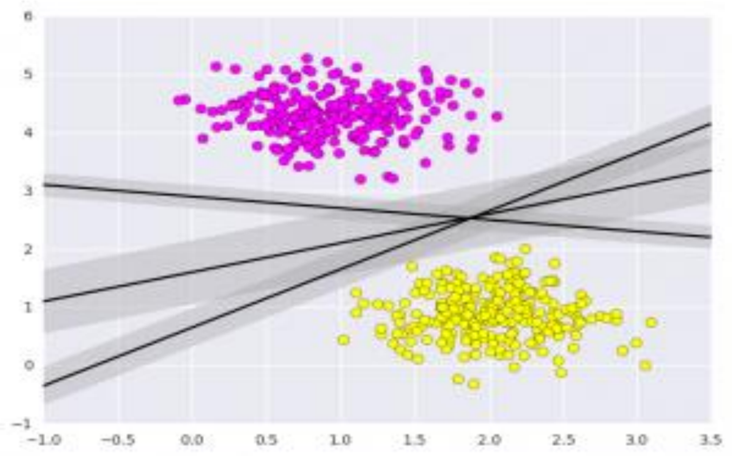
```
# creating datasets X containing n_samples
# Y containing two classes
X, Y = make_blobs(n_samples=500, centers=2,
                  random_state=0, cluster_std=0.40)

# plotting scatters
plt.scatter(X[:, 0], X[:, 1], c=Y, s=50, cmap='spring');
plt.show()
```

Output:



What Support vector machines do, is to not only draw a line between two classes here, but consider a region about the line of some given width. Here's an example of what it can look like:

```
# creating line space between -1 to 3.5
xfit = np.linspace(-1, 3.5)

# plotting scatter
plt.scatter(X[:, 0], X[:, 1], c=Y, s=50, cmap='spring')

# plot a line between the different sets of data
for m, b, d in [(1, 0.65, 0.33), (0.5, 1.6, 0.55), (-0.2, 2.9, 0.2)]:
    yfit = m * xfit + b
    plt.plot(xfit, yfit, '-k')
    plt.fill_between(xfit, yfit - d, yfit + d, edgecolor='none',
    color='#AAAAAA', alpha=0.4)

plt.xlim(-1, 3.5);
plt.show()
```



Importing datasets

This is the intuition of support vector machines, which optimize a linear discriminant model representing the perpendicular distance between the datasets. Now let's train the classifier using our training data. Before training, we need to import cancer datasets as csv file where we will train two features out of all features.

```
# importing required libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# reading csv file and extracting class column to y.
x = pd.read_csv("C:\...\cancer.csv")
a = np.array(x)
y  = a[:,30] # classes having 0 and 1

# extracting two features
x = np.column_stack((x.malignant,x.benign))
x.shape # 569 samples and 2 features
```

```
print (x),(y)
[[  122.8   1001.  ]
 [  132.9   1326.  ]
 [  130.    1203.  ]
 ...,
 [  108.3    858.1 ]
 [  140.1   1265.  ]
 [   47.92   181.  ]]

array([ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
         0.,  0.,  0.,  0.,  0.,  0.,  1.,  1.,  1.,  0.,  0.,  0.,  0.,
         0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  1.,  0.,
         0.,  0.,  0.,  0.,  0.,  0.,  0.,  1.,  0.,  1.,  1.,  1.,  1.,
         1.,  0.,  0.,  1.,  0.,  0.,  1.,  1.,  1.,  1.,  0.,  1., ....,
         1.])
```

Fitting a Support Vector Machine

Now we'll fit a Support Vector Machine Classifier to these points. While the mathematical details of the likelihood model are interesting, we'll let read about those elsewhere. Instead, we'll just treat the scikit-learn algorithm as a black box which accomplishes the above task.

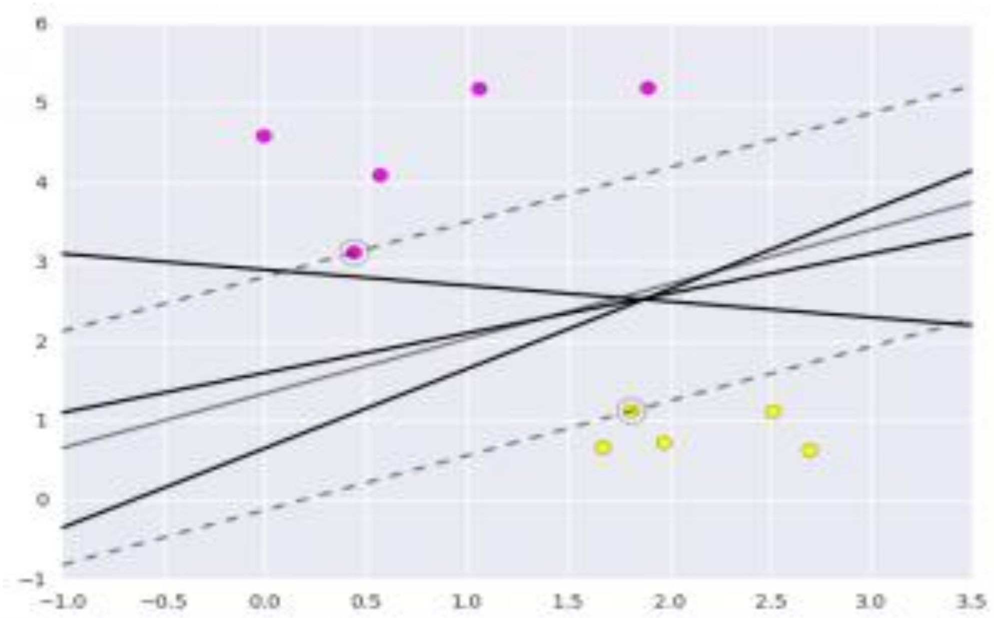
```
# import support vector classifier
from sklearn.svm import SVC # "Support Vector Classifier"
clf = SVC(kernel='linear')

# fitting x samples and y classes
clf.fit(x, y)
```

After being fitted, the model can then be used to predict new values:

```
clf.predict([[120, 990]])

clf.predict([[85, 550]])
```

```
array([ 0.])
array([ 1.])
```

Let’s have a look on the graph how does this show.



This is obtained by analyzing the data taken and pre-processing methods to make optimal hyperplanes using matplotlib function.

Classifying data using Support Vector Machines(SVMs) in R

In machine learning, Support vector machine(SVM) are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. It is mostly used in classification problems. In this algorithm, each data item is plotted as a point in n-dimensional space (where n is number of features), with the value of each feature being the value of a particular coordinate. Then, classification is performed by finding the hyper-plane that best differentiates the two classes.

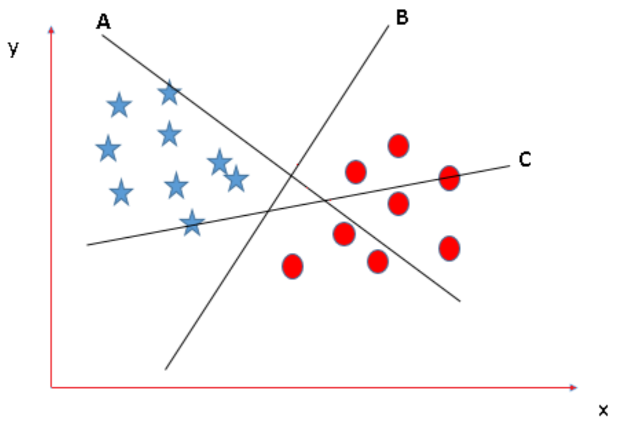
In addition to performing linear classification, SVMs can efficiently perform a non-linear classification, implicitly mapping their inputs into high-dimensional feature spaces.

How SVM works

A Support Vector Machine (SVM) is a discriminative classifier formally defined by a separating hyperplane. In other words, given labeled training data (supervised learning), the algorithm outputs an optimal hyperplane which categorizes new examples.

The most important question that arise while using SVM is how to decide right hyper plane. Consider the following scenarios:

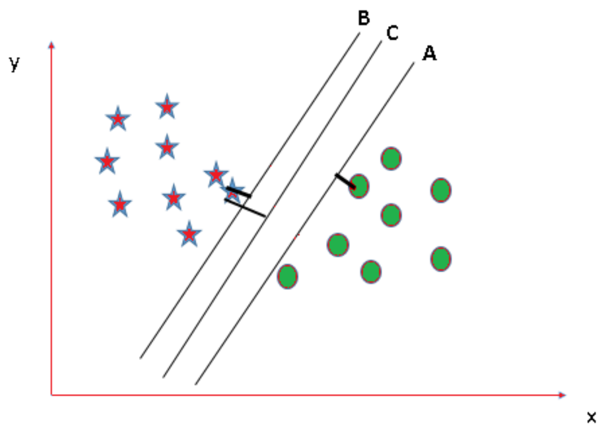
- **Scenario 1:**
  In this scenario there are three hyper planes called A,B,C. Now the problem is to identify the right hyper-plane which best differentiates the stars and the circles.



  The thumb rule to be known, before finding the right hyper plane, to classify star and circle is that the hyper plane should be selected which segregate two classes better.

  In this case B classify star and circle better, hence it is right hyper plane.

- **Scenario 2:**
  Now take another Scenario where all three planes are segregating classes well. Now the question arises how to identify the right plane in this situation.



In such scenarios, calculate the margin which is the distance between nearest data point and hyper-plane. The plane having the maximum distance will be considered as the right hyper plane to classify the classes better.

Here C is having the maximum margin and hence it will be considered as right hyper plane.

Above are some scenario to identify the right hyper-plane.

Note: For details on Classifying using SVM in Python, refer Classifying data using Support Vector Machines(SVMs) in Python

Implementation of SVM in R

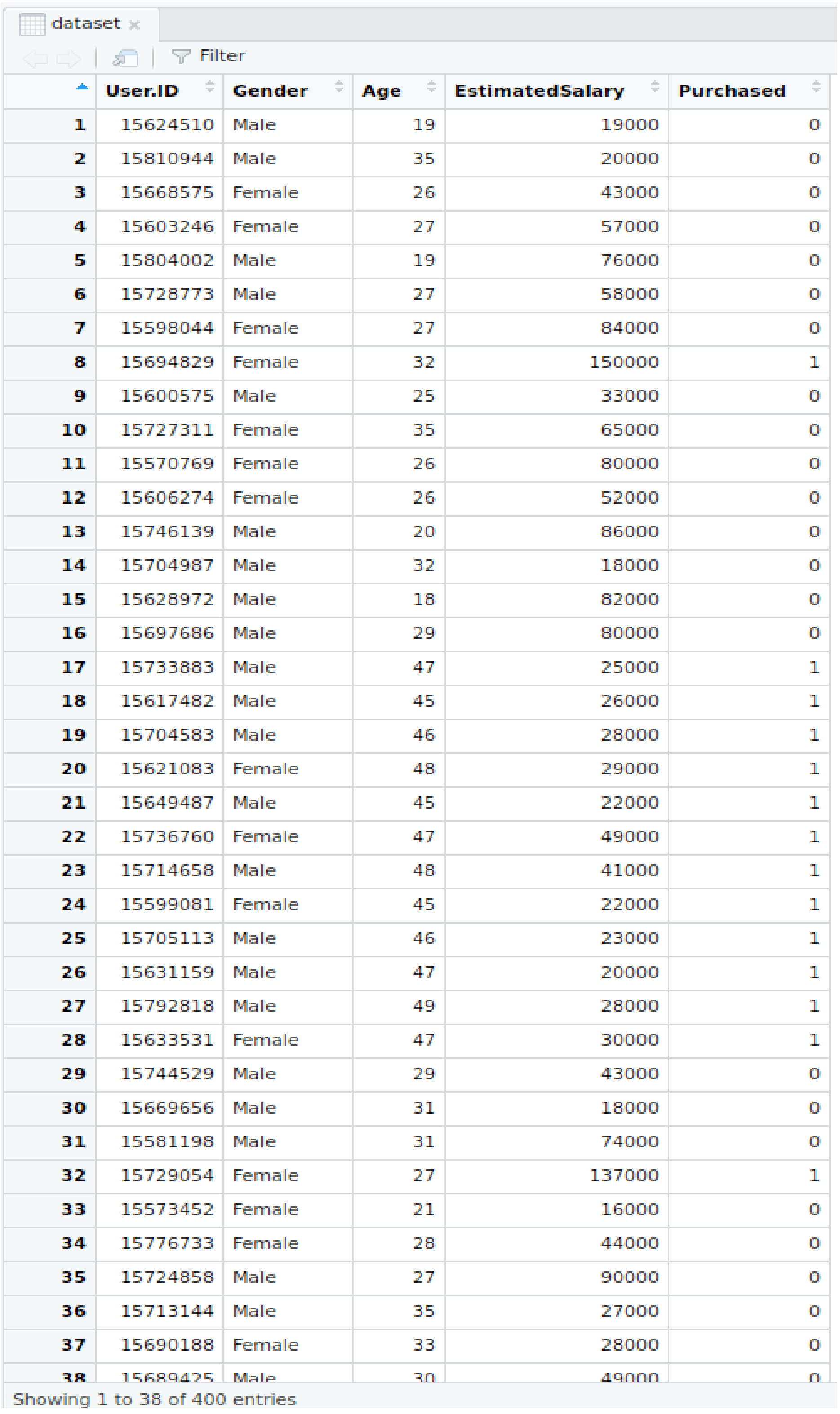
Here, an example is taken by importing a dataset of Social network aids from file Social.csv

The implementation is explained in the following steps:

- **Importing the dataset**

```
# Importing the dataset
dataset = read.csv('Social_Network_Ads.csv')
dataset = dataset[3:5]
```

- **Output:**



	User.ID	Gender	Age	EstimatedSalary	Purchased
1	15624510	Male	19	19000	0
2	15810944	Male	35	20000	0
3	15668575	Female	26	43000	0
4	15603246	Female	27	57000	0
5	15804002	Male	19	76000	0
6	15728773	Male	27	58000	0
7	15598044	Female	27	84000	0
8	15694829	Female	32	150000	1
9	15600575	Male	25	33000	0
10	15727311	Female	35	65000	0
11	15570769	Female	26	80000	0
12	15606274	Female	26	52000	0
13	15746139	Male	20	86000	0
14	15704987	Male	32	18000	0
15	15628972	Male	18	82000	0
16	15697686	Male	29	80000	0
17	15733883	Male	47	25000	1
18	15617482	Male	45	26000	1
19	15704583	Male	46	28000	1
20	15621083	Female	48	29000	1
21	15649487	Male	45	22000	1
22	15736760	Female	47	49000	1
23	15714658	Male	48	41000	1
24	15599081	Female	45	22000	1
25	15705113	Male	46	23000	1
26	15631159	Male	47	20000	1
27	15792818	Male	49	28000	1
28	15633531	Female	47	30000	1
29	15744529	Male	29	43000	0
30	15669656	Male	31	18000	0
31	15581198	Male	31	74000	0
32	15729054	Female	27	137000	1
33	15573452	Female	21	16000	0
34	15776733	Female	28	44000	0
35	15724858	Male	27	90000	0
36	15713144	Male	35	27000	0
37	15690188	Female	33	28000	0
38	15689425	Male	30	49000	0

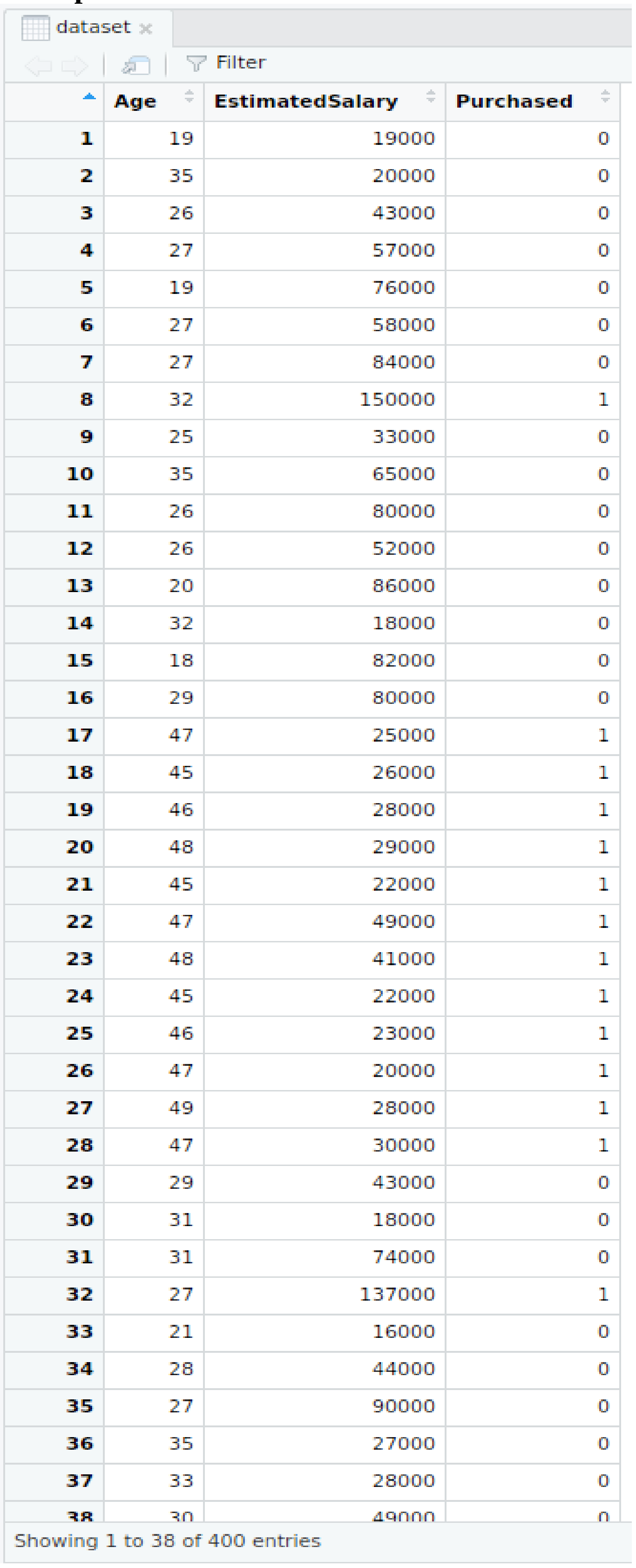
Showing 1 to 38 of 400 entries

- **Selecting columns 3-5**

This is done for the ease of computation and implementation (to keep the example simple).

```
# Taking columns 3-5
dataset = dataset[3:5]
```

- **Output:**



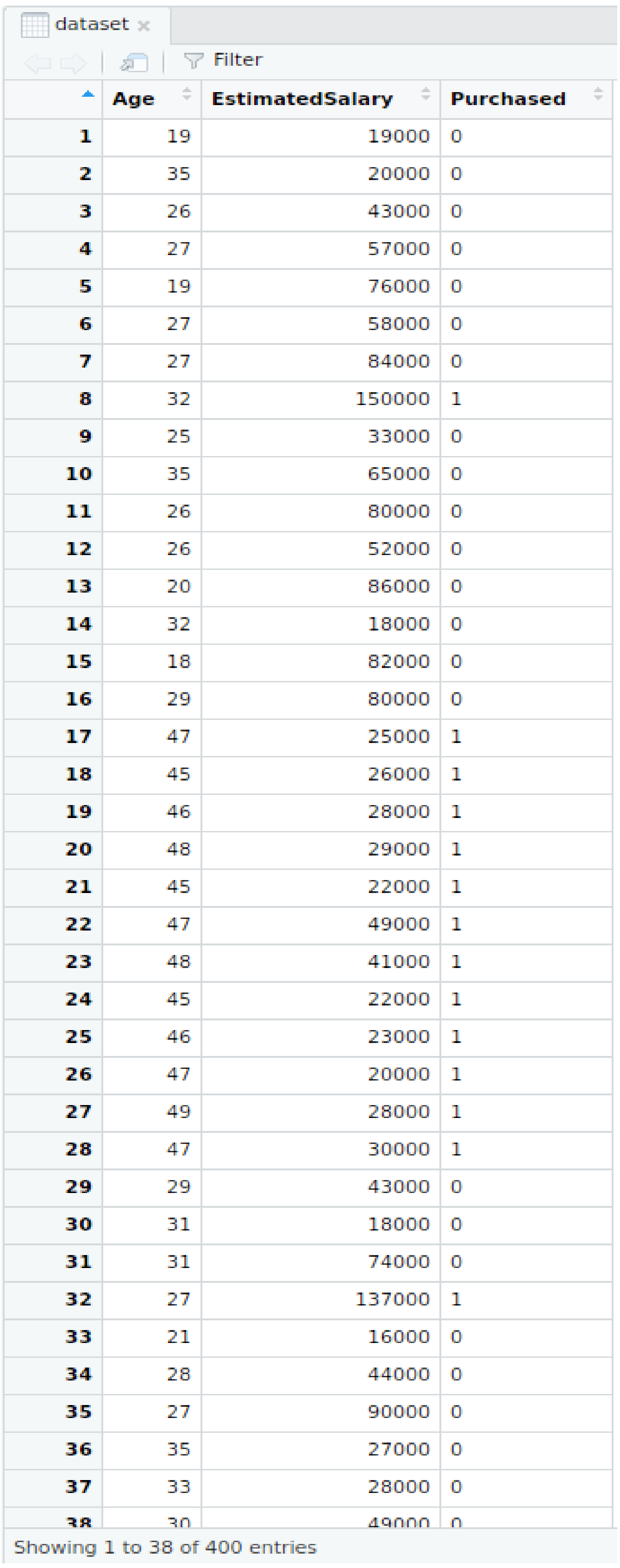
	Age	EstimatedSalary	Purchased
1	19	19000	0
2	35	20000	0
3	26	43000	0
4	27	57000	0
5	19	76000	0
6	27	58000	0
7	27	84000	0
8	32	150000	1
9	25	33000	0
10	35	65000	0
11	26	80000	0
12	26	52000	0
13	20	86000	0
14	32	18000	0
15	18	82000	0
16	29	80000	0
17	47	25000	1
18	45	26000	1
19	46	28000	1
20	48	29000	1
21	45	22000	1
22	47	49000	1
23	48	41000	1
24	45	22000	1
25	46	23000	1
26	47	20000	1
27	49	28000	1
28	47	30000	1
29	29	43000	0
30	31	18000	0
31	31	74000	0
32	27	137000	1
33	21	16000	0
34	28	44000	0
35	27	90000	0
36	35	27000	0
37	33	28000	0
38	30	49000	0

Showing 1 to 38 of 400 entries

- **Encoding the target feature**

```
# Encoding the target feature as factor
dataset$Purchased = factor(dataset$Purchased, levels = c(0, 1))
```

- **Output:**



	Age	EstimatedSalary	Purchased
1	19	19000	0
2	35	20000	0
3	26	43000	0
4	27	57000	0
5	19	76000	0
6	27	58000	0
7	27	84000	0
8	32	150000	1
9	25	33000	0
10	35	65000	0
11	26	80000	0
12	26	52000	0
13	20	86000	0
14	32	18000	0
15	18	82000	0
16	29	80000	0
17	47	25000	1
18	45	26000	1
19	46	28000	1
20	48	29000	1
21	45	22000	1
22	47	49000	1
23	48	41000	1
24	45	22000	1
25	46	23000	1
26	47	20000	1
27	49	28000	1
28	47	30000	1
29	29	43000	0
30	31	18000	0
31	31	74000	0
32	27	137000	1
33	21	16000	0
34	28	44000	0
35	27	90000	0
36	35	27000	0
37	33	28000	0
38	30	49000	0

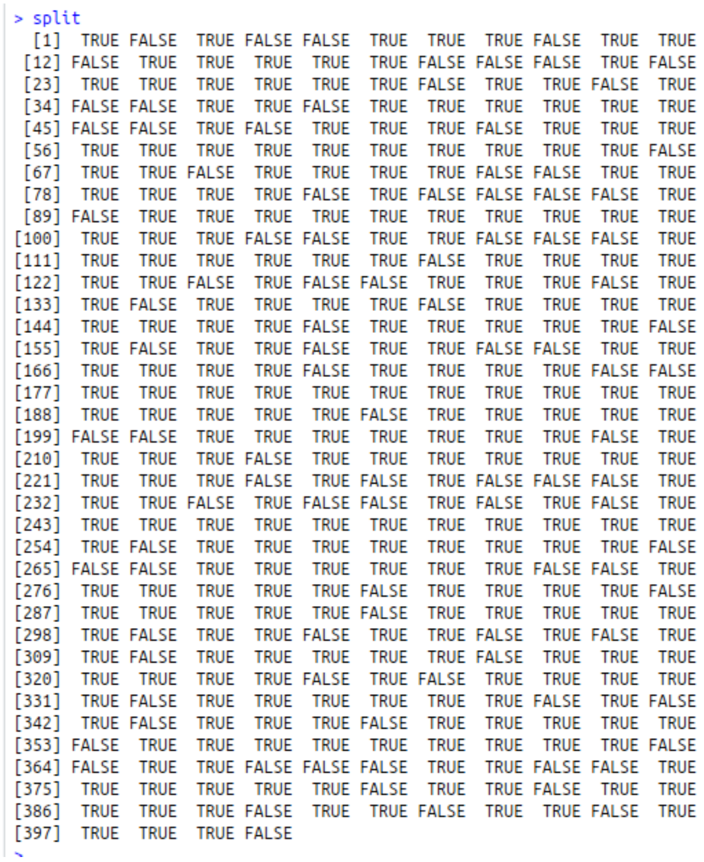
Showing 1 to 38 of 400 entries

- **Splitting the dataset**

```
# Splitting the dataset into the Training set and Test set
install.packages('caTools')
```

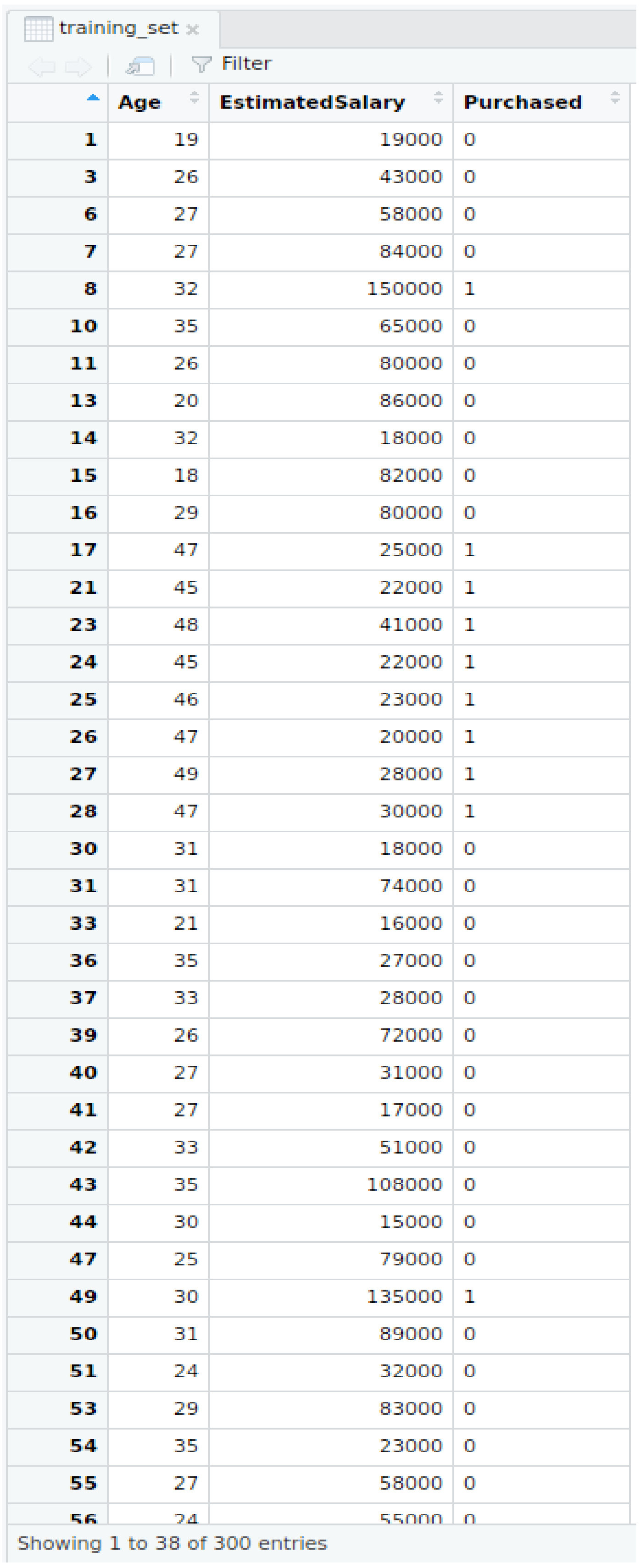
```
library(caTools)

set.seed(123)
split = sample.split(dataset$Purchased, SplitRatio = 0.75)

training_set = subset(dataset, split == TRUE)
test_set = subset(dataset, split == FALSE)
```

- **Output:**
  - Splitter



```
> split
  [1]  TRUE FALSE  TRUE FALSE FALSE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE
 [12] FALSE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE  TRUE FALSE
 [23]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE FALSE  TRUE
 [34] FALSE FALSE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
 [45] FALSE FALSE  TRUE FALSE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE
 [56]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE
 [67]  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE FALSE FALSE  TRUE  TRUE
 [78]  TRUE  TRUE  TRUE  TRUE FALSE  TRUE FALSE FALSE FALSE FALSE  TRUE
 [89] FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
[100]  TRUE  TRUE  TRUE FALSE FALSE  TRUE  TRUE FALSE FALSE FALSE  TRUE
[111]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE
[122]  TRUE  TRUE FALSE  TRUE FALSE FALSE  TRUE  TRUE  TRUE FALSE  TRUE
[133]  TRUE FALSE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE
[144]  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE
[155]  TRUE FALSE  TRUE  TRUE FALSE  TRUE  TRUE FALSE FALSE  TRUE  TRUE
[166]  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE FALSE FALSE
[177]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
[188]  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
[199] FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE
[210]  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
[221]  TRUE  TRUE  TRUE FALSE  TRUE FALSE  TRUE FALSE FALSE FALSE  TRUE
[232]  TRUE  TRUE FALSE  TRUE FALSE FALSE  TRUE FALSE  TRUE FALSE  TRUE
[243]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
[254]  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE
[265] FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE  TRUE
[276]  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE FALSE
[287]  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
[298]  TRUE FALSE  TRUE  TRUE FALSE  TRUE  TRUE FALSE  TRUE FALSE  TRUE
[309]  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE
[320]  TRUE  TRUE  TRUE  TRUE FALSE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE
[331]  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE FALSE
[342]  TRUE FALSE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
[353] FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE
[364] FALSE  TRUE  TRUE FALSE FALSE FALSE  TRUE  TRUE FALSE FALSE  TRUE
[375]  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE FALSE  TRUE  TRUE
[386]  TRUE  TRUE  TRUE FALSE  TRUE  TRUE FALSE  TRUE  TRUE FALSE  TRUE
[397]  TRUE  TRUE  TRUE FALSE
>
```

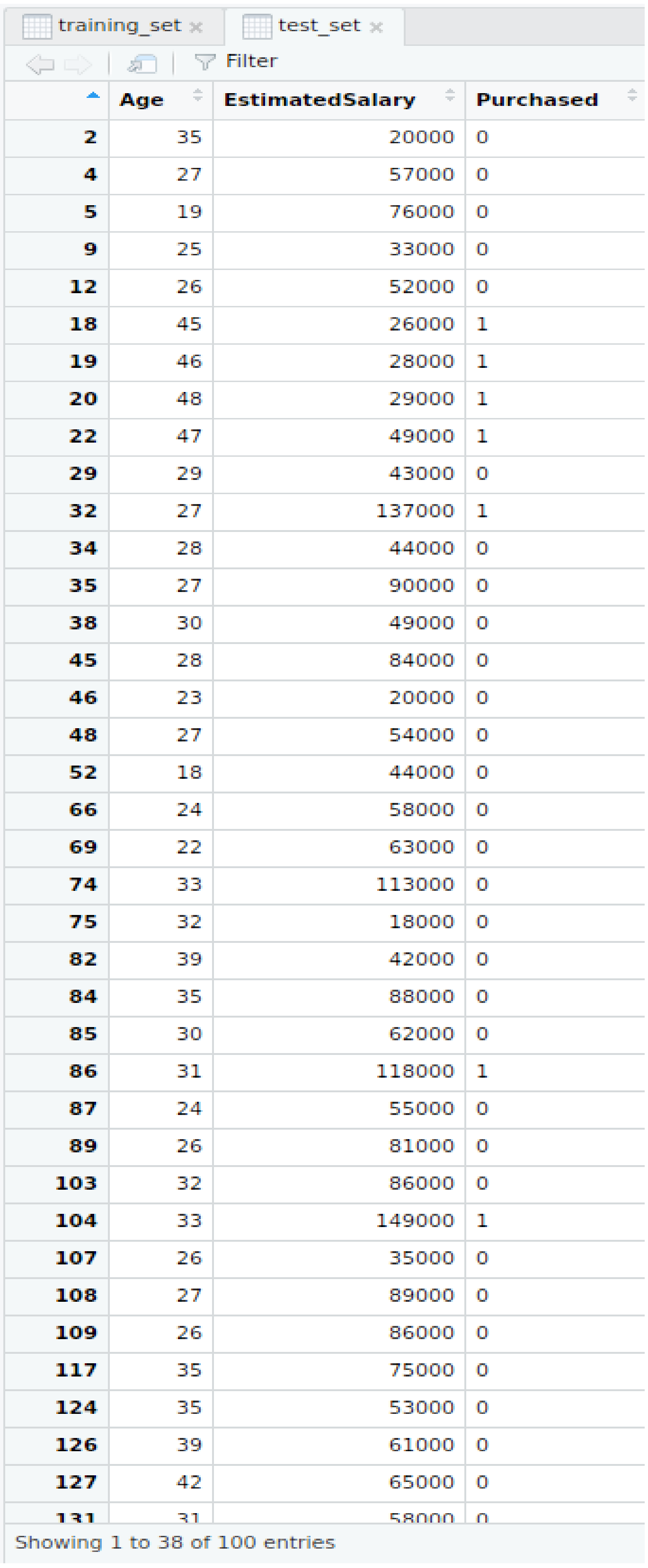
- Training dataset



	Age	EstimatedSalary	Purchased
1	19	19000	0
3	26	43000	0
6	27	58000	0
7	27	84000	0
8	32	150000	1
10	35	65000	0
11	26	80000	0
13	20	86000	0
14	32	18000	0
15	18	82000	0
16	29	80000	0
17	47	25000	1
21	45	22000	1
23	48	41000	1
24	45	22000	1
25	46	23000	1
26	47	20000	1
27	49	28000	1
28	47	30000	1
30	31	18000	0
31	31	74000	0
33	21	16000	0
36	35	27000	0
37	33	28000	0
39	26	72000	0
40	27	31000	0
41	27	17000	0
42	33	51000	0
43	35	108000	0
44	30	15000	0
47	25	79000	0
49	30	135000	1
50	31	89000	0
51	24	32000	0
53	29	83000	0
54	35	23000	0
55	27	58000	0
56	24	55000	0

Showing 1 to 38 of 300 entries

- Test dataset



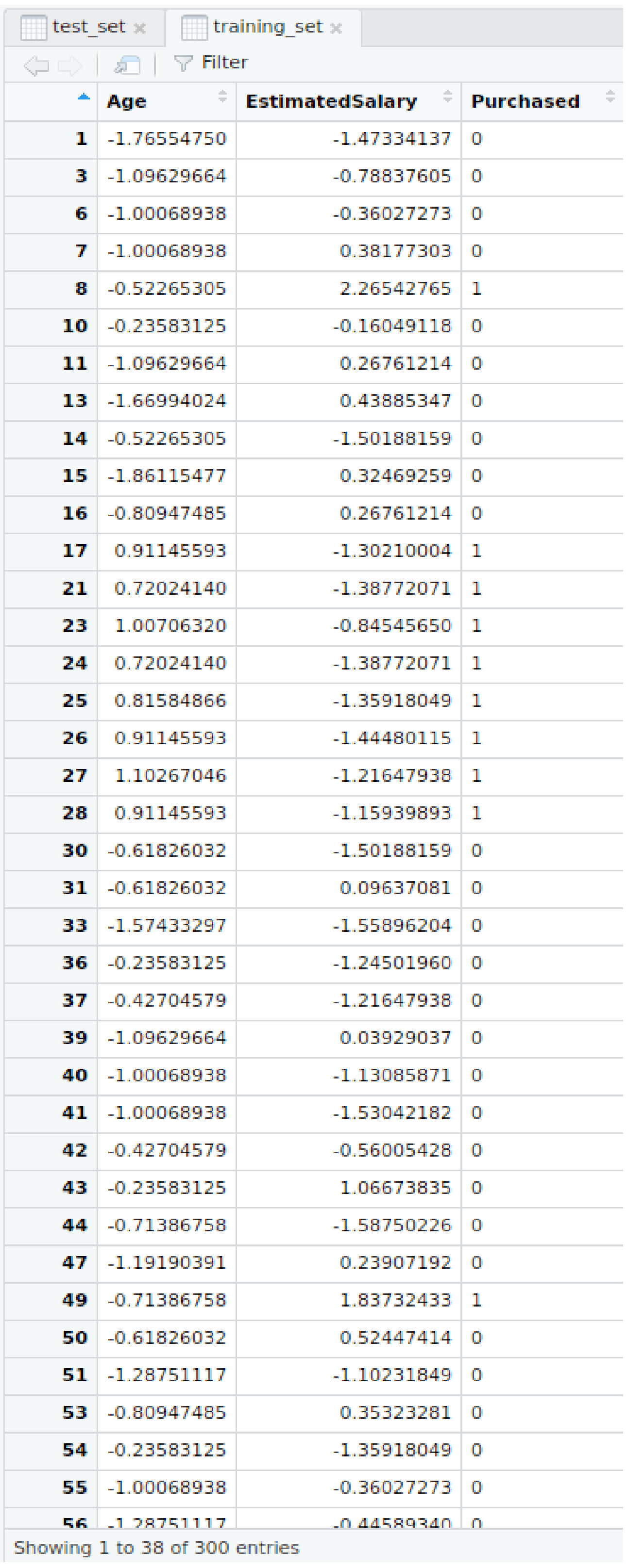
	Age	EstimatedSalary	Purchased
2	35	20000	0
4	27	57000	0
5	19	76000	0
9	25	33000	0
12	26	52000	0
18	45	26000	1
19	46	28000	1
20	48	29000	1
22	47	49000	1
29	29	43000	0
32	27	137000	1
34	28	44000	0
35	27	90000	0
38	30	49000	0
45	28	84000	0
46	23	20000	0
48	27	54000	0
52	18	44000	0
66	24	58000	0
69	22	63000	0
74	33	113000	0
75	32	18000	0
82	39	42000	0
84	35	88000	0
85	30	62000	0
86	31	118000	1
87	24	55000	0
89	26	81000	0
103	32	86000	0
104	33	149000	1
107	26	35000	0
108	27	89000	0
109	26	86000	0
117	35	75000	0
124	35	53000	0
126	39	61000	0
127	42	65000	0
131	31	58000	0

Showing 1 to 38 of 100 entries

- **Feature Scaling**

```
# Feature Scaling
training_set[-3] = scale(training_set[-3])
test_set[-3] = scale(test_set[-3])
```

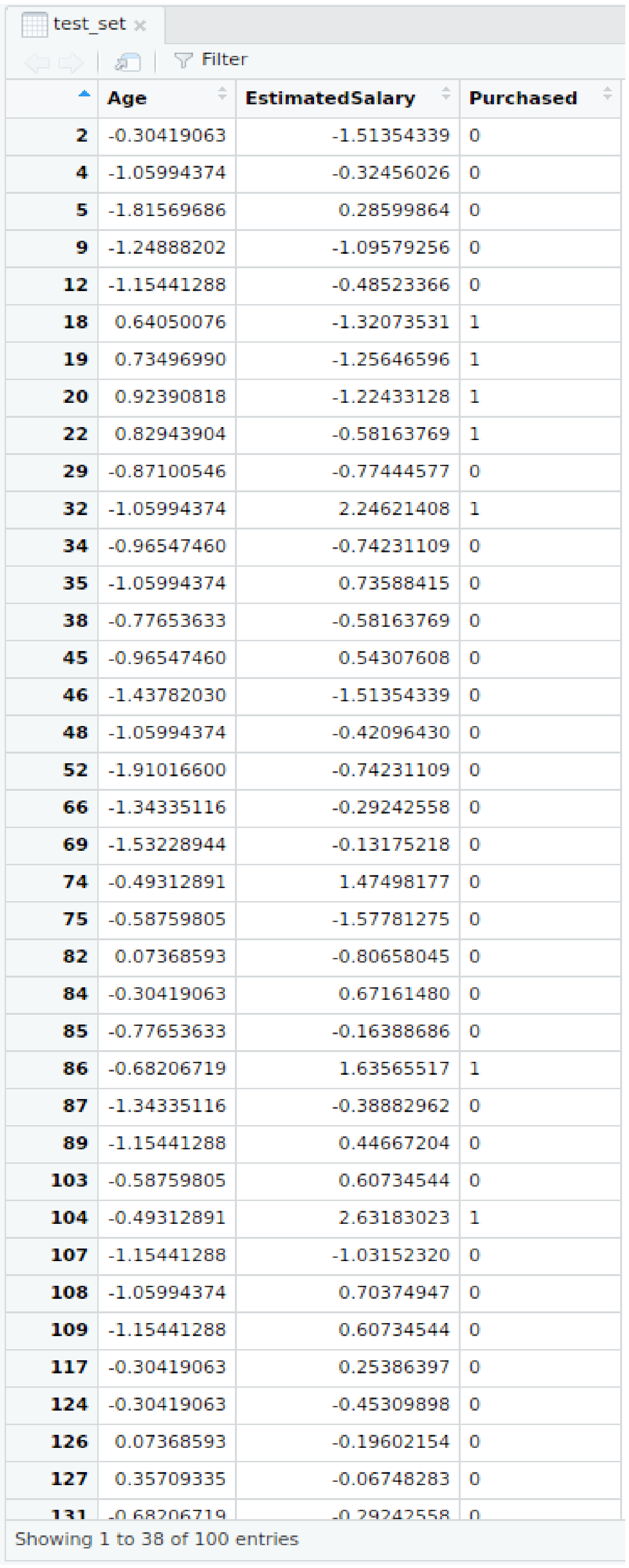
- **Output:**
  - Feature scaled training dataset



	Age	EstimatedSalary	Purchased
1	-1.76554750	-1.47334137	0
3	-1.09629664	-0.78837605	0
6	-1.00068938	-0.36027273	0
7	-1.00068938	0.38177303	0
8	-0.52265305	2.26542765	1
10	-0.23583125	-0.16049118	0
11	-1.09629664	0.26761214	0
13	-1.66994024	0.43885347	0
14	-0.52265305	-1.50188159	0
15	-1.86115477	0.32469259	0
16	-0.80947485	0.26761214	0
17	0.91145593	-1.30210004	1
21	0.72024140	-1.38772071	1
23	1.00706320	-0.84545650	1
24	0.72024140	-1.38772071	1
25	0.81584866	-1.35918049	1
26	0.91145593	-1.44480115	1
27	1.10267046	-1.21647938	1
28	0.91145593	-1.15939893	1
30	-0.61826032	-1.50188159	0
31	-0.61826032	0.09637081	0
33	-1.57433297	-1.55896204	0
36	-0.23583125	-1.24501960	0
37	-0.42704579	-1.21647938	0
39	-1.09629664	0.03929037	0
40	-1.00068938	-1.13085871	0
41	-1.00068938	-1.53042182	0
42	-0.42704579	-0.56005428	0
43	-0.23583125	1.06673835	0
44	-0.71386758	-1.58750226	0
47	-1.19190391	0.23907192	0
49	-0.71386758	1.83732433	1
50	-0.61826032	0.52447414	0
51	-1.28751117	-1.10231849	0
53	-0.80947485	0.35323281	0
54	-0.23583125	-1.35918049	0
55	-1.00068938	-0.36027273	0
56	-1.28751117	-0.44589340	0

Showing 1 to 38 of 300 entries

- Feature scaled test dataset


test_set

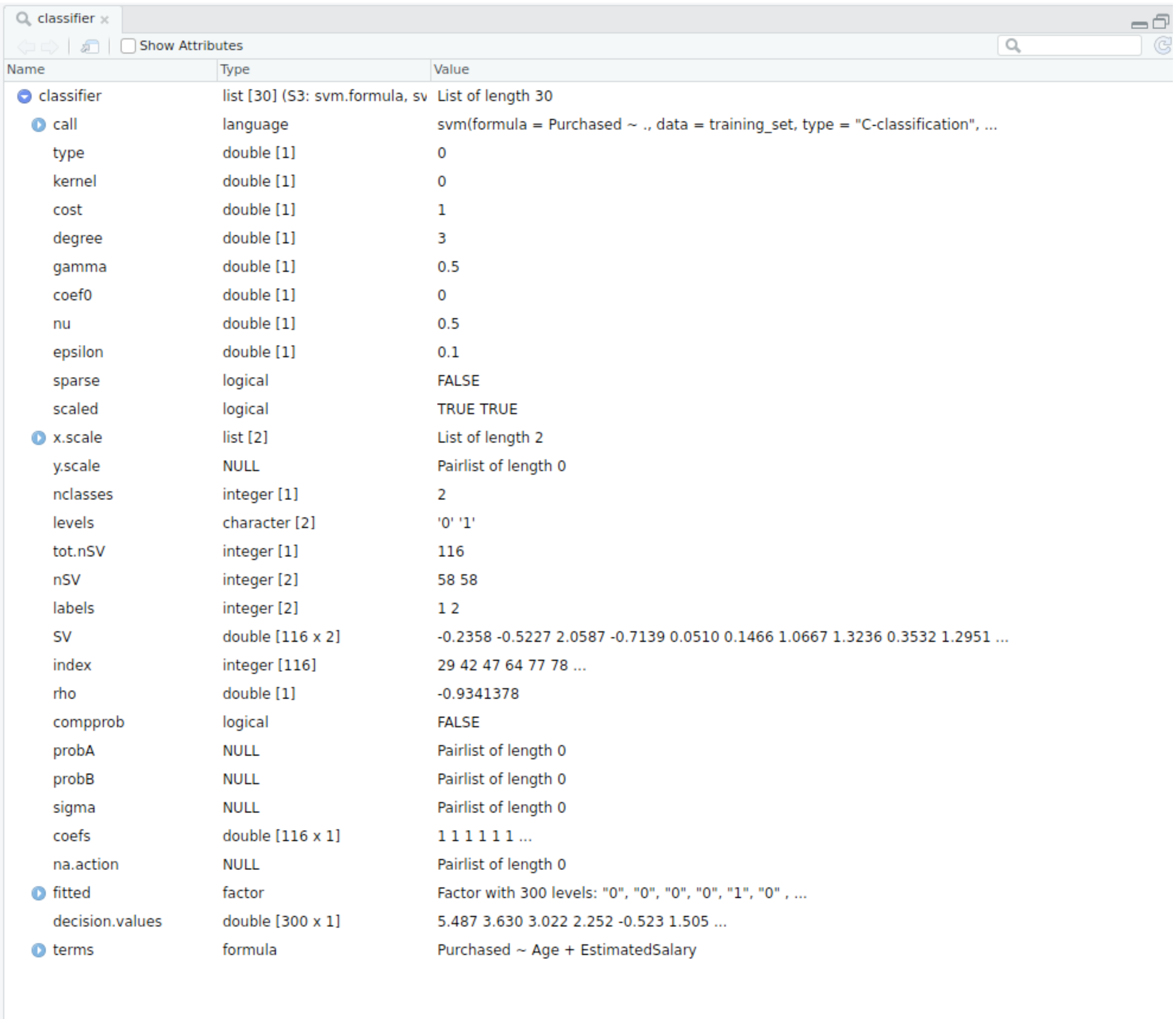
	Age	EstimatedSalary	Purchased
2	-0.30419063	-1.51354339	0
4	-1.05994374	-0.32456026	0
5	-1.81569686	0.28599864	0
9	-1.24888202	-1.09579256	0
12	-1.15441288	-0.48523366	0
18	0.64050076	-1.32073531	1
19	0.73496990	-1.25646596	1
20	0.92390818	-1.22433128	1
22	0.82943904	-0.58163769	1
29	-0.87100546	-0.77444577	0
32	-1.05994374	2.24621408	1
34	-0.96547460	-0.74231109	0
35	-1.05994374	0.73588415	0
38	-0.77653633	-0.58163769	0
45	-0.96547460	0.54307608	0
46	-1.43782030	-1.51354339	0
48	-1.05994374	-0.42096430	0
52	-1.91016600	-0.74231109	0
66	-1.34335116	-0.29242558	0
69	-1.53228944	-0.13175218	0
74	-0.49312891	1.47498177	0
75	-0.58759805	-1.57781275	0
82	0.07368593	-0.80658045	0
84	-0.30419063	0.67161480	0
85	-0.77653633	-0.16388686	0
86	-0.68206719	1.63565517	1
87	-1.34335116	-0.38882962	0
89	-1.15441288	0.44667204	0
103	-0.58759805	0.60734544	0
104	-0.49312891	2.63183023	1
107	-1.15441288	-1.03152320	0
108	-1.05994374	0.70374947	0
109	-1.15441288	0.60734544	0
117	-0.30419063	0.25386397	0
124	-0.30419063	-0.45309898	0
126	0.07368593	-0.19602154	0
127	0.35709335	-0.06748283	0
131	-0.68206719	-0.29242558	0

Showing 1 to 38 of 100 entries

- **Fitting SVM to the training set**

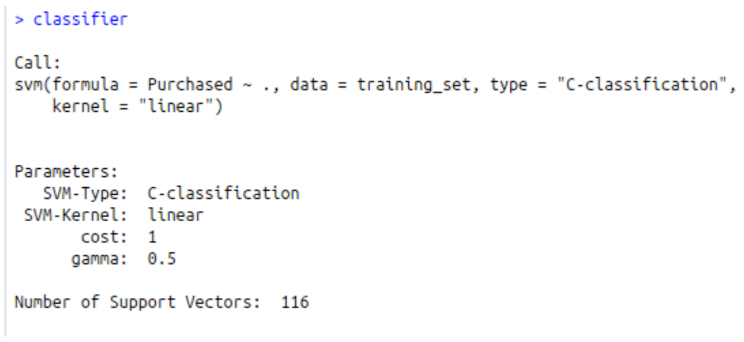
```
# Fitting SVM to the Training set
install.packages('e1071')
library(e1071)

classifier = svm(formula = Purchased ~ .,
                 data = training_set,
                 type = 'C-classification',
                 kernel = 'linear')
```

- **Output:**
  - Classifier detailed



Name	Type	Value
classifier	list [30] (S3: svm.formula, sv	List of length 30
call	language	svm(formula = Purchased ~ ., data = training_set, type = "C-classification", ...
type	double [1]	0
kernel	double [1]	0
cost	double [1]	1
degree	double [1]	3
gamma	double [1]	0.5
coef0	double [1]	0
nu	double [1]	0.5
epsilon	double [1]	0.1
sparse	logical	FALSE
scaled	logical	TRUE TRUE
x.scale	list [2]	List of length 2
y.scale	NULL	Pairlist of length 0
nclasses	integer [1]	2
levels	character [2]	'0' '1'
tot.nSV	integer [1]	116
nSV	integer [2]	58 58
labels	integer [2]	1 2
SV	double [116 x 2]	-0.2358 -0.5227 2.0587 -0.7139 0.0510 0.1466 1.0667 1.3236 0.3532 1.2951 ...
index	integer [116]	29 42 47 64 77 78 ...
rho	double [1]	-0.9341378
compprob	logical	FALSE
probA	NULL	Pairlist of length 0
probB	NULL	Pairlist of length 0
sigma	NULL	Pairlist of length 0
coefs	double [116 x 1]	1 1 1 1 1 1 ...
na.action	NULL	Pairlist of length 0
fitted	factor	Factor with 300 levels: "0", "0", "0", "0", "1", "0" , ...
decision.values	double [300 x 1]	5.487 3.630 3.022 2.252 -0.523 1.505 ...
terms	formula	Purchased ~ Age + EstimatedSalary

- **Classifier in nutshell**

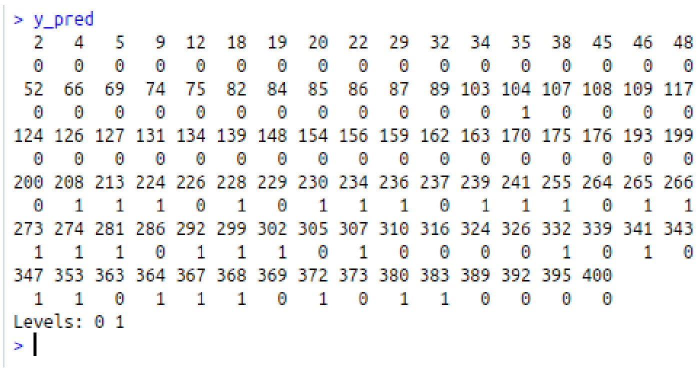


```
> classifier

Call:
svm(formula = Purchased ~ ., data = training_set, type = "C-classification",
    kernel = "linear")


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  1
      gamma:  0.5

Number of Support Vectors:  116
```

- **Predicting the test set result**

```
# Predicting the Test set results
y_pred = predict(classifier, newdata = test_set[-3])
```

- **Output:**

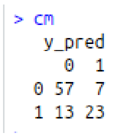


```
> y_pred
  2   4   5   9  12  18  19  20  22  29  32  34  35  38  45  46  48 
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 
 52  66  69  74  75  82  84  85  86  87  89 103 104 107 108 109 117 
  0   0   0   0   0   0   0   0   0   0   0   0   1   0   0   0   0 
124 126 127 131 134 139 148 154 156 159 162 163 170 175 176 193 199 
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 
200 208 213 224 226 228 229 230 234 236 237 239 241 255 264 265 266 
  0   1   1   1   0   1   0   1   1   1   0   1   1   1   0   1   1 
273 274 281 286 292 299 302 305 307 310 316 324 326 332 339 341 343 
  1   1   1   0   1   1   1   0   1   0   0   0   0   1   0   1   0 
347 353 363 364 367 368 369 372 373 380 383 389 392 395 400 
  1   1   0   1   1   1   0   1   0   1   1   0   0   0   0 
Levels: 0 1
> |
```

- **Making Confusion Matrix**

```
# Making the Confusion Matrix
cm = table(test_set[, 3], y_pred)
```

- **Output:**



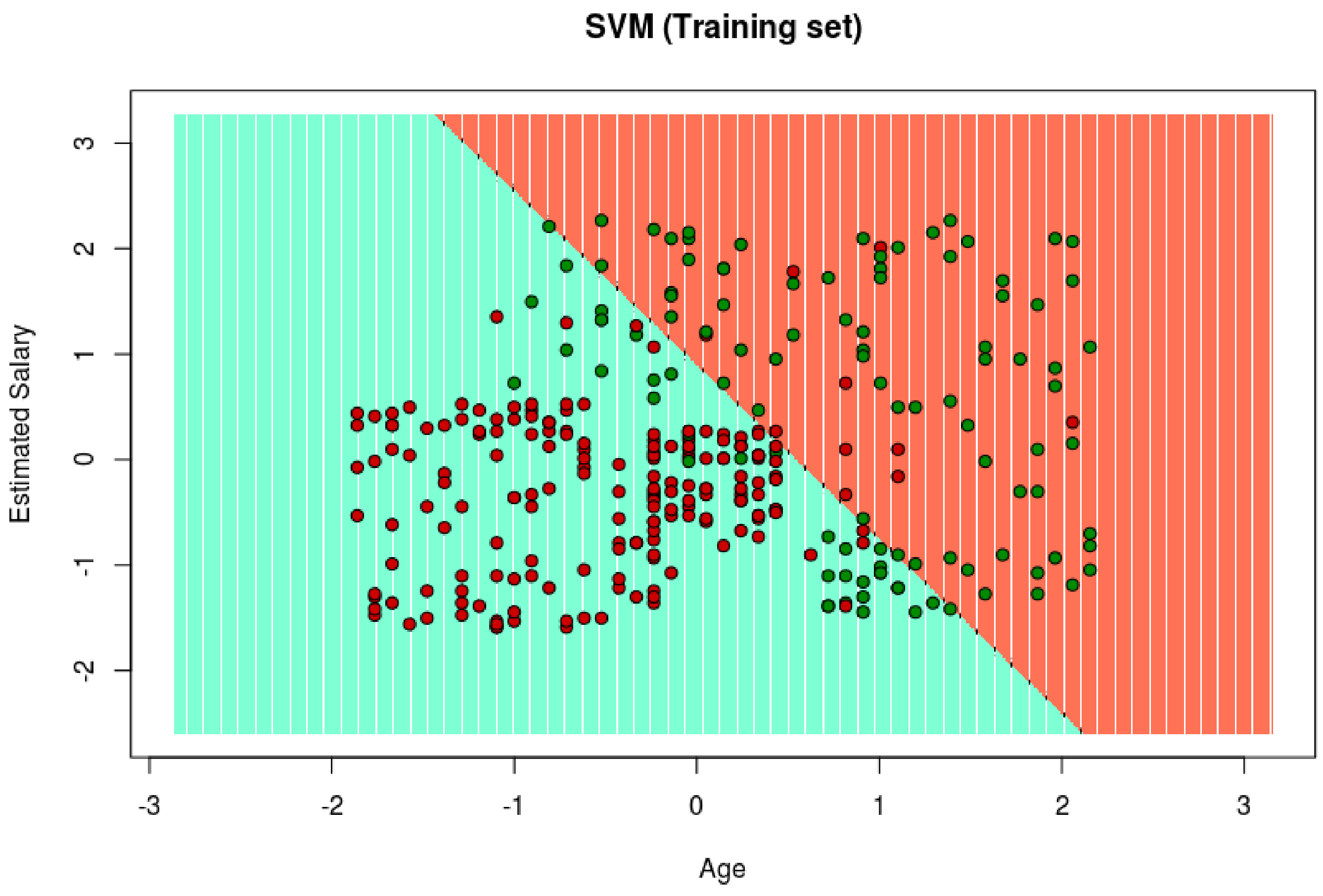
```
> cm
   y_pred
     0  1
  0 57  7
  1 13 23
```

- **Visualizing the Training set results**

```
# installing library ElemStatLearn
library(ElemStatLearn)

# Plotting the training data set results
set = training_set
X1 = seq(min(set[, 1]) - 1, max(set[, 1]) + 1, by = 0.01)
```

```
X2 = seq(min(set[, 2]) - 1, max(set[, 2]) + 1, by = 0.01)

grid_set = expand.grid(X1, X2)
colnames(grid_set) = c('Age', 'EstimatedSalary')
y_grid = predict(classifier, newdata = grid_set)

plot(set[, -3],
     main = 'SVM (Training set)',
     xlab = 'Age', ylab = 'Estimated Salary',
     xlim = range(X1), ylim = range(X2))

contour(X1, X2, matrix(as.numeric(y_grid), length(X1), length(X2)), add
= TRUE)

points(grid_set, pch = '.', col = ifelse(y_grid == 1, 'coral1',
'aquamarine'))

points(set, pch = 21, bg = ifelse(set[, 3] == 1, 'green4', 'red3'))
```

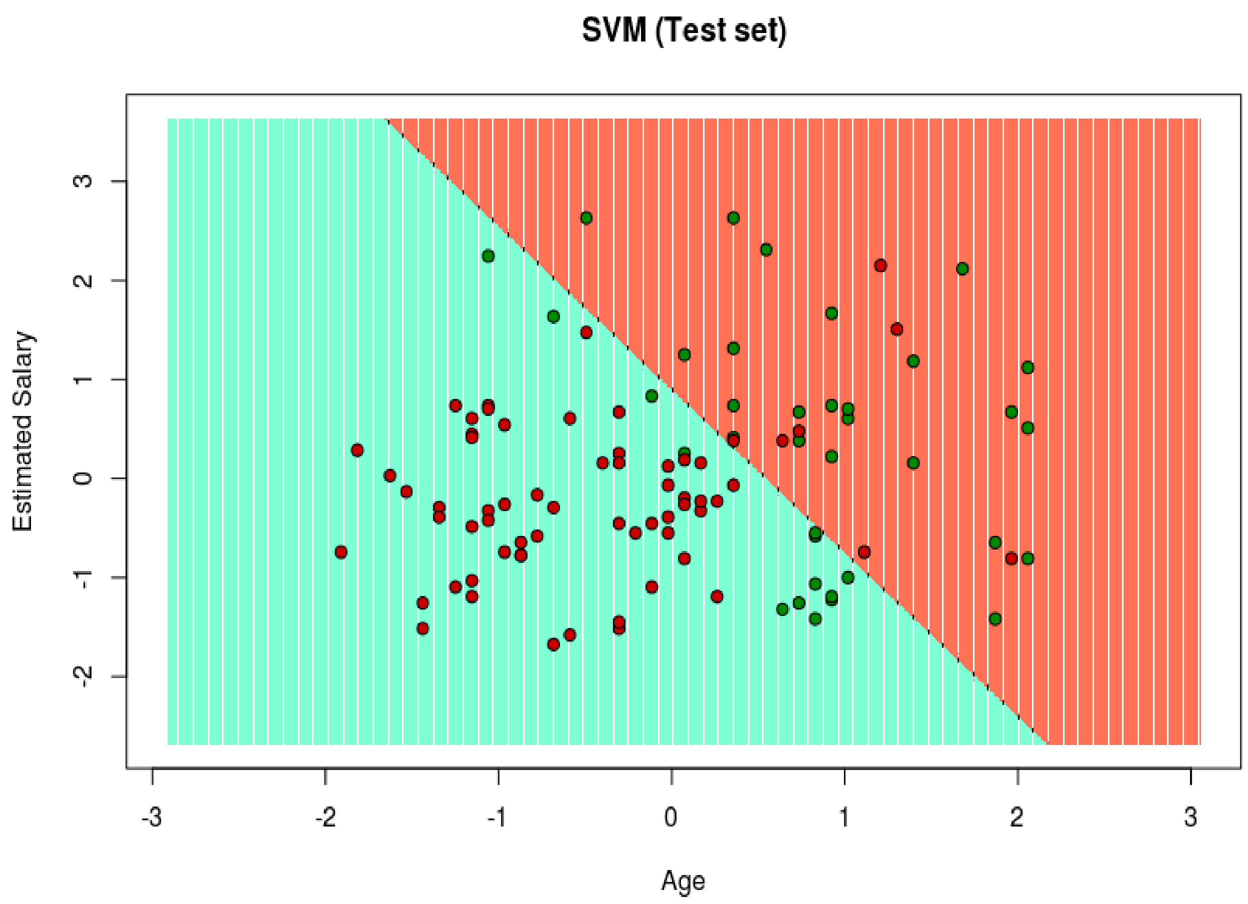
- **Output:**



- **Visualizing the Test set results**

```
set = test_set
X1 = seq(min(set[, 1]) - 1, max(set[, 1]) + 1, by = 0.01)
X2 = seq(min(set[, 2]) - 1, max(set[, 2]) + 1, by = 0.01)

grid_set = expand.grid(X1, X2)
colnames(grid_set) = c('Age', 'EstimatedSalary')
y_grid = predict(classifier, newdata = grid_set)

plot(set[, -3], main = 'SVM (Test set)',
     xlab = 'Age', ylab = 'Estimated Salary',
     xlim = range(X1), ylim = range(X2))

contour(X1, X2, matrix(as.numeric(y_grid), length(X1), length(X2)), add
= TRUE)

points(grid_set, pch = '.', col = ifelse(y_grid == 1, 'coral1',
'aquamarine'))

points(set, pch = 21, bg = ifelse(set[, 3] == 1, 'green4', 'red3'))
```

- **Output:**

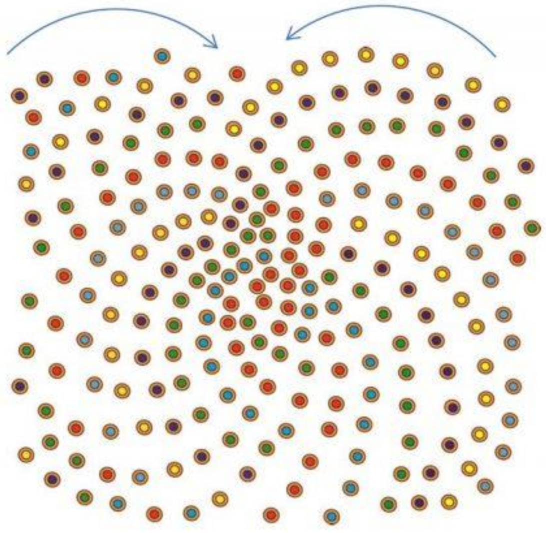


Since in the result, a hyper-plane has been found in the Training set result and verified to be the best one in the Test set result. Hence, SVM has been successfully implemented in R.

Phyllotaxis pattern in Python | A unit of Algorithmic Botany

Introduction | Phyllotaxis

Phyllotaxis/phyllotaxy is the arrangement of leaves on a plant stem & the Phyllotactic spirals form a distinctive class of patterns in nature. The word itself comes from the Greek phullon, meaning "leaf, " and taxis, meaning "arrangement".The basic floral phyllotaxic arrangements include:
1. Spiral Phyllotaxis – In spiral phyllotaxy, the individual floral organs are created in a regular time interval with the same divergent angle. The divergent angle in a flower with spiral phyllotaxy approximates 137.5 degrees, which is indicative of a pattern that follows a **Fibonacci series**.The image below shows the spiral phyllotaxy patterns having both clockwise and anticlockwise spiral patterns.



Important points to note:

1. Fibonacci series typically describe spirals found in nature. It is calculated as a series where the previous pair of numbers sum to the next number in the series. The series is 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89 … .
2. There is actually one set of spirals in a clockwise direction and one set in a counter-clockwise direction.
3. Floral organ spirals follow a numerator and denominator set of offset Fibonacci numbers (1/2, 1/3, 2/5, 3/8, 5/13, 8/21, 13/34 …). The numerator is the number of times or turns around the axis to get back to the initiation origin. The denominator indicates the number of organs initiated during the turns. Therefore, a 2/5 would indicate 2 turns around the axis and 5 organs to return to the origin.
4. e.g – In the pine we have (2, 3), (5, 3), and (5, 8) phyllotaxes, in capituli the pairs found are (21, 34), (55, 34), (55, 89), and (89, 144), and on pineapples with hexagonal scales the triplets (8, 13, 21) or (13, 21, 34) are found, depending on the size of the specimens .
5. The prevalence of the Fibonacci sequence in phyllotaxis is often referred to as "the mystery of phyllotaxis."

Other types of floral phyllotaxic arrangements are:
2. Whorled Phyllotaxis, 3. Simple-whorled Phyllotaxis, 4. Complex-whorled Phyllotaxis & 5. Irregular Phyllotaxis

Formation of the Pattern : Summary

The beautiful arrangement of leaves in some plants, called phyllotaxis, obeys a number of subtle mathematical relationships. For instance, the florets in the head of a sunflower form two oppositely directed spirals: 55 of them clockwise and 34 counterclockwise. Surprisingly,

1. These numbers are consecutive Fibonacci numbers.
2. The ratios of alternate Fibonacci numbers are given by the convergents to ϕ^(-2), where ϕ is the golden ratio, and are said to measure the fraction of a turn between successive leaves on the stalk of a plant:
3. e.g : 1/2 for elm and linden, 1/3 for beech and hazel, 2/5 for oak and apple, 3/8 for poplar and rose, 5/13 for willow and almond, etc.
4. Each new leaf on a plant stem is positioned at a certain angle to the previous one and that this angle is constant between leaves: usually about 137.5 degrees.

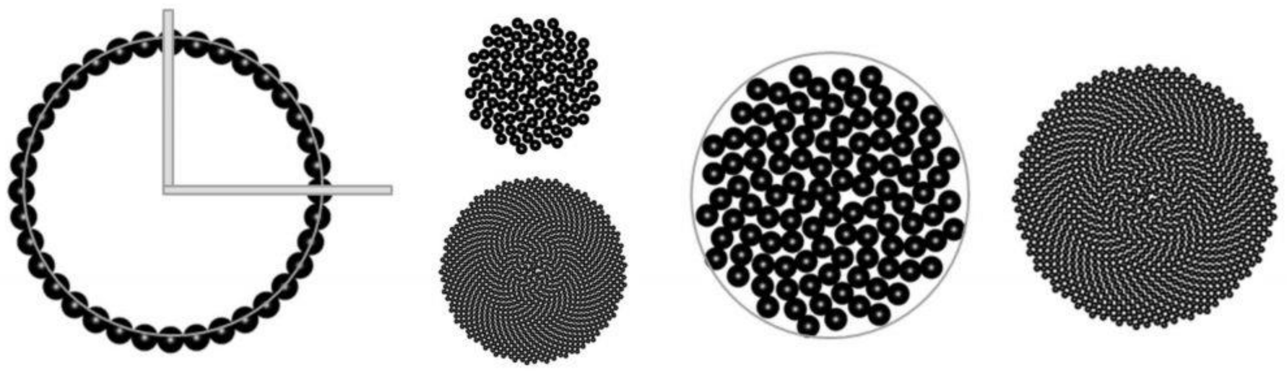
That is, if you look down from above on the plant and measure the angle formed between a line drawn from the stem to the leaf and a corresponding line for the next leaf, you will find that there is generally a fixed angle, called the divergence angle.
Here, we are interested in Spiral phyllotaxy and we will code to form Spiral Phyllotaxy pattern in python using turtle graphics.

Designing the Code

1. We will code two functions, one to draw the phyllotaxy pattern and the other to draw the petals.
2. The petals need to be drawn only after the phyllotaxis pattern is completed.So, we will call the drawPetal() function from inside the drawPhyllPattern() function with the last x & y coordinates being visited after drawing the Phyllotaxis pattern.
3. The drawPetal() function will draw the petals with turtle functions and features, refer Turtle programming.

To code the phyllotaxis pattern, we need to follow these equations:

```
x = r*cos(θ)
y = r*sin(θ)

r, θ can also vary - so the to form phyllotactic pattern we substitutethe cartesian
form
by polar form:

r = c*sqrt(n)
θ = n*137.508°
```



```
Reduces the problem to optimal packing on a disc, so
    r = c*sqrt(n) is from the area of the circle
        Area = πr² and n fills the Area in some units
        c1 * n/π = r²,    c is 1/sqrt(c1/π)
So, r = some constant c * sqrt(n)
```

PseudoCode : Phyllotaxis Pattern

```
IMPORT MODULES ( MATH, TURTLE )

FUNCTION - DrawPhyllotaxisPattern( turtle, t length, petalstart, angle = 137.508, size,
cspread)
    turtleColor("Black")
    FillColor('"Orange")
    Convert angle to radians (Φ)
    initialize ( xcenter,ycenter ) = ( 0,0 )
    Drawing the Pattern Starts:
    For n in Range ( 0,t ):
        r = cspread * sqrt(n)
        θ = n * Φ

        x = r * cos(θ) + xcenter
            y = r * sin(θ) + ycenter

        TURTLE POSITION(x,y)
        START DRAWING():
        if Drawing pattern ends:
            DrawFlowerPetals()

FUNCTION - DrawFlowerPetals(Turtle, x coordinate, y coordinate)
    DRAW using Turtle methods

Create Turtle  =  gfg
Call DrawPhyllotaxisPattern( gfg, t length, petalstart, angle = 137.508, size, cspread)

END
```

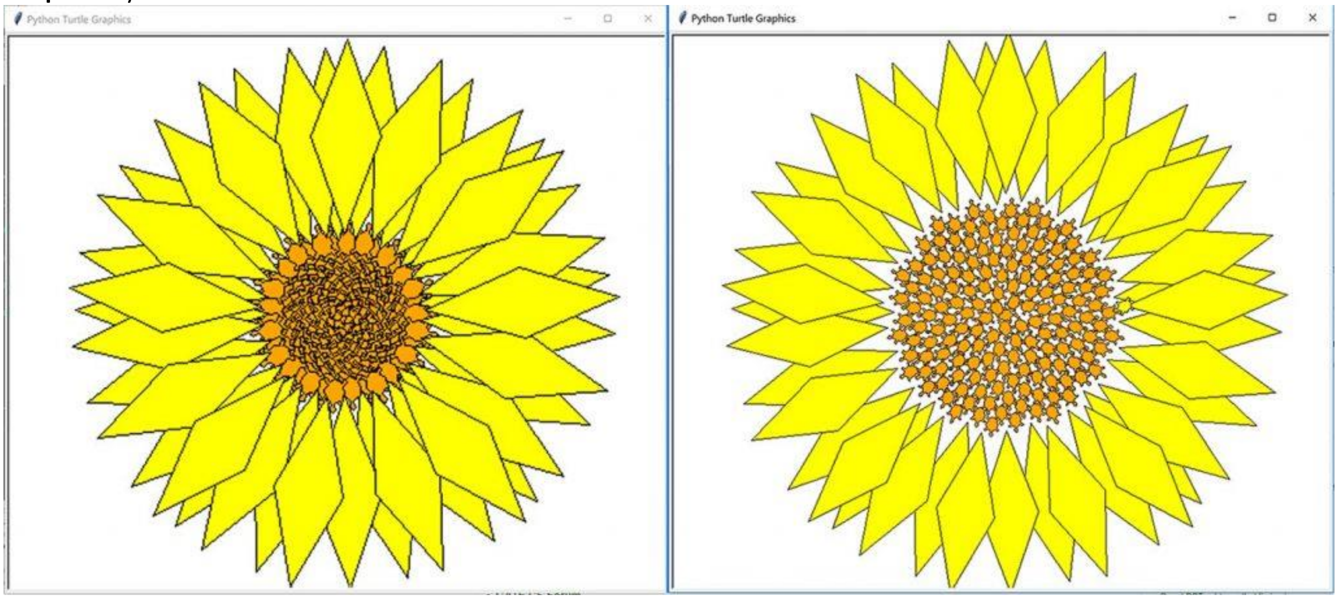
<u>Python Pattern A:</u>

```
import math
import turtle

def drawPhyllPattern(turtle, t, petalstart, angle = 137.508, size = 2,
cspread = 4 ):
    """print a pattern of circles using spiral phyllotactic data"""
    # initialize position
    # turtle.pen(outline=1, pencolor="black", fillcolor="orange")
    turtle.color('black')
    turtle.fillcolor("orange")
    phi = angle * ( math.pi / 180.0 ) #we convert to radian
    xcenter = 0.0
    ycenter = 0.0

    # for loops iterate in this case from the first value until < 4, so
    for n in range (0, t):
        r = cspread * math.sqrt(n)
        theta = n * phi

        x = r * math.cos(theta) + xcenter
        y = r * math.sin(theta) + ycenter
        # move the turtle to that position and draw
        turtle.up()
        turtle.setpos(x, y)
        turtle.down()
        # orient the turtle correctly
        turtle.setheading(n * angle)
        if n > petalstart-1:
            turtle.color("yellow")
            drawPetal(turtle, x, y)
        else: turtle.stamp()


def drawPetal(turtle, x, y ):
    turtle.penup()
    turtle.goto(x, y)
    turtle.pendown()
    turtle.color('black')
    turtle.fillcolor('yellow')
    turtle.begin_fill()
    turtle.right(20)
    turtle.forward(70)
    turtle.left(40)
    turtle.forward(70)
    turtle.left(140)
    turtle.forward(70)
    turtle.left(40)
    turtle.forward(70)
    turtle.penup()
    turtle.end_fill() # this is needed to complete the last petal


gfg = turtle.Turtle()
gfg.shape("turtle")
gfg.speed(0) # make the turtle go as fast as possible
drawPhyllPattern(gfg, 200, 160, 137.508 )
gfg.penup()
gfg.forward(1000)
```

Python Pattern B:

```
import math
import turtle

def drawPhyllotacticPattern( t, petalstart, angle = 137.508, size = 2,
cspread = 4 ):
        """print a pattern of circles using spiral phyllotactic data"""
        # initialize position
        turtle.pen(outline=1, pencolor="black", fillcolor="orange")
        # turtle.color("orange")
        phi = angle * ( math.pi / 180.0 )
        xcenter = 0.0
        ycenter = 0.0
        # for loops iterate in this case from the first value until < 4, so
        for n in range (0, t):
                r = cspread * math.sqrt(n)
                theta = n * phi

                x = r * math.cos(theta) + xcenter
                y = r * math.sin(theta) + ycenter

                # move the turtle to that position and draw
                turtle.up()
                turtle.setpos(x, y)
                turtle.down()
                # orient the turtle correctly
                turtle.setheading(n * angle)


                if n > petalstart-1:
                        #turtle.color("yellow")
                        drawPetal(x, y)
                else: turtle.stamp()


def drawPetal( x, y ):
        turtle.up()
        turtle.setpos(x, y)
        turtle.down()
        turtle.begin_fill()
        #turtle.fill(True)
        turtle.pen(outline=1, pencolor="black", fillcolor="yellow")
        turtle.right(20)
        turtle.forward(100)
        turtle.left(40)
        turtle.forward(100)
        turtle.left(140)
        turtle.forward(100)
        turtle.left(40)
        turtle.forward(100)
        turtle.up()
        turtle.end_fill() # this is needed to complete the last petal



turtle.shape("turtle")
turtle.speed(0) # make the turtle go as fast as possible
drawPhyllotacticPattern( 200, 160, 137.508, 4, 10 )
turtle.exitonclick() # lets you x out of the window when outside of idle
```

Output: Phyllotaxis Patterns.



Link: https://www.geeksforgeeks.org/algorithmic-botany-phyllotaxis-python/

How to get synonyms/antonyms from NLTK WordNet in Python?

WordNet is a large lexical database of English. Nouns, verbs, adjectives and adverbs are grouped into sets of cognitive synonyms (synsets), each expressing a distinct concept. Synsets are interlinked by means of conceptual-semantic and lexical relations.
WordNet's structure makes it a useful tool for computational linguistics and natural language processing.

WordNet superficially resembles a thesaurus, in that it groups words together based on their meanings. However, there are some important distinctions.

- First, WordNet interlinks not just word forms—strings of letters—but specific senses of words. As a result, words that are found in close proximity to one another in the network are semantically disambiguated.
- Second, WordNet labels the semantic relations among words, whereas the groupings of words in a thesaurus does not follow any explicit pattern other than meaning similarity.

```
# First, you're going to need to import wordnet:
from nltk.corpus import wordnet

# Then, we're going to use the term "program" to find synsets like so:
syns = wordnet.synsets("program")

# An example of a synset:
print(syns[0].name())
```

```
# Just the word:
print(syns[0].lemmas()[0].name())

# Definition of that first synset:
print(syns[0].definition())

# Examples of the word in use in sentences:
print(syns[0].examples())
```

The output will look like:

```
plan.n.01
plan
a series of steps to be carried out or goals to be accomplished
['they drew up a six-step plan', 'they discussed plans for a new bond issue']
```

Next, how might we discern synonyms and antonyms to a word? The lemmas will be synonyms, and then you can use .antonyms to find the antonyms to the lemmas. As such, we can populate some lists like:

```
import nltk
from nltk.corpus import wordnet
synonyms = []
antonyms = []

for syn in wordnet.synsets("good"):
    for l in syn.lemmas():
        synonyms.append(l.name())
        if l.antonyms():
            antonyms.append(l.antonyms()[0].name())

print(set(synonyms))
print(set(antonyms))
```

The output will be two sets of synonyms and antonyms

```
{'beneficial', 'just', 'upright', 'thoroughly', 'in_force', 'well', 'skilful', 'skillful', 'sound', 'unspoiled', 'expert', 'proficient', 'in_effect', 'honorable', 'adept', 'secure', 'commodity', 'estimable', 'soundly', 'right', 'respectable', 'good', 'serious', 'ripe', 'salutary', 'dear', 'practiced', 'goodness', 'safe', 'effective', 'unspoilt', 'dependable', 'undecomposed', 'honest', 'full', 'near', 'trade_good'} {'evil', 'evilness', 'bad', 'badness', 'ill'}
```

Now , let's compare the similarity index of any two words

```
import nltk
from nltk.corpus import wordnet
# Let's compare the noun of "ship" and "boat:"

w1 = wordnet.synset('run.v.01') # v here denotes the tag verb
w2 = wordnet.synset('sprint.v.01')
print(w1.wup_similarity(w2))
```

```
Output:
0.857142857143
```

```
w1 = wordnet.synset('ship.n.01')
w2 = wordnet.synset('boat.n.01') # n denotes noun
print(w1.wup_similarity(w2))
```

```
Output:
0.9090909090909091
```

Removing stop words with NLTK in Python

The process of converting data to something a computer can understand is referred to as **pre-processing.** One of the major forms of pre-processing is to filter out useless data. In natural language processing, useless words (data), are referred to as stop words.

What are Stop words?

Stop Words: A stop word is a commonly used word (such as "the", "a", "an", "in") that a search engine has been programmed to ignore, both when indexing entries for searching and when retrieving them as the result of a search query.

We would not want these words taking up space in our database, or taking up valuable processing time. For this, we can remove them easily, by storing a list of words that you consider to be stop words. NLTK(Natural Language Toolkit) in python has a list of stopwords stored in 16 different languages. You can find them in the nltk_data directory. *home/pratima/nltk_data/corpora/stopwords* is the directory address.(Do not forget to change your home directory name)

Sample text with Stop Words	Without Stop Words
GeeksforGeeks – A Computer Science Portal for Geeks	GeeksforGeeks , Computer Science, Portal ,Geeks
Can listening be exhausting?	Listening, Exhausting
I like reading, so I read	Like, Reading, read

To check the list of stopwords you can type the following commands in the python shell.

```
import nltk
from nltk.corpus import stopwords
 set(stopwords.words('english'))
```

{'ourselves', 'hers', 'between', 'yourself', 'but', 'again', 'there', 'about', 'once', 'during', 'out', 'very', 'having', 'with', 'they', 'own', 'an', 'be', 'some', 'for', 'do', 'its', 'yours', 'such', 'into', 'of', 'most', 'itself', 'other', 'off', 'is', 's', 'am', 'or', 'who', 'as', 'from', 'him', 'each', 'the', 'themselves', 'until', 'below', 'are', 'we', 'these', 'your', 'his', 'through', 'don', 'nor', 'me', 'were', 'her', 'more', 'himself', ‘this’, ‘down’, ‘should’, ‘our’, ‘their’, ‘while’, ‘above’, ‘both’, ‘up’, ‘to’, ‘ours’, ‘had’, ‘she’, ‘all’, ‘no’, ‘when’, ‘at’, ‘any’, ‘before’, ‘them’, ‘same’, ‘and’, ‘been’, ‘have’, ‘in’, ‘will’, ‘on’, ‘does’, ‘yourselves’, ‘then’, ‘that’, ‘because’, ‘what’, ‘over’, ‘why’, ‘so’, ‘can’, ‘did’, ‘not’, ‘now’, ‘under’, ‘he’, ‘you’, ‘herself’, ‘has’, ‘just’, ‘where’, ‘too’, ‘only’, ‘myself’, ‘which’, ‘those’, ‘i’, ‘after’, ‘few’, ‘whom’, ‘t’, ‘being’, ‘if’, ‘theirs’, ‘my’, ‘against’, ‘a’, ‘by’, ‘doing’, ‘it’, ‘how’, ‘further’, ‘was’, ‘here’, ‘than’}

Note: You can even modify the list by adding words of your choice in the english .txt. file in the stopwords directory.

Removing stop words with NLTK

The following program removes stop words from a piece of text:

```
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

example_sent = "This is a sample sentence, showing off the stop words
filtration."

stop_words = set(stopwords.words('english'))

word_tokens = word_tokenize(example_sent)

filtered_sentence = [w for w in word_tokens if not w in stop_words]

filtered_sentence = []

for w in word_tokens:
    if w not in stop_words:
        filtered_sentence.append(w)

print(word_tokens)
print(filtered_sentence)
```

Output:

```
['This', 'is', 'a', 'sample', 'sentence', ',', 'showing',
'off', 'the', 'stop', 'words', 'filtration', '.']
['This', 'sample', 'sentence', ',', 'showing', 'stop',
'words', 'filtration', '.']
```

Performing the Stopwords operations in a file

In the code below, text.txt is the original input file in which stopwords are to be removed. filteredtext.txt is the output file. It can be done using following code:

```
import io
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
#word_tokenize accepts a string as an input, not a file.
stop_words = set(stopwords.words('english'))
file1 = open("text.txt")
```

```
line = file1.read()# Use this to read file content as a stream:
words = line.split()
for r in words:
    if not r in stop_words:
        appendFile = open('filteredtext.txt','a')
        appendFile.write(" "+r)
        appendFile.close()
```

This is how we are making our processed content more efficient by removing words that do not contribute to any future operations.

Tokenize text using NLTK in python

To run the below python program, (NLTK) natural language toolkit has to be installed in your system.
The NLTK module is a massive tool kit, aimed at helping you with the entire Natural Language Processing (NLP) methodology.
In order to install NLTK run the following commands in your terminal.

- **sudo pip install nltk**
- Then, enter the python shell in your terminal by simply typing **python**
- Type **import nltk**
- **nltk.download('all')**

The above installation will take quite some time due to the massive amount of tokenizers, chunkers, other algorithms, and all of the corpora to be downloaded.

Some terms that will be frequently used are :

- **Corpus** – Body of text, singular. Corpora is the plural of this.
- **Lexicon** – Words and their meanings.
- **Token** – Each "entity" that is a part of whatever was split up based on rules. For examples, each word is a token when a sentence is "tokenized" into words. Each sentence can also be a token, if you tokenized the sentences out of a paragraph.

So basically tokenizing involves splitting sentences and words from the body of the text.

```
# import the existing word and sentence tokenizing
# libraries
from nltk.tokenize import sent_tokenize, word_tokenize


text = "Natural language processing (NLP) is a field " + \
       "of computer science, artificial intelligence " + \
       "and computational linguistics concerned with " + \
       "the interactions between computers and human " + \
       "(natural) languages, and, in particular, " + \
       "concerned with programming computers to " + \
       "fruitfully process large natural language " + \
       "corpora. Challenges in natural language " + \
       "processing frequently involve natural " + \
       "language understanding, natural language" + \
       "generation frequently from formal, machine" + \
       "-readable logical forms), connecting language " + \
       "and machine perception, managing human-" + \
       "computer dialog systems, or some combination " + \
       "thereof."

print(sent_tokenize(text))
print(word_tokenize(text))`
```

OUTPUT
[‘Natural language processing (NLP) is a field of computer science, artificial intelligence and computational linguistics concerned with the interactions between computers and human (natural) languages, and, in particular, concerned with programming computers to fruitfully process large natural language corpora.’, ‘Challenges in natural language processing frequently involve natural language understanding, natural language generation (frequently from formal, machine-readable logical forms), connecting language and machine perception, managing human-computer dialog systems, or some combination thereof.’]
*[‘Natural’, ‘language’, ‘processing’, ‘(‘, ‘NLP’, ‘)’, ‘is’, ‘a’, ‘field’, ‘of’, ‘computer’, ‘science’, ‘,’, ‘artificial’, ‘intelligence’, ‘and’, ‘computational’, ‘linguistics’, ‘concerned’, ‘with’, ‘the’, ‘interactions’, ‘between’, ‘computers’, ‘and’, ‘human’, ‘(‘, ‘natural’, ‘)’, ‘languages’, ‘,’, ‘and’, ‘,’, ‘in’, ‘particular’, ‘,’, ‘concerned’, ‘with’, ‘programming’, ‘computers’, ‘to’, ‘fruitfully’, ‘process’, ‘large’, ‘natural’, ‘language’, ‘corpora’, ‘.’, ‘Challenges’, ‘in’, ‘natural’, ‘language’, ‘processing’, ‘frequently’, ‘involve’, ‘natural’, ‘language’, ‘understanding’, ‘,’, ‘natural’, ‘language’, ‘generation’, ‘(‘, ‘frequently’, ‘from’, ‘formal’, ‘,’, ‘machine-readable’, ‘logical’, ‘forms’, ‘)’, ‘,’, ‘connecting’, ‘language’, ‘and’, ‘machine’, ‘perception’, ‘,’, ‘managing’, ‘human-*computer*’, ‘dialog’, ‘systems’, ‘,’, ‘or’, ‘some’, ‘combination’, ‘thereof’, ‘.’]*

So there, we have created tokens, which are sentences initially and words later.

Solve ML Problem from here: https://practice.geeksforgeeks.org/explore/?page=1&sortBy=accuracy