SRM VALLIAMMAI ENGINEERING COLLEGE

(An Autonomous Institution)

SRM Nagar, Kattankulathur-603203

DEPARTMENT OF INFORMATION TECHNOLOGY

ACADEMIC YEAR: 2026- 2027

ODD SEMESTER

LAB MANUAL

(REGULATION - 2023)

DS3165 - MACHINE LEARNING LABORATORY

FIRST SEMESTER

M.Tech-Data Science

Prepared By

Ms. M. Janaki, AP (OG)

E.No	EXPERIMENT NAME	Pg. No.
A	PEO, PO, PSO	**03**
B	Syllabus with co-po matrix	**06**
C	Co, Co-Po Matrix, Co-PSO Matrix	**08**
D	Mode of Assessment	**09**
E	Introduction/ Description of major software & Hardware involved in lab	**10**
1	Study and usage of python tool.	**11**
2	Implement a classifier	**20**
3	Develop a predictive model.	**39**
4	Implement a decision tree algorithm	**50**
5	Implement a back propagation algorithm.	**72**
6	Implement similarity and non- similarity -based clustering algorithm	**78**
7	Implement clustering algorithm for any data set.	**84**
8	Apply reinforcement learning algorithm.	**86**
9	Implement and demonstrate the Candidate-Elimination algorithm.	**100**
10	Implement a k-Nearest Neighbor algorithm.	**104**
11	Implement the non-parametric Locally Weighted Regression algorithm	**112**

PROGRAMME EDUCATIONAL OBJECTIVES (PEOs)

PROGRAMME OUTCOMES (POs)

After going through the four years of study, Information Technology Graduates will exhibit ability to:

PO#	Graduate Attribute	Programme Outcome
1	Engineering knowledge	Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization for the solution of complex engineering problems.
2	Problem analysis	Identify, formulate, research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3	Design/development of solutions	Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for public health and safety, and cultural, societal, and environmental considerations.
4	Conduct investigations of complex problems	Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions

5	Modern tool usage	Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools, including prediction and modeling to complex engineering activities, with an understanding of the limitations.
6	The engineer and society	Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal, and cultural issues and the consequent responsibilities relevant to the professional engineering practice
7	Environment and sustainability	Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8	Ethics	Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice
9	Individual and team work	Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings
10	Communication	Communicate effectively on complex engineering activities with the engineering community and with the society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations,and give and receive clear instructions
11	Project management and finance	Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments
12	Life-long learning	Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change

PROGRAM SPECIFIC OUTCOMES (PSOs)

By the completion of Information Technology program the student will have following Program specific outcomes

DS3165.1	**PSO1**	**1**	Used to analyze the datasets and provide the security.
	PSO2	**2**	Provide the advanced data storages and processing.
DS3165.2	**PSO1**	**1**	Using various libraries for machine learning techniques.
	PSO2	**2**	Using python to solve machine learning algorithms.
DS3165.3	**PSO1**	**1**	Used to solve problems using reinforcement algorithms.
	PSO2	**1**	Uses reinforcement algorithms for next generation database analysis.
	PSO4	**1**	New concepts and preprocessing used to solve problems.
DS3165.4	**PSO1**	**2**	Classification and clustering algorithms used for next generation data sets.
	PSO2	**2**	Advanced algorithms used in storage and processing.
DS3165.5	**PSO1**	**1**	Used to analyze various machine learning algorithms.
	PSO2	**1**	Uses the new concepts and preprocessing to solve the problems.

OBJECTIVES:

- To make use of Data sets in implementing the machine learning algorithms
- To implement the machine learning concepts and algorithms in any suitable language of choice.
- To understand the reinforcement concept
- To evaluate the classification and clustering algorithms
- Learn the usage of python in implementing machine Learning algorithms

LIST OF EXPERIMENTS:

1. Study and usage of python tool.
2. Implement a classifier.
3. Develop a predictive model.
4. Implement a decision tree algorithm.
5. Implement back propagation algorithm.
6. Implement similarity based clustering algorithm.
7. Implement clustering algorithm for any data set.
8. Apply reinforcement learning.
9. Implement and demonstrate the Candidate-Elimination algorithm
10. Implement k-Nearest Neighbour algorithm
11. Implement the non-parametric Locally Weighted Regression algorithm

TOTAL: 45 PERIODS

OUTCOME

At the end of the course, the student should be able to:

- ❖ Apply various classification and clustering techniques for problems using tools likePython.
- ❖ Implement solutions for various prediction problems using tools.
- ❖ Design and development of game and traffic control system using reinforcementlearning.
- ❖ Implement the machine learning algorithms using python
- ❖ Design and develop solutions for the machine learning algorithms using various datasets

LIST OF EQUIPMENT FOR A BATCH OF 20 STUDENTS

SOFTWARE:

Python 3 interpreter for Windows/Linux, Open Source tools

HARDWARE:

Standalone desktops 20 Nos.

CO-PO MATRIX

CO	PO												PSO			
	1	2	3	4	5	6	7	8	9	10	11	12	1	2	3	4
CO 1	3	3	3	1					1			2	1	2		
CO 2	3	3	3	2		1			1			1	1	2		
CO 3	3	3	3	1					1			1	1	1		1
CO 4	3	3	3	2					1			2	2	1		
CO 5	3	3	3	1									1	1		
Average	3	3	3	1		1			1			2	1	1		1

CO-PSO MATRIX

CO	PSO1	PSO2	PSO3	PSO4
DS3165	1	1	-	1

EVALUATION PROCEDURE FOR EACH EXPERIMENTS

S.No	Description	Mark
1.	Aim & Pre-Lab discussion	20
2.	Observation	20
3.	Conduction and Execution	30
4.	Output & Result	10
5.	Viva	20
Total		**100**

INTERNAL ASSESSMENT FOR LABORATORY

S.No	Description	Mark
1.	Observation	10
2.	Record	5
3.	Model Test	5
Total		**20**

SOFTWARE DESCRIPTION

Anaconda is a distribution of the python and R programming languages for scientific computing (data science, machine learning applications, large-scale data processing, predictive analytics, etc.), that aims to simply package management and deployment.

Developer(s): Anaconda , Inc.(previously cont..)

Operating system: Windows, macOS, Linux

License: Freemium (Miniconda and the Individ..)

Ex No: 1 STUDY AND USAGE OF PYTHON TOOL

Aim:

To study about python and its additional libraries needed for machine learningimplementation

Description of python:

- Python is an object-oriented (based around data), high-level (easier for humans to understand) programming language. First launched in 1992, it's built in a way that it's relatively intuitive to write and understand. As such, it's an ideal coding language for those who want rapid development.
- There are many reasons for the ubiquity of Python, including:

- **Its ease of use**. For those who are new to coding and programming, Python can be an excellent firststep. It's relatively easy to learn, making it a great way to start building your programming knowledge.
- **Its simple syntax**. Python is relatively easy to read and understand, as its syntax is more like English. Its straightforward layout means that you can work out what each line of code is doing.
- **Its thriving community**. As it's an open-source language, anyone can use Python to code. What's more, there is a community that supports and develops the ecosystem, adding their own contributions and libraries.
- **Its versatility**. As we'll explore in more detail, there are many uses for Python. Whether you're interested in data visualisation, artificial intelligence or web development, you can find a use for the language.

Why learn Python?

So, we know why Python is so popular at the moment, but why should you learn how to use it? Aside from the ease of use and versatility mentioned above, there are several good reasons to learn Python:

- **Python developers are in demand**. Across a wide range of fields, there is a demand for those with Python skills. If you're looking to start or change your career, it could be a vital skill to help you.
- **It could lead to a well-paid career**. Data suggests that the median annual salary for those with Python skills is around £65,000 in the UK.
- **There will be many job opportunities**. Given that Python can be used in many emerging technologies, such as AI, machine learning, and data analytics, it's likely that it's a future-proof skill. Learning Python now could benefit you across your career.

Python can be used for:

1.*AI and machine learning*

Because Python is such a stable, flexible, and simple programming language, it's perfect for various machine learning (ML) and artificial intelligence (AI) projects. In fact, Python is among the favourite languages among data scientists, and there are many Python machine learning and AI libraries and packages available.

If you're interested in this application of Python, our Deep Learning and Python Programming for AI with Microsoft Azure ExpertTrack can help you develop your skills in these areas. You can discover the uses of Python and deep learning while boosting your career in AI.

2. *Data analytics*

Much like AI and machine learning, data analytics is another rapidly developing field that utilises Python programming. At a time when we're creating more data than ever before, there is a need for those who can collect, manipulate and organise the information.

Python for data science and analytics makes sense. The language is easy-to-learn, flexible, and well-supported, meaning it's relatively quick and easy to use for analysing data. When working with large amounts of information, it's useful for manipulating data and carrying out repetitive tasks.
You can learn about data analytics using Python with our ExpertTrack, which will help you develop practical data analytics skills.

3. *Data visualisation*

Data visualisation is another popular and developing area of interest. Again, it plays into many of the strengths of Python. As well as its flexibility and the fact it's open-source, Python provides a variety of graphing libraries with all kinds of features.
Whether you're looking to create a simple graphical representation or a more interactive plot, you can find a library to match your needs. Examples include Pandas Visualization and Plotly. The possibilities are vast, allowing you to transform data into meaningful insights.
If data visualisation with Python sounds appealing, check out our 12-week ExpertTrack on the subject. You'll learn how to leverage Python libraries to interpret and analyse data sets.

4. *Programming applications*

You can program all kinds of applications using Python. The general-purpose language can be used to read and create file directories, create GUIs and APIs, and more. Whether it's blockchain applications, audio and video apps, or machine learning applications, you can build them all with Python.
We also have an ExpertTrack on programming applications with Python, which can help to kick-start your programming career. Over the course of 12 weeks, you'll gain an introduction on how to use Python, and start programming your own applications using it.

5. *Web development*

Python is a great choice for web development. This is largely due to the fact that there are many Python web development frameworks to choose from, such as Django, Pyramid, and Flask. These frameworks have been used to create sites and services such as Spotify, Reddit and Mozilla.

Thanks to the extensive libraries and modules that come with Python frameworks, functions such as database access, content management, and data authorisation are all possible and easily accessible. Given its versatility, it's hardly surprising that Python is so widely used in web development.

6. *Game development*

Although far from an industry-standard in game development, Python does have its uses in the industry. It's possible to create simple games using the programming language, which means it can be a useful tool

for quickly developing a prototype. Similarly, certain functions (such as dialogue tree creation) are possible in Python.

If you're new to either Python or game development, then you can also discover how to make a text-based game in Python. In doing so, you can work on a variety of skills and improve your knowledge in various areas.

7. *Language development*

The simple and elegant design of Python and its syntax means that it has inspired the creation of new programming languages. Languages such as Cobra, CoffeeScript, and Go all use a similar syntax to Python.

This fact also means that Python is a useful gateway language. So, if you're totally new to programming, understanding Python can help you branch out into other areas more easily.

8. *Finance*

Python is increasingly being utilised in the world of finance, often in areas such as quantitative and qualitative analysis. It can be a valuable tool in determining asset price trends and predictions, as well as in automating workflows across different data sources.

As mentioned already, Python is an ideal tool for working with big data sets, and there are many libraries available to help with compiling and processing information. As such, it's one of the preferred languages in the finance industry.

Python can help.
Emerging technologies such as natural language processing (NLP) are also likely to be relevant to those working in SEO. Python can be a powerful tool in developing these NLP skills and understanding how people search and how search engines return results.

10. Design

When asking 'what is Python used for?' you probably weren't expecting design to feature on the list. However, Python can be used to develop graphic design applications. Surprisingly, the language is used across a range of 2D imaging software, such as Paint Shop Pro and Gimp.

Python is even used in 3D animation software such as Lightwave, Blender, and Cinema 4D, showing just how versatile the language is.

Python libraries:

- Python's standard library is very extensive, offering a wide range of facilities as indicated by the long table of contents listed below. The library contains built-in modules (written in C) that provide access to system functionality such as file I/O that would otherwise be inaccessible to Python programmers, as well as modules written in Python that provide standardized solutions for many problems that occur in everyday programming. Some of these modules are explicitly designed to encourage and enhance the portability of Python programs by abstracting away platform-specifics into platform-neutral APIs.
- The Python installers for the Windows platform usually include the entire standard library and often

also include many additional components. For Unix-like operating systems Python is normally provided as a collection of packages, so it may be necessary to use the packaging tools provided with the operating system to obtain some or all of the optional components.

❖ In addition to the standard library, there is a growing collection of several thousand components (from individual programs and modules to packages and entire application development frameworks), available from the Python Package Index.

These are also the Python libraries for Data Science.

- Matplotlib. Matplotlib helps with data analyzing, and is a numerical plotting library. ...
- Pandas. Like we've said before, Pandas is a must for data-science. ...
- Requests. ...
- NumPy. ...
- SQLAlchemy. ...
- BeautifulSoup. ...
- Pyglet. ...
- SciPy.

Top Python Libraries List

Now that we do understand a bit about what libraries are and what Python is, let us do a deep dive into some of the most commonly used libraries in Python:

Pandas

Pandas is a BSD (Berkeley Software Distribution) licensed open source library. This popular library is widely used in the field of data science. They are primarily used for data analysis, manipulation, cleaning, etc. Pandas allow for simple data modeling and data analysis operations without the need to switch to another language such as R. Usually, Python libraries use the following types of data:

- Data in a dataset.
- Time series containing both ordered and unordered data.
- Rows and columns of matrix data are labelled.
- Unlabeled information
- Any other type of statistical information

Pandas can do a wide range of tasks, including:

- The data frame can be sliced using Pandas.
- Data frame joining and merging can be done using Pandas.
- Columns from two data frames can be concatenated using Pandas.
- In a data frame, index values can be changed using Pandas.
- In a column, the headers can be changed using Pandas.
- Data conversion into various forms can also be done using Pandas and many more.

NumPy

NumPy is one of the most widely used open-source Python libraries, focusing on scientific computation. It features built-in mathematical functions for quick computation and supports big matrices and multidimensional data. “Numerical Python” is defined by the term “NumPy.” It can be used in linear algebra, as a multi-dimensional container for generic data, and as a random number generator, among other

things. Some of the important functions in NumPy are arcsin(), arccos(), tan(), radians(), etc. NumPy Array is a Python object which defines an N-dimensional array with rows and columns. In Python, NumPy Array is preferred over lists because it takes up less memory and is faster and more convenient to use.

Numpy

Features:

1. Interactive: Numpy is a very interactive and user friendly library.
2. Mathematics: NumPy simplifies the implementation of difficult mathematical equations.
3. Intuitive: It makes coding and understanding topics a breeze.
4. Lot of Interaction: There is a lot of interaction in it because it is widely utilised, hence there is a lot of open source contribution.

The NumPy interface can be used to represent images, sound waves, and other binary raw streams as an N-dimensional array of real values for visualization. Numpy knowledge is required for full-stack developers to implement this library for machine learning.

Keras

Keras is a Python-based open-source neural network library that lets us experiment with deep neural networks quickly. With deep learning becoming more common, Keras emerges as a great option because, according to the creators, it is an API (Application Programming Interface) designed for humans, not machines. Keras has a higher adoption rate in the industry and research community than TensorFlow or Theano. It is recommended that you install the TensorFlow backend engine before installing Keras.

Features:

1. It runs without a hitch on both the CPU (Central Processing Unit) and GPU (Graphics Processing Unit).
2. Keras supports nearly all neural network models, including fully connected, convolutional, pooling, recurrent, embedding, and so forth. These models can also be merged to create more sophisticated models.
3. Keras' modular design makes it very expressive, adaptable, and suited really well to cutting edge research.
4. Keras is a Python based framework, making it simple to debug and explore different models and projects.

Keras-powered features are already in use at various companies, for instance, Netflix, Uber, Yelp, Instacart, Zocdoc, Square, and a slew of other companies. It is particularly popular among firms that use deep learning to power their products. Keras includes a lot of implementations of standard neural network building elements such as layers, objectives, activation functions, optimizers, and a slew of other tools for working with picture and text data. It also includes a number of pre-processed data sets and pre-trained models, such as MNIST, VGG, Inception, SqueezeNet, ResNet, etc.

TensorFlow

TensorFlow is a high-performance numerical calculation library that is open source. It is also employed in deep learning algorithms and machine learning algorithms. It was created by the Google Brain team

researchers within the Google AI organization and is currently widely utilized by math, physics, and machine learning researchers for complicated mathematical computations. TensorFlow is designed to be fast, and it employs techniques such as XLA (XLA or Accelerated Linear Algebra is a domain-specific compiler for linear algebra that can accelerate TensorFlow models with potentially no source code changes.) to do speedy linear algebra computations.

Features:

- Responsive Construct: We can easily visualize each and every part of the graph with TensorFlow, which is not possible with Numpy or SciKit.
- Adaptable: One of the most essential Tensorflow features is that it is flexible in its operation relatedto Machine Learning models, which means that it has modularity and allows you to make sections of it stand alone.
- It is Simple to Train Machine Learning Models in TensorFlow: Machine Learning models can be readily trained using TensorFlow on both the CPU and GPU for distributed computing.
- Parallel Neural Network Training: TensorFlow allows you to train many neural networks and GPUsat the same time.
- Open Source and a large community: Without a doubt, if it was developed by Google, there is already a significant team of software experts working on constant stability improvements. The nicest partabout this machine learning library is that it is open-source, which means that anyone with internet access can use it.

TensorFlow is used on a regular basis, but only inadvertently, through services like Google Voice Search and Google Photos. TensorFlow's libraries are developed entirely in C and C++. It does, however, have a sophisticated Python front end. Your Python code will be compiled and run on the TensorFlow distributed execution engine, which is written in C and C++. TensorFlow has an almost infinite amount of applications, which is one of its most appealing features.

Scikit Learn

Scikit Learn is an open-source library for machine learning algorithms that runs on the Python environment. It can be used with both supervised and unsupervised learning algorithms. The library includes popular algorithms as well as the NumPy, Matplotlib, and SciPy packages. Scikit learns most well-known use is for music suggestions in Spotify. Let us now deep dive into some of the key features of Scikit Learn:

- Cross-Validation: There are several methods for checking the accuracy of supervised models on unseen data with Scikit Learn for example the **train_test_split** method, **cross_val_score,** etc.
- Unsupervised learning techniques: There is a wide range of unsupervised learning algorithms available, ranging from clustering, factor analysis, principal component analysis, and unsupervised neuralnetworks.
- Feature extraction: Extracting features from photos and text is a useful tool (e.g. Bag of words) Scikit Learn

Scikit Learn includes a large number of algorithms and can be used for performing common machine learning and data mining tasks such as dimensionality reduction, classification, regression, clustering, and model selection.

Eli5

The outcomes of machine learning model predictions are frequently inaccurate, and the Eli5 Python machine learning library aids in addressing this difficulty of inaccurate predictions. It is a combination of visualization and debugging all machine learning models, as well as tracking all of an algorithm's working processes. Some of the libraries supported by Eli5 include XGBoost, lightning, scikit-learn, and sklearn crfsuite.

Let us now talk about some of Eli5's applications:

- ELI5 is a python package that is used to inspect Machine Learning classifiers and explain their predictions. It is popularly used to debug algorithms such as sklearn regressors and classifiers, XGBoost, CatBoost, Keras, etc.
- Where there are dependencies with other Python packages, Eli5 is essential.
- Eli5 is also used in a variety of industries where legacy software and innovative approaches are being implemented.

SciPy

Scipy is a free, open-source Python library used for scientific computing, data processing, and high-performance computing. The library contains a huge number of user-friendly routines for quick computation. The package is based on the NumPy extension, which allows for data processing and visualization as well as high-level commands. Scipy is used for mathematical computations alongside NumPy. NumPy enables the sorting and indexing of array data, while SciPy stores the numerical code. Cluster, constants, fftpack, integrate, interpolate, io, linalg, ndimage, odr, optimize, signal, sparse, spatial, special, and stats are only a few of the many subpackages available in SciPy. "from scipy import subpackage-name" can be used to import them from SciPy. NumPy, SciPy library, Matplotlib, IPython, Sympy, and Pandas are, however, the essential packages of SciPy.

Features:

- SciPy's key characteristic is that it was written in NumPy, and its array makes extensive use of NumPy.
- SciPy uses its specialised submodules to provide all of the efficient numerical algorithms such as optimization, numerical integration, and many others.
- All functions in SciPy's submodules are extensively documented. SciPy's primary data structure is NumPy arrays, and it includes modules for a variety of popular scientific programming applications. SciPyhandles tasks like linear algebra, integration (calculus), solving ordinary differential equations, and signal processing with ease.

PyTorch

PyTorch is a Python library introduced first by Facebook in the year 2017 that combines the following two high-level capabilities:

- Tensor computation with substantial GPU acceleration (similar to NumPy)
- Platforms based on deep neural networks offering flexibility and speed.

Features:

- Python and its libraries are supported by PyTorch.
- Facebook's Deep Learning requirements necessitated the use of this technology.
- It provides an easy to use API that improves usability and comprehension.
- Graphs can be set up dynamically and computed dynamically at any point during code execution in PyTorch.
- In PyTorch, coding is simple and processing is quick.
- Because CUDA (CUDA is a parallel computing platform and application programming interface that allows software to use certain types of graphics processing unit for general purpose processing – an approach called general-purpose computing on GPUs) is supported, it can be run on GPU machines.

PyTorch is mostly used for natural language processing applications. It was developed primarily by Facebook's artificial-intelligence research lab, and Uber's "Pyro" probabilistic programming software is based on it. PyTorch outperforms TensorFlow in a variety of areas, and it has recently gained a lot of attention due to its features.

LightGBM

Gradient Boosting is a prominent machine learning package that assists developers in developing new algorithms by redefining simple models, such as decision trees. As a result, there exist dedicated libraries that may be used to implement this Gradient Boosting method quickly and efficiently. LightGBM, XGBoost, and CatBoost are the libraries in question. All of these libraries are rivals that help solve the same problem and may be used in nearly the same way.

Features:

1. High production efficiency is ensured by very quick computation.
2. It is user-friendly because it is intuitive.
3. Many other deep learning libraries take longer to train than this one.
4. When using NaN values and other canonical values, there will be no errors.

These libraries offer highly scalable, efficient, and quick gradient boosting implementations, making them popular among machine learning engineers.

Theano

Theano, like other mathematical operations libraries, allows users to define, optimize, and evaluate mathematical expressions. For efficient mathematical processing, it uses massive multi-dimensional arrays. When dealing with large amounts of data, standard C-based codes become slower. Theano, on the other hand, makes it possible to quickly implement code because of its rich library. Unstable expressions can be recognized and computed, making the library more useful over NumPy.

Features:

- NumPy integration: Theano has the ability to use NumPy arrays entirely in Theano compiled functions.
- Use of a GPU that is transparent: It can be used to perform data intensive operations significantly faster than with a CPU.

- Efficient symbolic differentiation: Theano does derivatives for functions with one or more inputs using efficient symbolic differentiation.
- Optimizations for speed and stability: For problems like getting the correct solution for log(1+x) even when x is very small, Theano works really well. This is just one of the many pieces of evidence of Theano's stability.
- C code generation that is dynamic: Theano can evaluate expressions really fast resulting in a significant increase in efficiency.
- Extensive unit-testing and self-verification: Theano can help detect and diagnose numerous types of problems and ambiguities in the model with extensive unit testing and self verification.

Theano expressions use a symbolic syntax, which might be confusing for newcomers who are used to traditional program development. In particular, expressions are specified in an abstract sense, compiled, and then used to do calculations. It is designed to handle the types of processing required by Deep Learning's huge neural network algorithms. It is an industry-standard for Deep Learning research and development and was one of the first libraries of its sort. As of today, Theano is the backbone of a slew of neural network projects, and its popularity is only growing.

Viva questions

Result:

Thus the introduction about Linux is studied.

Ex No: 2.IMPLEMENT A CLASSIFIER

Aim:

To study and implement a classifier

Introduction:

A classifier in machine learning is **an algorithm that automatically orders or categorizes data into one or more of a set of "classes**."

What is classification?

Classification is the process of predicting the class of given data points. Classes are sometimes called as targets/ labels or categories. Classification predictive modeling is the task of approximating a mapping function (f) from input variables (X) to discrete output variables (y).

For example, spam detection in email service providers can be identified as a classification problem. This is s binary classification since there are only 2 classes as spam and not spam. A classifier utilizes some training data to understand how given input variables relate to the class. In this case, known spam and non-spam emails have to be used as the training data. When the classifier is trained accurately, it can be used to detect an unknown email.Classification belongs to the category of supervised learning where the targets also provided with the inputdata. There are many applications in classification in many domains such as in credit approval, medical diagnosis, target marketing etc.

There are two types of learners in classification as lazy learners and eager learners.

1. *Lazy learners*

Lazy learners simply store the training data and wait until a testing data appear. When it does, classification is conducted based on the most related data in the stored training data. Compared to eager learners, lazy learners have less training time but more time in predicting.

Ex. k-nearest neighbor, Case-based reasoning

2. *Eager learners*

Eager learners construct a classification model based on the given training data before receiving data for classification. It must be able to commit to a single hypothesis that covers the entire instance space. Due to

the model construction, eager learners take a long time for train and less time to predict.

Ex. Decision Tree, Naive Bayes, Artificial Neural Networks

Classification algorithms

There is a lot of classification algorithms available now but it is not possible to conclude which one is superior to other. It depends on the application and nature of available data set. For example, if the classes are linearly separable, the linear classifiers like Logistic regression, Fisher's linear discriminant can outperform sophisticated models and vice versa.

- Logistic Regression.
- Naive Bayes.
- K-Nearest Neighbors.
- Decision Tree.
- Support Vector Machines.

Algorithm Selection

1. Read the data.
2. Create dependent and independent data sets based on our dependent and independent features.
3. Split the data into training and testing sets.
4. Train the model using different algorithms such as KNN, Decision tree, SVM, etc.
5. Evaluate the classifier.
6. Choose the classifier with the most accuracy.

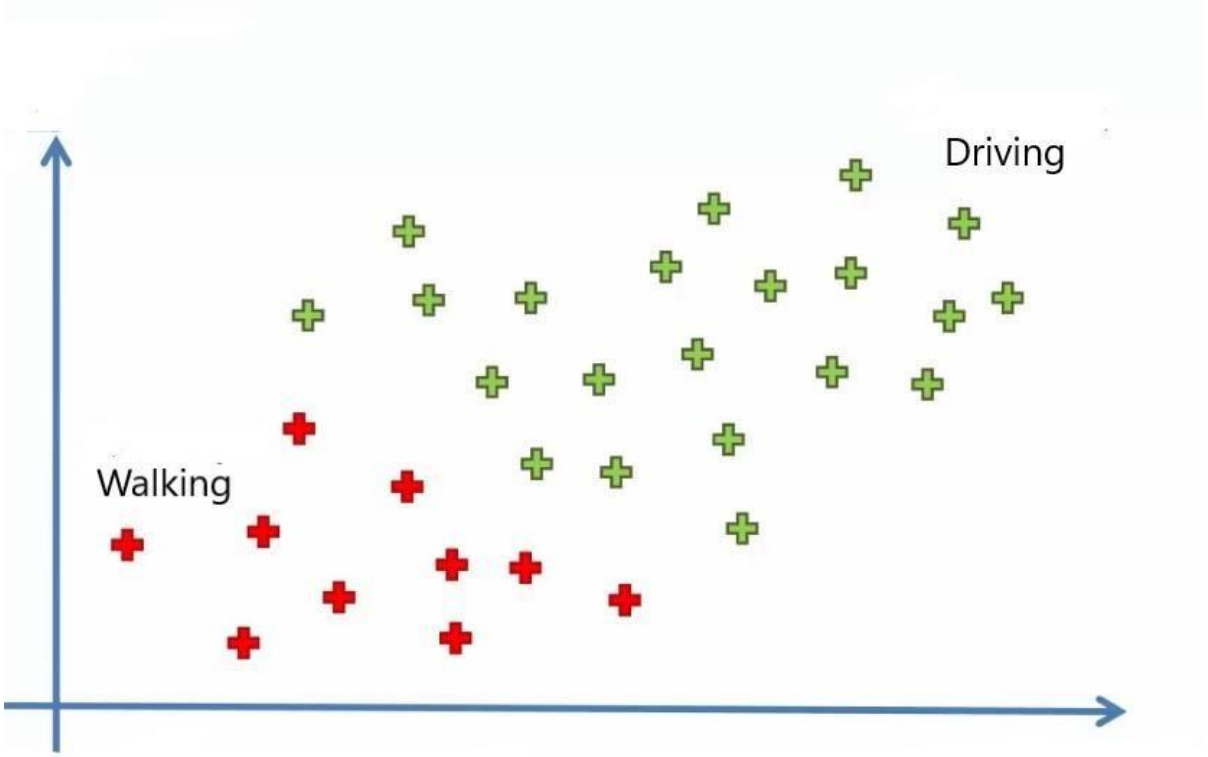
Introduction to Naive Bayes algorithm

Naive Bayes is a classification algorithm that works based on the Bayes theorem. Before explaining about Naive Bayes, first, we should discuss Bayes Theorem. Bayes theorem is used to find the probability of a hypothesis with given evidence.

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

In this, using Bayes theorem we can find the probability of A, given that B occurred. A is the hypothesis and B is the evidence.

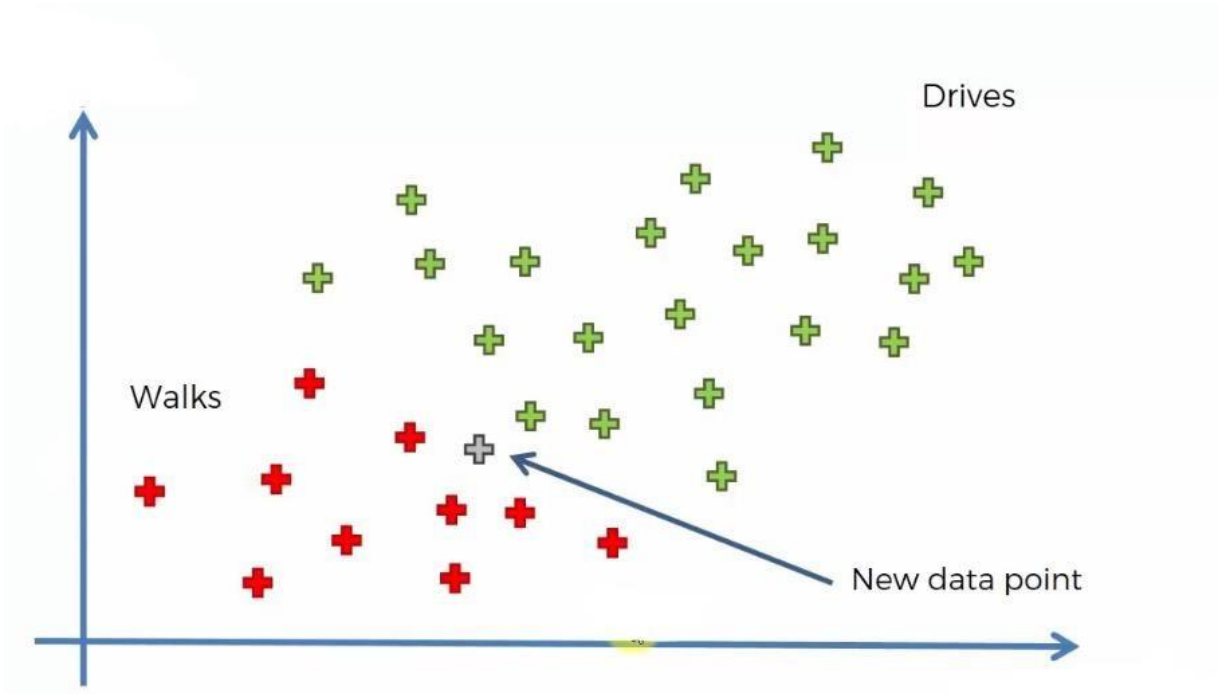
P(B|A) is the probability of B given that A is True.



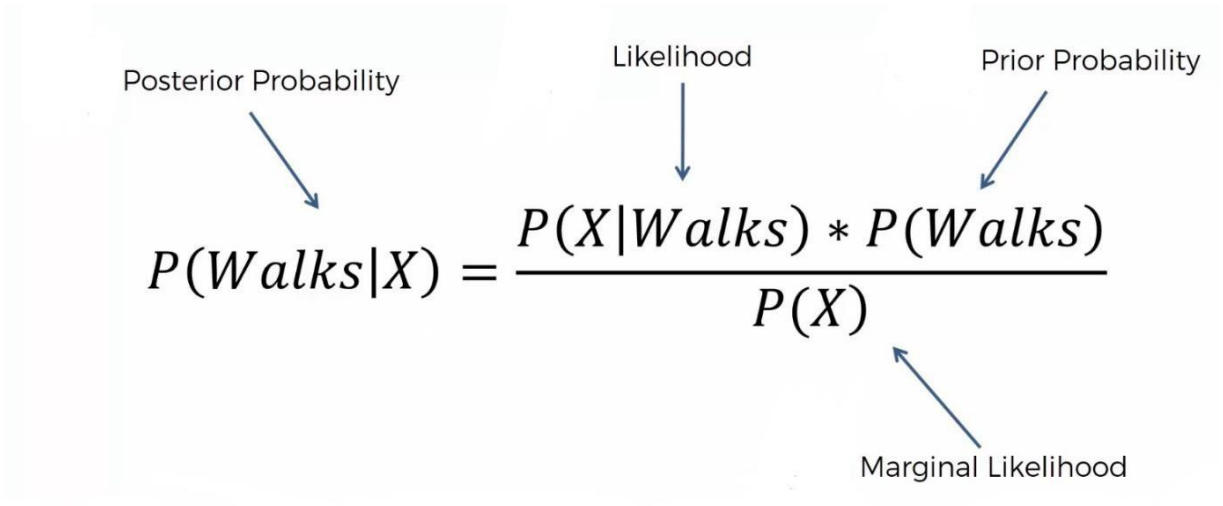
P(A) and P(B) is the independent probabilities of A and B.

The concept behind the algorithm

Let's understand the concept of the Naive Bayes Theorem through an example. We are taking a dataset of employees in a company, our aim is to create a model to find whether a person is going to the office by driving or walking using salary and age of the person.

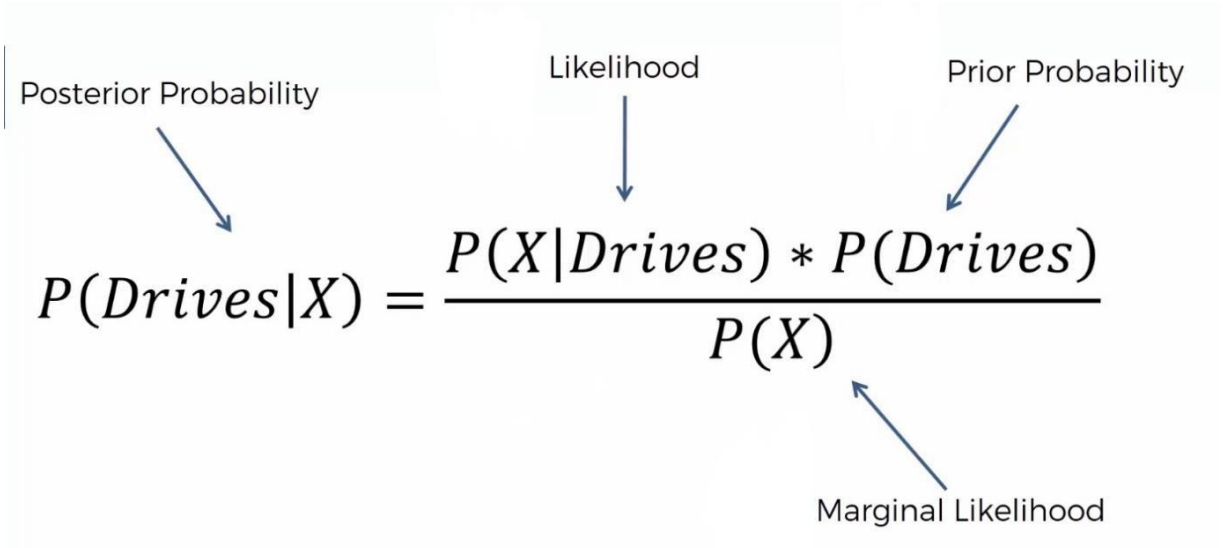


In the above, we can see 30 data points in which red points belong to those who are walking and green belongs to those who are driving. Now let's add a new data point into it. Our aim is to find the category that the new point belongs to.



Note that we are taken age on the X-axis and Salary on the Y-axis. We are using the Naive Bayes algorithm to find the category of the new data point. For this, we have to find the posterior probability of walking and driving for this data point. After comparing, the point belongs to the category having a higher probability.

The posterior probability of walking for the new data point is :



also for the driving is :

Steps involved in Naive Bayes algorithm

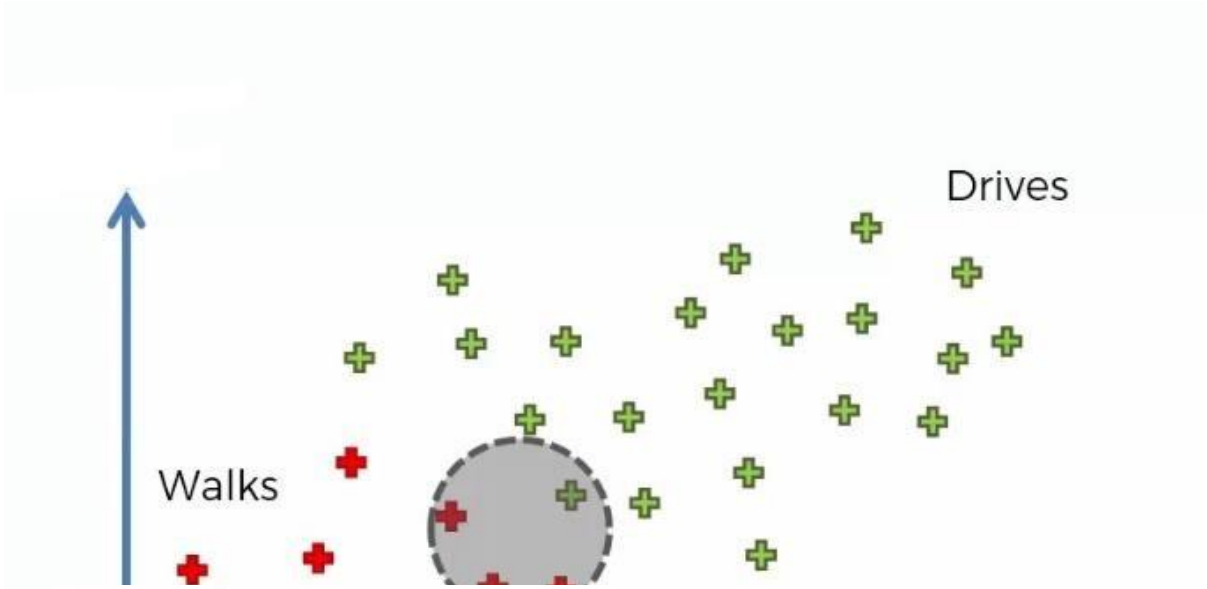
Step 1: We have to find all the probabilities required for the Bayes theorem for the calculation of posterior probability

P(Walks) is simply the probability of those who walk among all

$$P(Walks) = \frac{Number\ of\ Walkers}{Total\ Observations}$$

$$P(Walks) = \frac{10}{30}$$

In order to find the marginal likelihood, P(X), we have to consider a circle around the new data point of any radii including some red and green points.



$$P(X) = \frac{Number\ of\ Similar\ Observations}{Total\ Observations}$$

$$P(X) = \frac{4}{30}$$

P(X|Walks) can be found by :

$$P(X|Walks) = \frac{Number\ of\ Similar\ Observations\ Among\ those\ who\ Walk}{Total\ number\ of\ Walkers}$$

$$P(X|Walks) = \frac{3}{10}$$

Now we can find the posterior probability using the Bayes theorem,

$$P(Walks|X) = \frac{\frac{3}{10} * \frac{10}{30}}{\frac{4}{30}} = 0.75$$

step 2: Similarly we can find the posterior probability of Driving, and it is 0.25

Step 3: Compare both posterior probabilities. When comparing the posterior probability, we can find that P(walks|X) has greater values and the new point belongs to the walking category.

Worked Example of Naive Bayes

In this section, we will make the Naive Bayes calculation concrete with a small example on a machine learning dataset.

We can generate a small contrived binary (2 class) classification problem using the make_blobs() function from the scikit-learn API.
The example below generates 100 examples with two numerical input variables, each assigned one of two classes.

```
# example of generating a small classification dataset
from sklearn.datasets import make_blobs
# generate 2d classification dataset
X, y = make_blobs(n_samples=100, centers=2, n_features=2, random_state=1)
# summarize
print(X.shape, y.shape)
print(X[:5])
print(y[:5])
```

Running the example generates the dataset and summarizes the size, confirming the dataset was generated as expected.

The "*random_state*" argument is set to 1, ensuring that the same random sample of observations is generated each time the code is run.
The input and output elements of the first five examples are also printed, showing that indeed, the two input variables are numeric and the class labels are either 0 or 1 for each example.

```
(100, 2) (100,)
[[-10.6105446    4.11045368]
 [  9.05798365  0.99701708]
 [  8.705727     1.36332954]
 [ -8.29324753  2.35371596]
 [  6.5954554    2.4247682 ]]
[0 1 1 0 1]
```

We will model the numerical input variables using a Gaussian probability distribution.

This can be achieved using the norm SciPy API. First, the distribution can be constructed by specifying the parameters of the distribution, e.g. the mean and standard deviation, then the probability density function can be sampled for specific values using the norm.pdf() function.
We can estimate the parameters of the distribution from the dataset using the *mean()* and *std()* NumPy functions.
The *fit_distribution()* function below takes a sample of data for one variable and fits a data distribution.

```
# fit a probability distribution to a univariate data sample
def fit_distribution(data):
	# estimate parameters
	mu = mean(data)
	sigma = std(data)
	print(mu, sigma)
	# fit distribution
	dist = norm(mu, sigma)
	return dist
```

Recall that we are interested in the conditional probability of each input variable. This means we need one distribution for each of the input variables, and one set of distributions for each of the class labels, or four distributions in total.

First, we must split the data into groups of samples for each of the class labels.

```
...
# sort data into classes
Xy0 = X[y == 0]
Xy1 = X[y == 1]
print(Xy0.shape, Xy1.shape)
```

We can then use these groups to calculate the prior probabilities for a data sample belonging to each group.

This will be 50% exactly given that we have created the same number of examples in each of the two classes; nevertheless, we will calculate these priors for completeness.

```
...
# calculate priors
priory0 = len(Xy0) / len(X)
priory1 = len(Xy1) / len(X)
print(priory0, priory1)
```

Finally, we can call the *fit_distribution()* function that we defined to prepare a probability distribution for each variable, for each class label.

```
...
# create PDFs for y==0
```

```
X1y0 = fit_distribution(Xy0[:, 0])
X2y0 = fit_distribution(Xy0[:, 1])
# create PDFs for y==1
X1y1 = fit_distribution(Xy1[:, 0])
X2y1 = fit_distribution(Xy1[:, 1])
```

Tying this all together, the complete probabilistic model of the dataset is listed below.

```
# summarize probability distributions of the dataset
from sklearn.datasets import make_blobs
from scipy.stats import norm
from numpy import mean
from numpy import std

# fit a probability distribution to a univariate data sample
def fit_distribution(data):
        # estimate parameters
        mu = mean(data)
        sigma = std(data)
        print(mu, sigma)
        # fit distribution
        dist = norm(mu, sigma)
        return dist

# generate 2d classification dataset
X, y = make_blobs(n_samples=100, centers=2, n_features=2, random_state=1)
# sort data into classes
Xy0 = X[y == 0]
Xy1 = X[y == 1]
print(Xy0.shape, Xy1.shape)
# calculate priors
priory0 = len(Xy0) / len(X)
priory1 = len(Xy1) / len(X)
```

```
print(priory0, priory1)
# create PDFs for y==0
X1y0 = fit_distribution(Xy0[:, 0])
X2y0 = fit_distribution(Xy0[:, 1])
# create PDFs for y==1
X1y1 = fit_distribution(Xy1[:, 0])
X2y1 = fit_distribution(Xy1[:, 1])
```

Running the example first splits the dataset into two groups for the two class labels and confirms the size of each group is even and the priors are 50%.

Probability distributions are then prepared for each variable for each class label and the mean and standard deviation parameters of each distribution are reported, confirming that the distributions differ.

```
(50, 2) (50, 2)
0.5 0.5
-1.5632888906409914 0.787444265443213
4.426680361487157 0.958296071258367
-9.681177100524485 0.8943078901048118
-3.9713794295185845 0.9308177595208521
```

Next, we can use the prepared probabilistic model to make a prediction.

The independent conditional probability for each class label can be calculated using the prior for the class (50%) and the conditional probability of the value for each variable.

The *probability()* function below performs this calculation for one input example (array of two values) given the prior and conditional probability distribution for each variable. The value returned is a score rather than a probability as the quantity is not normalized, a simplification often performed when implementing naive bayes.

```
# calculate the independent conditional probability
def probability(X, prior, dist1, dist2):
	return prior * dist1.pdf(X[0]) * dist2.pdf(X[1])
```

We can use this function to calculate the probability for an example belonging to each class.

First, we can select an example to be classified; in this case, the first example in the dataset.

```
...
```

```
# classify one example
Xsample, ysample = X[0], y[0]
```

Next, we can calculate the score of the example belonging to the first class, then the second class, then report the results.

```
...
py0 = probability(Xsample, priory0, distX1y0, distX2y0)
py1 = probability(Xsample, priory1, distX1y1, distX2y1)
print('P(y=0 | %s) = %.3f' % (Xsample, py0*100))
print('P(y=1 | %s) = %.3f' % (Xsample, py1*100))
```

The class with the largest score will be the resulting classification.

Tying this together, the complete example of fitting the Naive Bayes model and using it to make one prediction is listed below.

```
# example of preparing and making a prediction with a naive bayes model
from sklearn.datasets import make_blobs
from scipy.stats import norm
from numpy import mean
from numpy import std

# fit a probability distribution to a univariate data sample
def fit_distribution(data):
        # estimate parameters
        mu = mean(data)
        sigma = std(data)
        print(mu, sigma)
        # fit distribution
        dist = norm(mu, sigma)
        return dist

# calculate the independent conditional probability
def probability(X, prior, dist1, dist2):
```

```
        return prior * dist1.pdf(X[0]) * dist2.pdf(X[1])

# generate 2d classification dataset
X, y = make_blobs(n_samples=100, centers=2, n_features=2, random_state=1)
# sort data into classes
Xy0 = X[y == 0]
Xy1 = X[y == 1]
# calculate priors
priory0 = len(Xy0) / len(X)
priory1 = len(Xy1) / len(X)
# create PDFs for y==0
distX1y0 = fit_distribution(Xy0[:, 0])
distX2y0 = fit_distribution(Xy0[:, 1])
# create PDFs for y==1
distX1y1 = fit_distribution(Xy1[:, 0])
distX2y1 = fit_distribution(Xy1[:, 1])
# classify one example
Xsample, ysample = X[0], y[0]
py0 = probability(Xsample, priory0, distX1y0, distX2y0)
py1 = probability(Xsample, priory1, distX1y1, distX2y1)
print('P(y=0 | %s) = %.3f' % (Xsample, py0*100))
print('P(y=1 | %s) = %.3f' % (Xsample, py1*100))
print('Truth: y=%d' % ysample)
```

Running the example first prepares the prior and conditional probabilities as before, then uses them to make a prediction for one example.

The score of the example belonging to *y=0* is about 0.3 (recall this is an unnormalized probability), whereas the score of the example belonging to *y=1* is 0.0. Therefore, we would classify the example as belonging to *y=0*.

In this case, the true or actual outcome is known, *y=0*, which matches the prediction by our Naive Bayes model.

```
P(y=0 | [-0.79415228  2.10495117]) = 0.348
P(y=1 | [-0.79415228  2.10495117]) = 0.000
```

```
Truth: y=0
```

In practice, it is a good idea to use optimized implementations of the Naive Bayes algorithm. The scikit-learn library provides three implementations, one for each of the three main probability distributions; for example, BernoulliNB, MultinomialNB, and GaussianNB for binomial, multinomial and Gaussian distributed input variables respectively.

To use a scikit-learn Naive Bayes model, first the model is defined, then it is fit on the training dataset. Once fit, probabilities can be predicted via the *predict_proba()* function and class labels can be predicted directly via the *predict()* function.

The complete example of fitting a Gaussian Naive Bayes model (GaussianNB) to the same test dataset is listed below.

```
# example of gaussian naive bayes
from sklearn.datasets import make_blobs
from sklearn.naive_bayes import GaussianNB
# generate 2d classification dataset
X, y = make_blobs(n_samples=100, centers=2, n_features=2, random_state=1)
# define the model
model = GaussianNB()
# fit the model
model.fit(X, y)
# select a single sample
Xsample, ysample = [X[0]], y[0]
# make a probabilistic prediction
yhat_prob = model.predict_proba(Xsample)
print('Predicted Probabilities: ', yhat_prob)
# make a classification prediction
yhat_class = model.predict(Xsample)
print('Predicted Class: ', yhat_class)
print('Truth: y=%d' % ysample)
```

Running the example fits the model on the training dataset, then makes predictions for the same first example that we used in the prior example.

In this case, the probability of the example belonging to *y=0* is 1.0 or a certainty. The probability of *y=1* is a very small value close to 0.0.

Finally, the class label is predicted directly, again matching the ground truth for the example.

```
Predicted Probabilities: [[1.00000000e+00 5.52387327e-30]]
Predicted Class: [0]
Truth: y=0
```

5 Tips When Using Naive Bayes

This section lists some practical tips when working with Naive Bayes models.

1. *Use a KDE for Complex Distributions*

If the probability distribution for a variable is complex or unknown, it can be a good idea to use a kernel density estimator or KDE to approximate the distribution directly from the data samples.

A good example would be the Gaussian KDE.

2. *Decreased Performance With Increasing Variable Dependence*

By definition, Naive Bayes assumes the input variables are independent of each other.

This works well most of the time, even when some or most of the variables are in fact dependent. Nevertheless, the performance of the algorithm degrades the more dependent the input variables happen to be.

3. *Avoid Numerical Underflow with Log*

The calculation of the independent conditional probability for one example for one class label involves multiplying many probabilities together, one for the class and one for each input variable. As such, the multiplication of many small numbers together can become numerically unstable, especially as the number of input variables increases.

To overcome this problem, it is common to change the calculation from the product of probabilities to the sum of log probabilities. For example:

- P(yi | x1, x2, …, xn) = log(P(x1|y1)) + log(P(x2|y1)) + … log(P(xn|y1)) + log(P(yi))

Calculating the natural logarithm of probabilities has the effect of creating larger (negative) numbers and adding the numbers together will mean that larger probabilities will be closer to zero. The resulting values can still be compared and maximized to give the most likely class label.

This is often called the log-trick when multiplying probabilities.

4. *Update Probability Distributions*

As new data becomes available, it can be relatively straightforward to use this new data with the old data to update the estimates of the parameters for each variable's probability distribution.

This allows the model to easily make use of new data or the changing distributions of data over time.

5. *Use as a Generative Model*

The probability distributions will summarize the conditional probability of each input variable value for each class label.

These probability distributions can be useful more generally beyond use in a classification model.

For example, the prepared probability distributions can be randomly sampled in order to create new plausible data instances. The conditional independence assumption assumed may mean that the examples are more or less plausible based on how much actual interdependence exists between the input variables in the dataset.

https://www.analyticsvidhya.com/blog/2021/01/a-guide-to-the-naive-bayes-algorithm/

Implementation of Naive Bayes

Now let's implement Naive Bayes using python

We are using the Social network ad dataset. The dataset contains the details of users in a social networking site to find whether a user buys a product by clicking the ad on the site based on their salary, age, and gender.

Index	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0
5	15728773	Male	27	58000	0
6	15598044	Female	27	84000	0
7	15694829	Female	32	150000	1
8	15600575	Male	25	33000	0
9	15727311	Female	35	65000	0
10	15570769	Female	26	80000	0

Let's start the programming by importing essential libraries required

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

Importing of dataset

```
dataset = pd.read_csv('Social_Network_Ads.csv')
X = dataset.iloc[:, [1, 2, 3]].values
y = dataset.iloc[:, -1].values
```

Since our dataset containing character variables we have to encode it using LabelEncoder

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
X[:,0] = le.fit_transform(X[:,0])
```

Train test Split

We are performing a train test split on our dataset. We are providing the test size as 0.20, that means our training sample contains 320 training set and test sample contains 80 test set

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state = 0)
```

Feature scaling

Next, we are doing **feature scaling** to the training and test set of independent variables

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Training the Naive Bayes model on the training set

```
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
classifier.fit(X_train, y_train)
```

Let’s predict the test results

```
y_pred  =  classifier.predict(X_test)
```

Predicted and actual value –

y_pred

	0
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	1
8	0

y_test

	0
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	1
8	0

For the first 8 values, both are the same. We can evaluate our matrix using the confusion matrix and accuracy score by comparing the predicted and actual test values

```
from sklearn.metrics import confusion_matrix,accuracy_score
cm = confusion_matrix(y_test, y_pred)
ac = accuracy_score(y_test,y_pred)
```

confusion matrix –

ac – **0.9125**

	0	1
0	55	3
1	4	18

Accuracy is good. Note that, you can achieve better results for this problem using different algorithms.

Full Code

```
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the dataset
dataset = pd.read_csv('Social_Network_Ads.csv')
X = dataset.iloc[:, [2, 3]].values
y = dataset.iloc[:, -1].values

# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state = 0)

# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
```

```
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Training the Naive Bayes model on the Training set
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
classifier.fit(X_train, y_train)

# Predicting the Test set results
y_pred = classifier.predict(X_test)

# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix, accuracy_score
ac = accuracy_score(y_test,y_pred)
cm = confusion_matrix(y_test, y_pred)
```

Result:

Thus the classifier is implemented.

Ex No: 3. DEVELOP A PREDICTIVE MODEL

Aim:

To develop a predictive model.

Introduction:

Let's dive into a real-life analogy to understand this concept further. A student named X wants to choose a course after his 10+2, and he is confused about the choice of course based on his skill set. So he decides to consult various people like his cousins, teachers, parents, degree students, and working people. He asks them varied questions like why he should choose, job opportunities with that course, course fee, etc. Finally, after consulting various people about the course he decides to take the course suggested by most of the people.

Working of Random Forest Algorithm

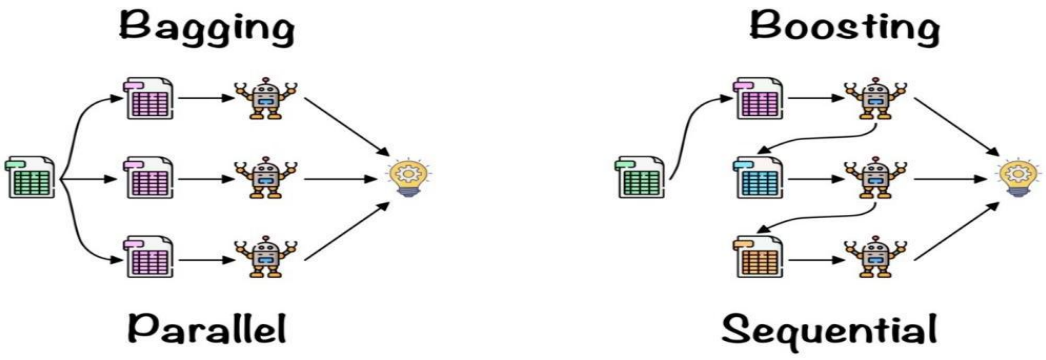


Before understanding the working of the random forest we must look into the ensemble technique. ***Ensemble*** simply means combining multiple models. Thus a collection of models is used to make predictions rather than an individual model.

Ensemble uses two types of methods:

1. **Bagging**– It creates a different training subset from sample training data with replacement & the final output is based on majority voting. For example, Random Forest.

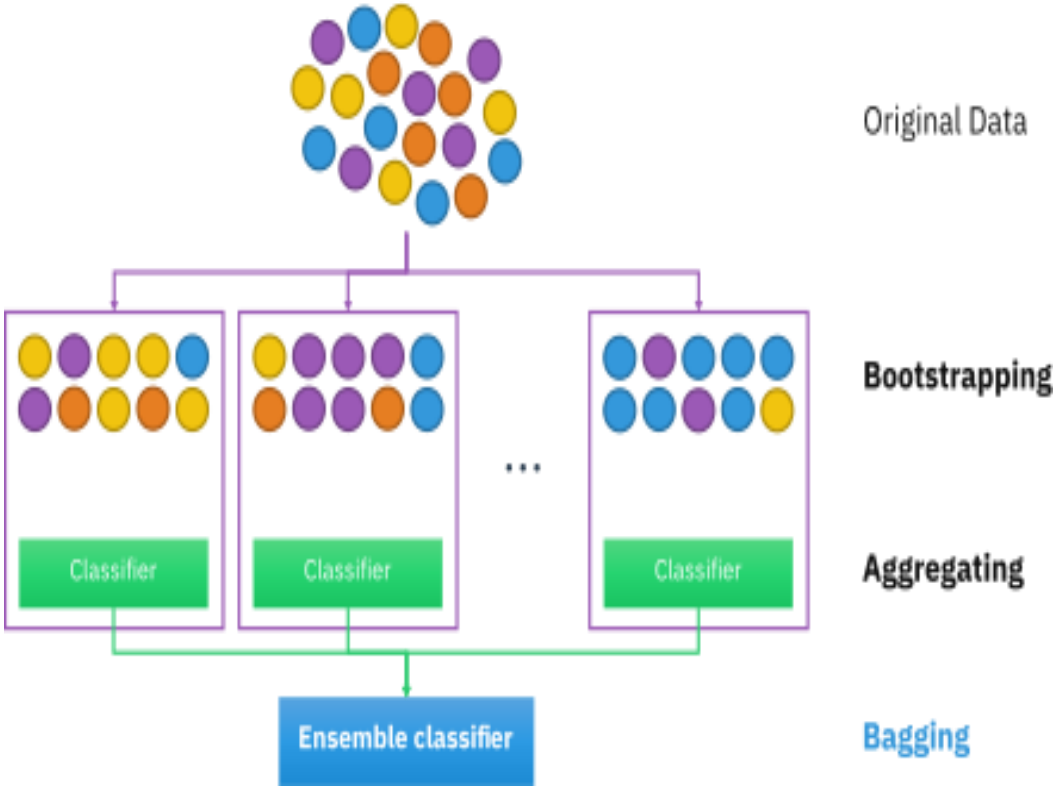
2. **Boosting**– It combines weak learners into strong learners by creating sequential models such that the final model has the highest accuracy. For example, ADA BOOST, XG BOOST



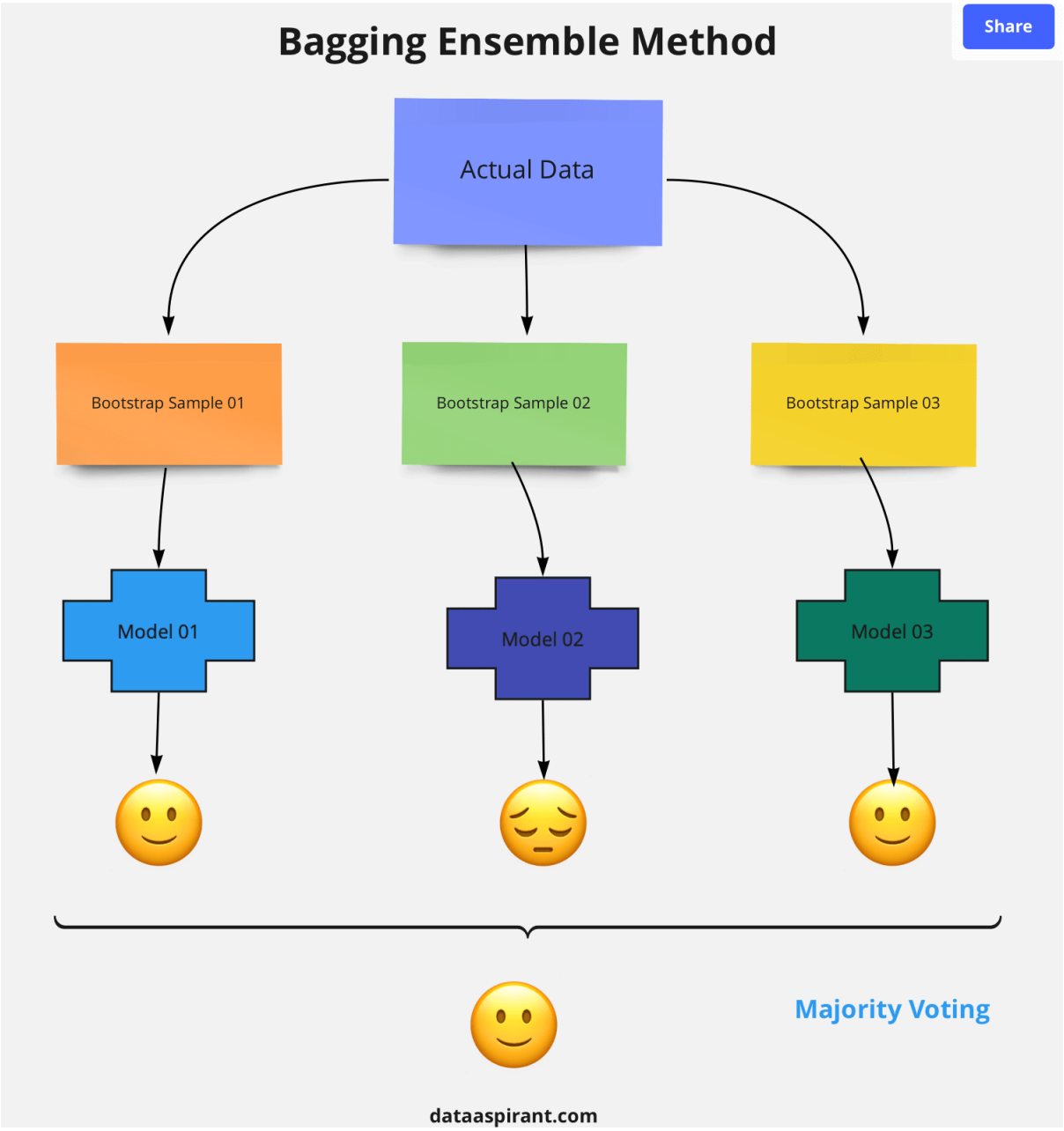
As mentioned earlier, Random forest works on the Bagging principle. Now let's dive in and understand bagging in detail.

Bagging

Bagging, also known as ***Bootstrap Aggregation*** is the ensemble technique used by random forest. Bagging chooses a random sample from the data set. Hence each model is generated from the samples (Bootstrap Samples) provided by the Original Data with replacement known as ***row sampling***. This step of row sampling with replacement is called ***bootstrap***. Now each model is trained independently which generates results. The final output is based on majority voting after combining the results of all models. This step which involves combining all the results and generating output based on majority voting is known as ***aggregation***.



Now let's look at an example by breaking it down with the help of the following figure. Here the bootstrap sample is taken from actual data (Bootstrap sample 01, Bootstrap sample 02, and Bootstrap sample 03) with a replacement which means there is a high possibility that each sample won't contain unique data. Now the model (Model 01, Model 02, and Model 03) obtained from this bootstrap sample is trained independently. Each model generates results as shown. Now Happy emoji is having a majority when compared to sad emoji. Thus based on majority voting final output is obtained as Happy emoji.



Steps involved in random forest algorithm:

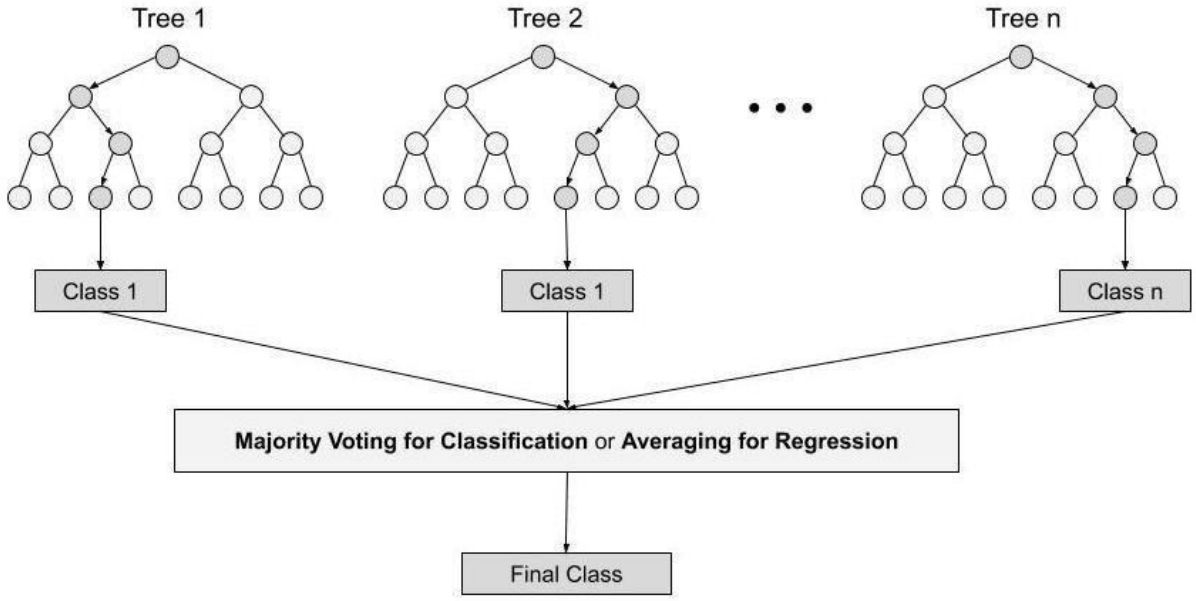
Step 1: In Random forest n number of random records are taken from the data set having k number of records.

Step 2: Individual decision trees are constructed for each sample.

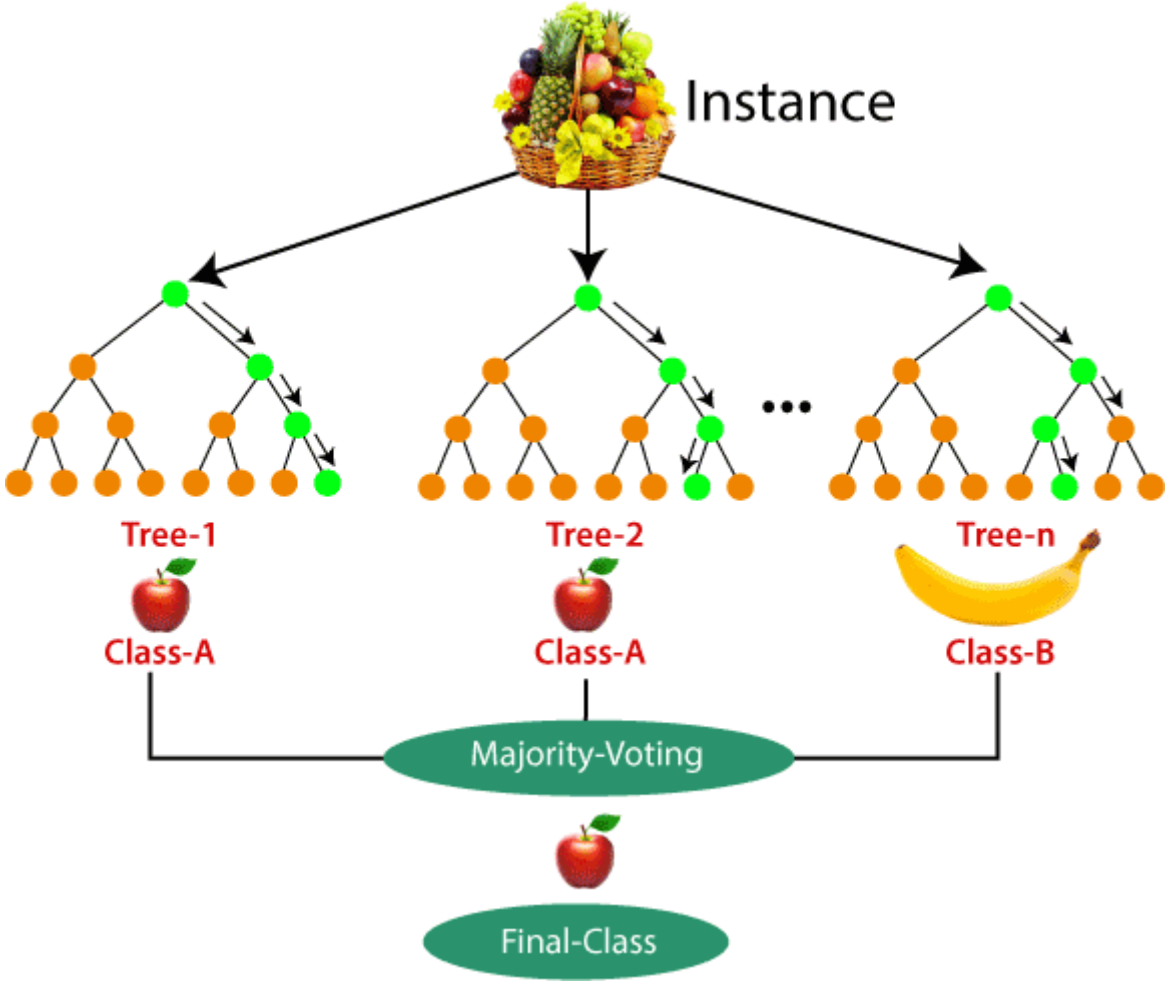
Step 3: Each decision tree will generate an output.

Step 4: Final output is considered based on ***Majority Voting or Averaging*** for Classification andregression respectively.

For example: consider the fruit basket as the data as shown in the figure below. Now n number of samples



are taken from the fruit basket and an individual decision tree is constructed for each sample. Each decision tree will generate an output as shown in the figure. The final output is considered based on majority voting. In the below figure you can see that the majority decision tree gives output as an apple when compared to a banana, so the final output is taken as an apple.



Important Features of Random Forest

1. Diversity- Not all attributes/variables/features are considered while making an individual tree, each tree is different.

2. Immune to the curse of dimensionality- Since each tree does not consider all the features, the feature space is reduced.

3. Parallelization-Each tree is created independently out of different data and attributes. This means that we can make full use of the CPU to build random forests.

4. Train-Test split- In a random forest we don't have to segregate the data for train and test as there will always be 30% of the data which is not seen by the decision tree.

5. Stability- Stability arises because the result is based on majority voting/ averaging.

Difference Between Decision Tree & Random Forest

Random forest is a collection of decision trees; still, there are a lot of differences in their behavior.

Decision trees	**Random Forest**
1. Decision trees normally suffer from theproblem of overfitting if it's allowed to growwithout any control.	1. Random forests are created from subsets of data and the final output is based on average or majority ranking and hence the problem of overfitting is taken care of.
2. A single decision tree is faster in computation.	2. It is comparatively slower.
3. When a data set with features is taken as input by a decision tree it will formulate some set of rules to do prediction.	3. Random forest randomly selects observations, builds a decision tree and the average result is taken. It doesn't use any set of formulas.

Thus random forests are much more successful than decision trees only if the trees are diverse and acceptable.

Important Hyperparameters

Hyperparameters are used in random forests to either enhance the performance and predictive power of models or to make the model faster.

Following hyperparameters increases the predictive power:

1. **n_estimators**– number of trees the algorithm builds before averaging the predictions.

2. ***max_features***– maximum number of features random forest considers splitting a node.

3. ***mini_sample_leaf***– determines the minimum number of leaves required to split an internal node.

Following hyperparameters increases the speed:

1. ***n_jobs***– it tells the engine how many processors it is allowed to use. If the value is 1, it can use only one processor but if the value is -1 there is no limit.
2. ***random_state***– controls randomness of the sample. The model will always produce the same results if it has a definite value of random state and if it has been given the same hyperparameters and the same training data.

3. ***oob_score*** *– OOB* means out of the bag. It is a random forest cross-validation method. In this one-third of the sample is not used to train the data instead used to evaluate its performance. These samples are called out of bag samples.

Coding in python – Random Forest

Now let's understand Random Forest with the help of code.

1. Let's import the libraries.

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

2. import the dataset.

```
# Reading the csv file and putting it into 'df' object
df = pd.read_csv('heart_v2.csv')
df.head()
```

	age	sex	BP	cholestrol	heart disease
0	70	1	130	322	1
1	67	0	115	564	0
2	57	1	124	261	1
3	64	1	128	263	0
4	74	0	120	269	0

3. Putting Feature Variable to X and Target variable to y.

```
# Putting response variable to y
y = df['heart disease']
```

```
# Putting feature variable to X
X = df.drop('heart disease',axis=1)
```

4. Train-Test-Split is performed

```
# now lets split the data into train and test
from sklearn.model_selection import train_test_split
# Splitting the data into train and test
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.7, random_state=42)
X_train.shape, X_test.shape
```

```
((189, 4), (81, 4))
```

3. Let's import RandomForestClassifier and fit the data.

```
from sklearn.ensemble import RandomForestClassifier
classifier_rf = RandomForestClassifier(random_state=42, n_jobs=-1, max_depth=5,
                                       n_estimators=100, oob_score=True)
%%time
classifier_rf.fit(X_train, y_train)
```

```
Wall time: 241 ms
RandomForestClassifier(max_depth=5, n_jobs=-1, oob_score=True, random_state=42)
```

```
# checking the oob score
classifier_rf.oob_score_
```

```
0.656084656084656
```

4. Let's do hyperparameter tuning for Random Forest using GridSearchCV and fit the data.

```
rf = RandomForestClassifier(random_state=42, n_jobs=-1)
params = {
    'max_depth': [2,3,5,10,20],
    'min_samples_leaf': [5,10,20,50,100,200],
    'n_estimators': [10,25,30,50,100,200]
```

```
}
from sklearn.model_selection import GridSearchCV
# Instantiate the grid search model
grid_search = GridSearchCV(estimator=rf,
                           param_grid=params,
                           cv = 4,
                           n_jobs=-1, verbose=1, scoring="accuracy")
%%time
grid_search.fit(X_train, y_train)
```

```
Fitting 4 folds for each of 180 candidates, totalling 720 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:    5.2s
[Parallel(n_jobs=-1)]: Done 192 tasks      | elapsed:   15.6s
[Parallel(n_jobs=-1)]: Done 442 tasks      | elapsed:   32.0s
Wall time: 51.6 s
[Parallel(n_jobs=-1)]: Done 720 out of 720 | elapsed:   51.4s finished
GridSearchCV(cv=4, estimator=RandomForestClassifier(n_jobs=-1, random_state=42),
             n_jobs=-1,
             param_grid={'max_depth': [2, 3, 5, 10, 20],
                         'min_samples_leaf': [5, 10, 20, 50, 100, 200],
                         'n_estimators': [10, 25, 30, 50, 100, 200]},
             scoring='accuracy', verbose=1)
```

```
grid_search.best_score_
```

```
0.6985815602836879
```

```
rf_best = grid_search.best_estimator_
rf_best
```

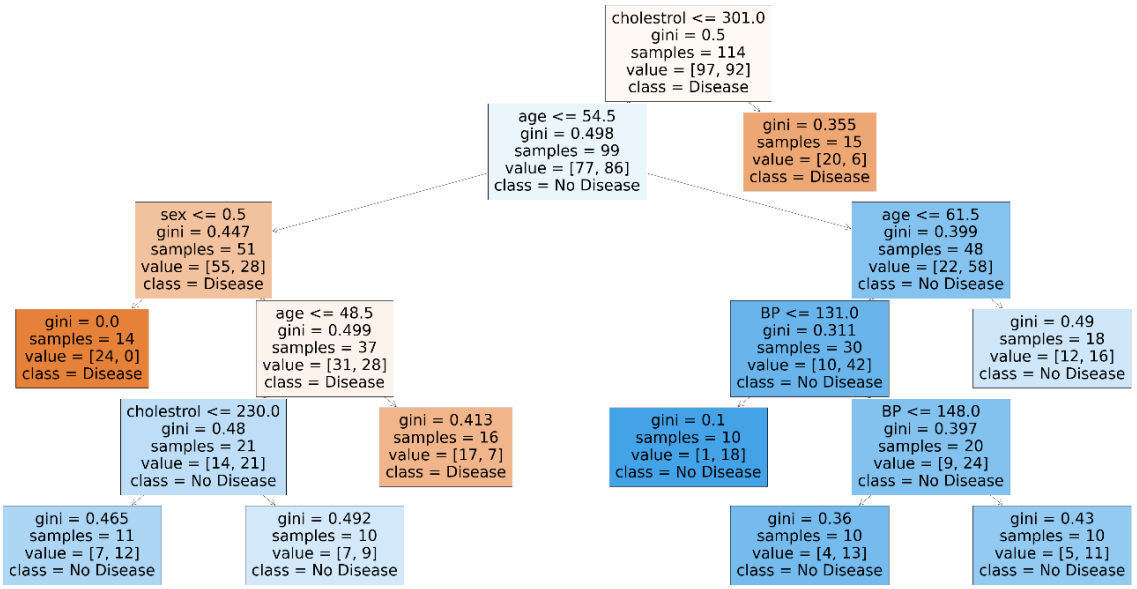
```
RandomForestClassifier(max_depth=5, min_samples_leaf=10, n_estimators=10,
                       n_jobs=-1, random_state=42)
```

From hyperparameter tuning, we can fetch the best estimator as shown. The best set of parameters identified were max_depth=5, min_samples_leaf=10,n_estimators=10

5. Now let's visualize

```
from sklearn.tree import plot_tree
plt.figure(figsize=(80,40))
```

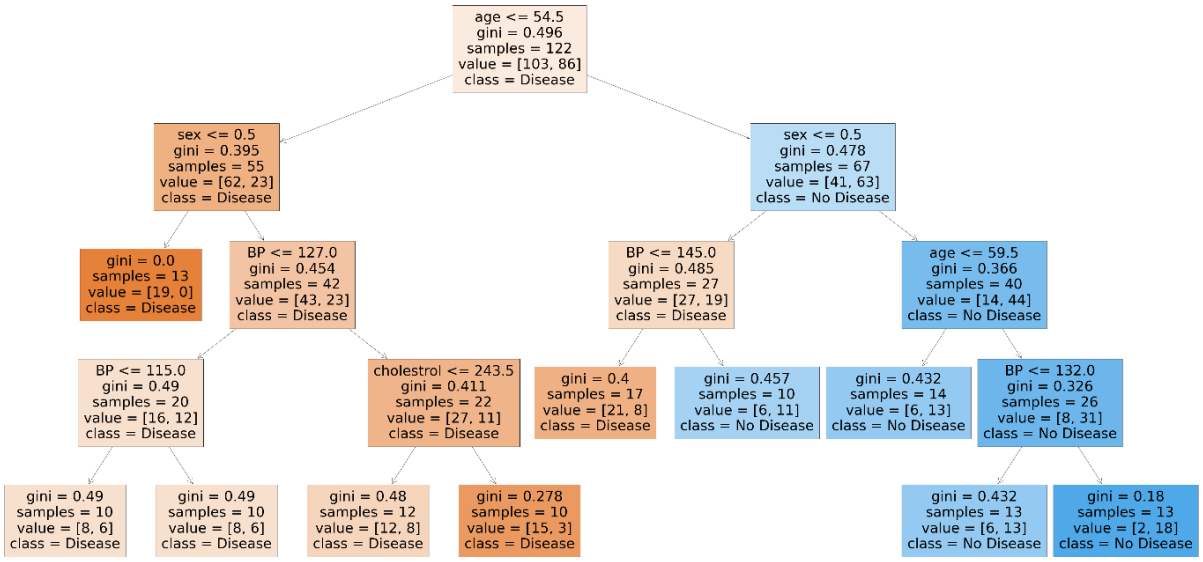
```
plot_tree(rf_best.estimators_[5], feature_names = X.columns,class_names=['Disease', "No Disease"],filled=True);
```




```
from sklearn.tree import plot_tree

plt.figure(figsize=(80,40))

plot_tree(rf_best.estimators_[7], feature_names = X.columns,class_names=['Disease', "No Disease"],filled=True);
```



The trees created by estimators_[5] and estimators_[7] are different. Thus we can say that each tree is independent of the other.

6. Now let's sort the data with the help of feature importance

```
rf_best.feature_importances_
```

```
array([0.46128487, 0.2180848 , 0.13174619, 0.18888413])
```

```
imp_df = pd.DataFrame({
    "Varname": X_train.columns,
    "Imp": rf_best.feature_importances_
})
imp_df.sort_values(by="Imp", ascending=False)
```

	Varname	Imp
0	age	0.461285
1	sex	0.218085
3	cholestrol	0.188884
2	BP	0.131746

Use Cases

This algorithm is widely used in E-commerce, banking, medicine, the stock market, etc.

For example: In the Banking industry it can be used to find which customer will default on the loan.

Advantages and Disadvantages of Random Forest AlgorithmAdvantages

1. It can be used in classification and regression problems.

2. It solves the problem of overfitting as output is based on majority voting or averaging.

3. It performs well even if the data contains null/missing values.

4. Each decision tree created is independent of the other thus it shows the property of parallelization.

5. It is highly stable as the average answers given by a large number of trees are taken.

6. It maintains diversity as all the attributes are not considered while making each decision tree though it is not true in all cases.

7. It is immune to the curse of dimensionality. Since each tree does not consider all the attributes, feature space is reduced.

8. We don't have to segregate data into train and test as there will always be 30% of the data which is not seen by the decision tree made out of bootstrap.

Disadvantages

1. Random forest is highly complex when compared to decision trees where decisions can be made by following the path of the tree.

2. Training time is more compared to other models due to its complexity. Whenever it has to make a prediction each decision tree has to generate output for the given input data.

https://www.analyticsvidhya.com/blog/2021/06/understanding-random-forest/

Result :

Thus the random forest algorithm is implemented using python.

Ex.No. 4.IMPLEMENT A DECISION TREE ALGORITHM

Aim:

To implement a decision tree algorithm.

Introduction



Photo by Tim Foster on Unsplash

If you see, you will find out that today, ensemble learnings are more popular and used by industry and rankers on Kaggle. Bootstrap aggregation, Random forest, gradient boosting, XGboost are all very important and widely used algorithms, to understand them in detail one needs to know the decision tree in depth. In this article, we are going to cover just that. Without any further due, let's just dive right into it.

Table of Content

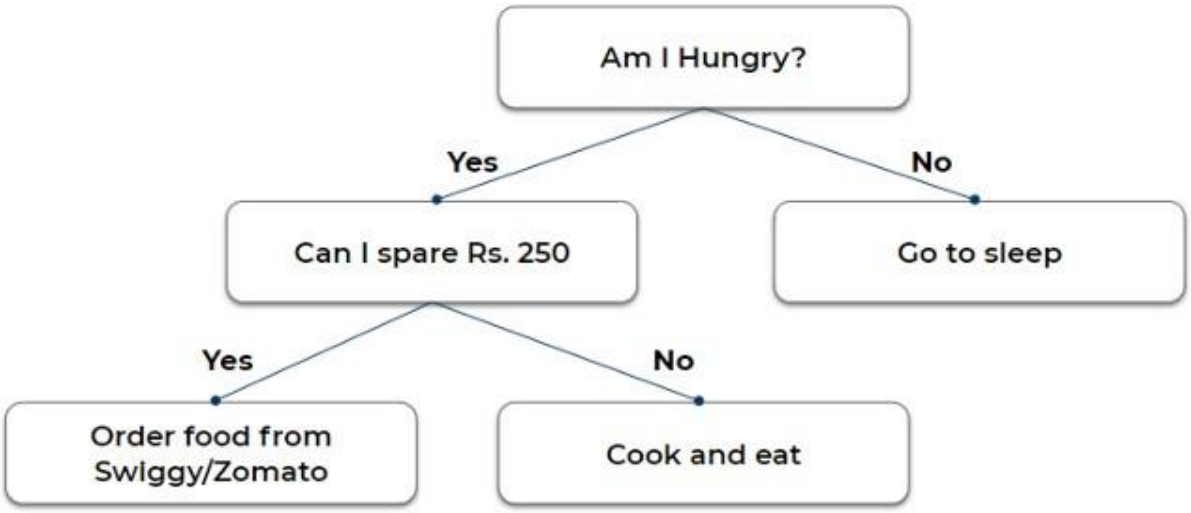
- Introduction to decision tree
- Types of Decision Tree
- How to Build a decision Tree from data
- Avoid over-fitting in decision trees
- Advantages and disadvantages of Decision Tree
- Implementing a decision tree using Python

Introduction to Decision Tree

Formally a decision tree is a **graphical representation of all possible solutions to a decision**. These days, tree-based algorithms are the most commonly used algorithms in the case of supervised learning scenarios. They are easier to interpret and visualize with great adaptability. We can use tree-based algorithms for both regression and classification problems, However, most of the time they are used for classification problem.

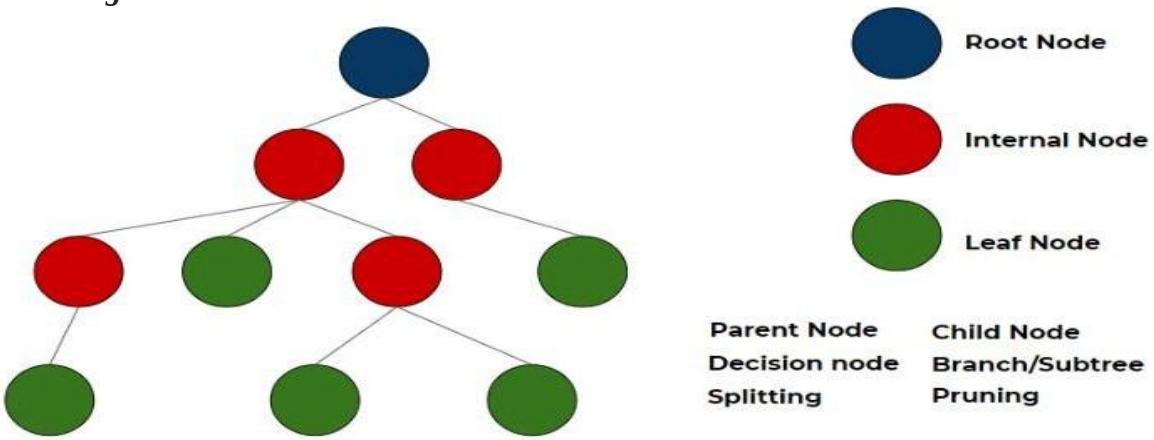
Let's understand a decision tree from an example: Yesterday evening, I *skipped dinner* at my usual time because I was busy taking care of some stuff. Later in the night, I felt butterflies in my stomach. I thought only if *I wasn't hungry*, I could have gone to sleep as it is but as that was not the case, *I decided to eat* something. I had two options, to *order something from outside* or cook myself.

I figured if I order, I will have to spare at least *INR 250* on it. I finally decided to order it anyway as it was pretty late and I was in no mood of cooking. This complete incident can be graphically represented as shown in the following figure.



This representation is nothing but a decision tree.

Terminologies associated with decision tree



- **Parent node**: In any two connected nodes, the one which is higher hierarchically, is a parent node.

- **Child node**: In any two connected nodes, the one which is lower hierarchically, is a child node.
- **Root node**: The starting node from which the tree starts, It has only child nodes. The root node does not have a parent node. (dark blue node in the above image)
- **Leaf Node/leaf:** Nodes at the end of the tree, which do not have any children are leaf nodes or called simply leaf. (green nodes in the above image)
- **Internal nodes/nodes**: All the in-between the root node and the leaf nodes are internal nodes or simply called nodes. internal nodes have both a parent and at least one child. (red nodes in the above image)
- **Splitting**: Dividing a node into two or more sun-nodes or adding two or more children to a node.
- **Decision node**: when a parent splits into two or more children nodes then that node is called a decision node.
- **Pruning:** When we remove the sub-node of a decision node, it is called pruning. You can understand it as the opposite process of splitting.
- **Branch/Sub-tree**: a subsection of the entire tree is called a branch or sub-tree.

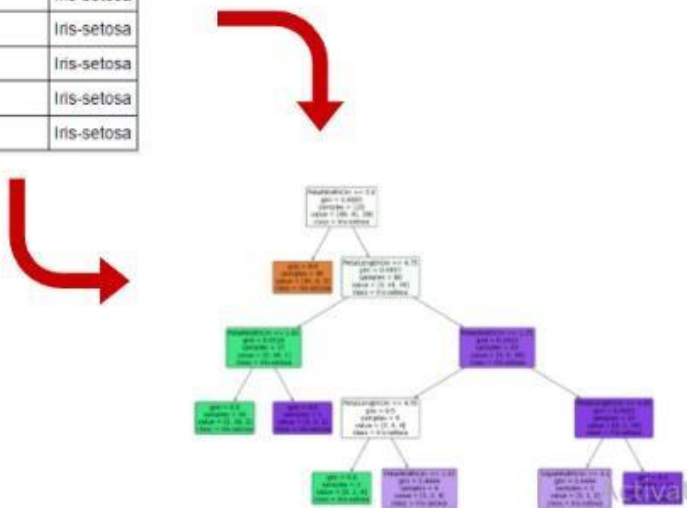
Types of Decision Tree
Regression Tree

A regression tree is used when the dependent variable is **continuous**. The value obtained by leaf nodes in the training data is the **mean** response of observation falling in that region. Thus, if an unseen data observation falls in that region, its prediction is made with the mean value. This means that even if the dependent variable in training data was continuous, it will only take discrete values in the test set. A regression tree follows a **top-down greedy approach.**
Classification Tree
A classification tree is used when the dependent variable is **categorical**. The value obtained by leaf nodes in the training data is the **mode** response of observation falling in that region It follows a **top-down greedy approach.**

Together they are called as CART(classification and regression tree)

Building a decision Tree from data

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
Id					
1	5.1	3.5	1.4	0.2	Iris-setosa
2	4.9	3.0	1.4	0.2	Iris-setosa
3	4.7	3.2	1.3	0.2	Iris-setosa
4	4.6	3.1	1.5	0.2	Iris-setosa
5	5.0	3.6	1.4	0.2	Iris-setosa



How to create a tree from tabular data? which feature should be selected as the root node? on what basis should a node be split? To all these questions answer is in this section

The decision of making strategic splits heavily affects a tree's accuracy. The purity of the node should increase with respect to the target variable after each split. The decision tree splits the nodes on all available variables and then selects the split which results in the most homogeneous sub-nodes.

The following are the most commonly used algorithms for splitting

1. Gini impurity

Gini says, if we select two items from a population at random then they must be of the same class and the probability for this is 1 if the population is pure.

1. It works with the categorical target variable "Success" or "Failure".
2. It performs only Binary splits
3. Higher the value of Gini higher the homogeneity.
4. CART (Classification and Regression Tree) uses the Gini method to create binary splits.

Steps to Calculate Gini impurity for a split

1. Calculate Gini impurity for sub-nodes, using the formula subtracting the sum of the square of probability for success and failure from one.
   $1-(p^2+q^2)$
   where p =P(Success) & q=P(Failure)
2. Calculate Gini for split using the weighted Gini score of each node of that split
3. Select the feature with the least Gini impurity for the split.

2. Chi-Square

It is an algorithm to find out the statistical significance between the differences between sub-nodes and parent node. We measure it by the sum of squares of standardized differences between observed and expected frequencies of the target variable.

1. It works with the categorical target variable "Success" or "Failure".
2. It can perform two or more splits.
3. Higher the value of Chi-Square higher the statistical significance of differences between sub-node and Parent node.
4. Chi-Square of each node is calculated using the formula,
5. Chi-square = $((\text{Actual} - \text{Expected})^2 / \text{Expected})^{1/2}$
6. It generates a tree called CHAID (Chi-square Automatic Interaction Detector)

Steps to Calculate Chi-square for a split:

1. Calculate Chi-square for an individual node by calculating the deviation for Success and Failure both
2. Calculated Chi-square of Split using Sum of all Chi-square of success and Failure of each node of the split
3. Select the split where Chi-Square is maximum.

3. Information Gain

A less impure node requires less information to describe it and, a more impure node requires more information. Information theory is a measure to define this degree of disorganization in a system known as Entropy. If the sample is completely homogeneous, then the entropy is zero and if the sample is equally divided (50% — 50%), it has an entropy of one. Entropy is calculated as follows.

$$Entropy = -p.log_2^p - q.log_2^q$$

Steps to calculate entropy for a split:

1. Calculate the entropy of the parent node
2. Calculate entropy of each individual node of split and calculate the weighted average of all sub-nodes available in the split. The lesser the entropy, the better it is.
3. calculate information gain as follows and chose the node with the highest information gain for splitting

$$Information\ Gain = 1 - Entropy$$

4. Reduction in Variance

Till now, we have discussed the algorithms for the categorical target variable. Reduction in variance is an algorithm used for continuous target variables (regression problems).

1. Used for continuous variables
2. This algorithm uses the standard formula of variance to choose the best split.
3. The split with lower variance is selected as the criteria to split the population

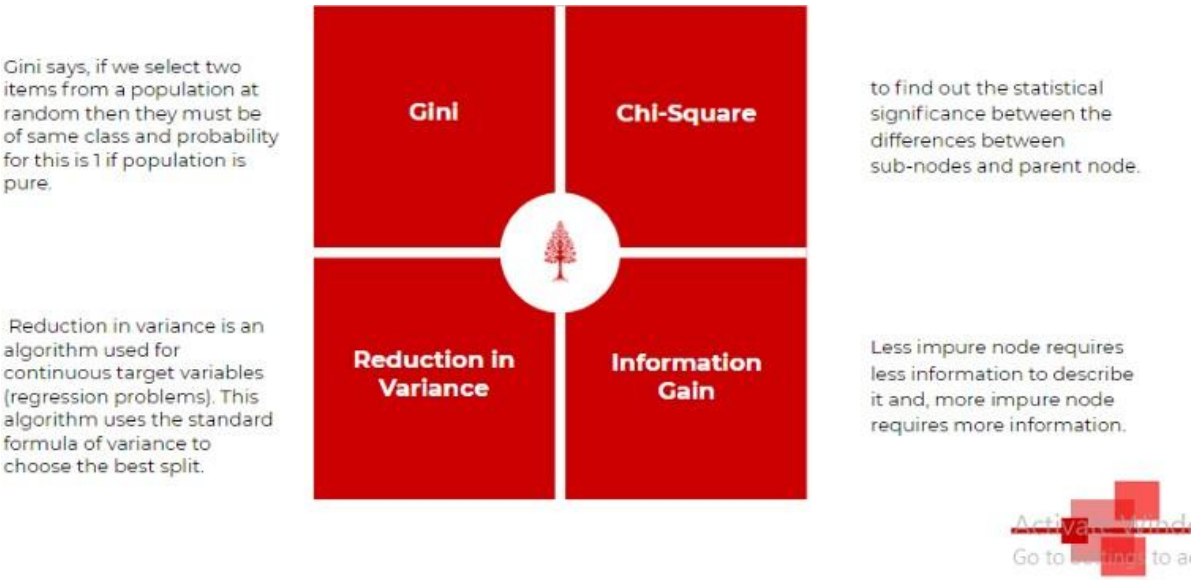
Steps to calculate Variance:

1. Calculate variance for each node.

$$\sigma^2 = \frac{\sum_{i=1}^{N} (x_i - \mu)^2}{N}$$

1. Calculate variance for each split as a weighted average of each node variance.
2. The node with lower variance is selected as the criteria to split.

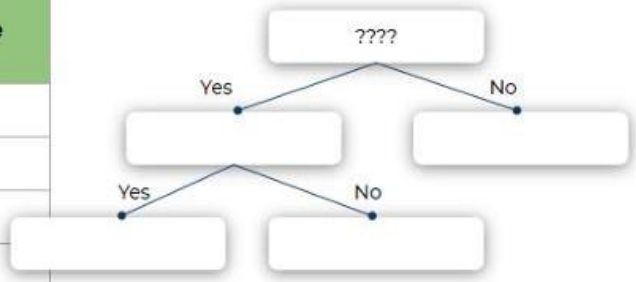
In summary:



In practice, most of the time Gini impurity is used as it gives good results for splitting and its computation is inexpensive. Let's build a tree with a pen and paper. We have a dummy dataset below, the features(X) are **Chest pain, Good blood circulation, Blocked arteries** and to be predicted column is **Heart disease(y).** Every column has two possible options **yes** and **no.**

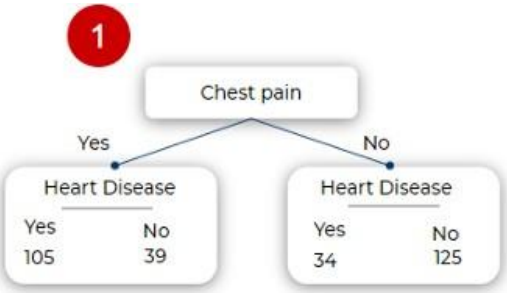
We aim to build a decision tree where given a new record of chest pain, good blood circulation, and blocked arteries we should be able to tell if that person has heart disease or not. At the start, all our samples are in the root node. We will have to decide on which of the feature the root node should be divided first.

Chest pain	Good blood circulation	Blocked arteries	Heart disease
yes	no	yes	no
yes	yes	yes	yes
no	no	yes	no
yes	yes	no	yes
...	...	...	...



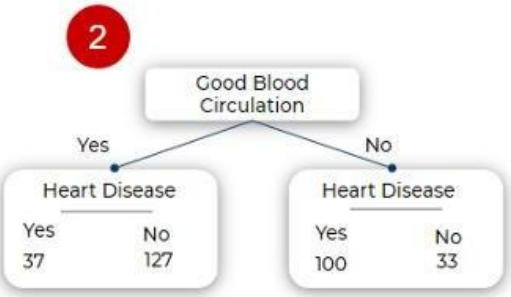
We will focus first on how heart disease is changing with Chest pain (ignoring good blood circulation and blood arteries). the dummy numbers are shown below

Chest pain	Good blood circulation	Blocked arteries	Heart disease
yes	no	yes	no
yes	yes	yes	yes
no	no	yes	no
yes	yes	no	yes
...	...	...	...



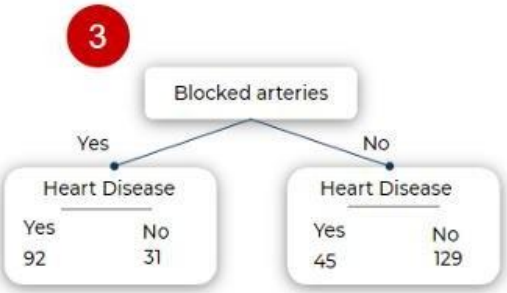
Similarly, we divide based on good communication as shown in the below image.

Chest pain	Good blood circulation	Blocked arteries	Heart disease
yes	no	yes	no
yes	yes	yes	yes
no	no	yes	no
yes	yes	no	yes
...	...	...	...



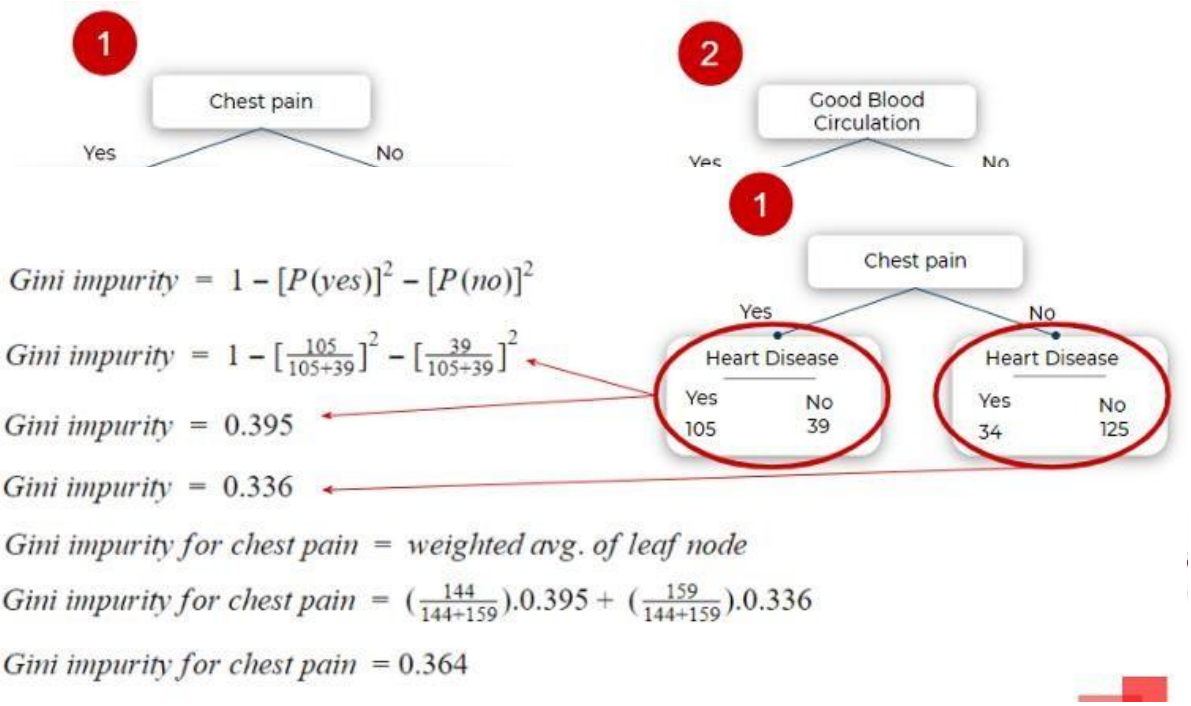
And with respect to blocked arteries as below image.

Chest pain	Good blood circulation	Blocked arteries	Heart disease
yes	no	yes	no
yes	yes	yes	yes
no	no	NA	no
yes	yes	no	yes
...	...	...	...

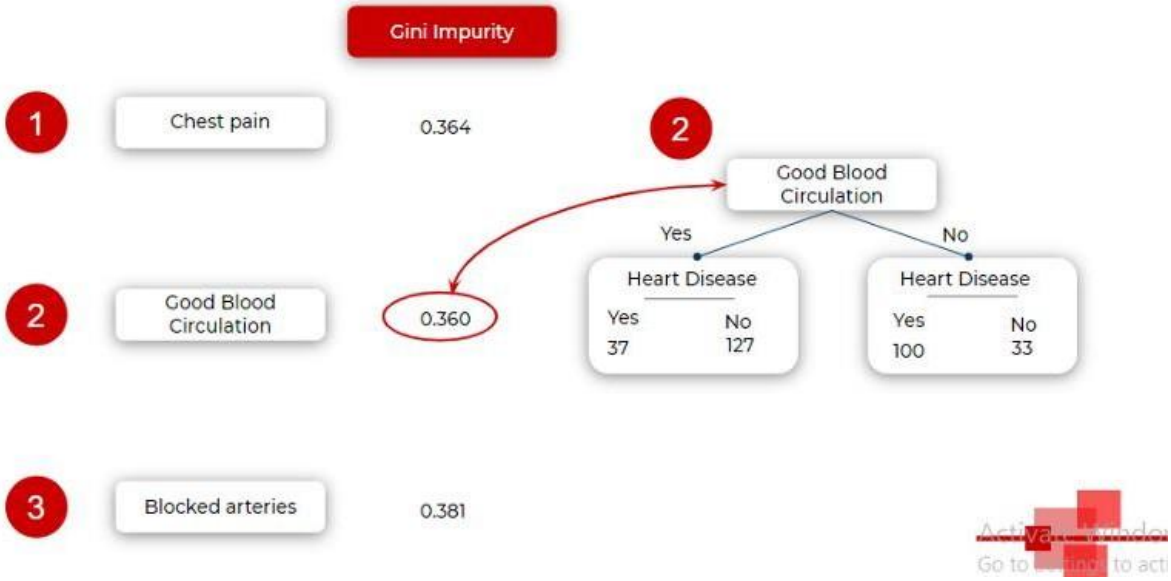


Taking all three splits at one place in the below image. We can observe that, it is not a great split on any of the feature alone for heart disease yes or no which means that one of these can be a root node but its not a full tree, we will have to split again down the tree in hope of better split.

To decide on which one feature should the root node be split, we need to calculate the Gini impurity for

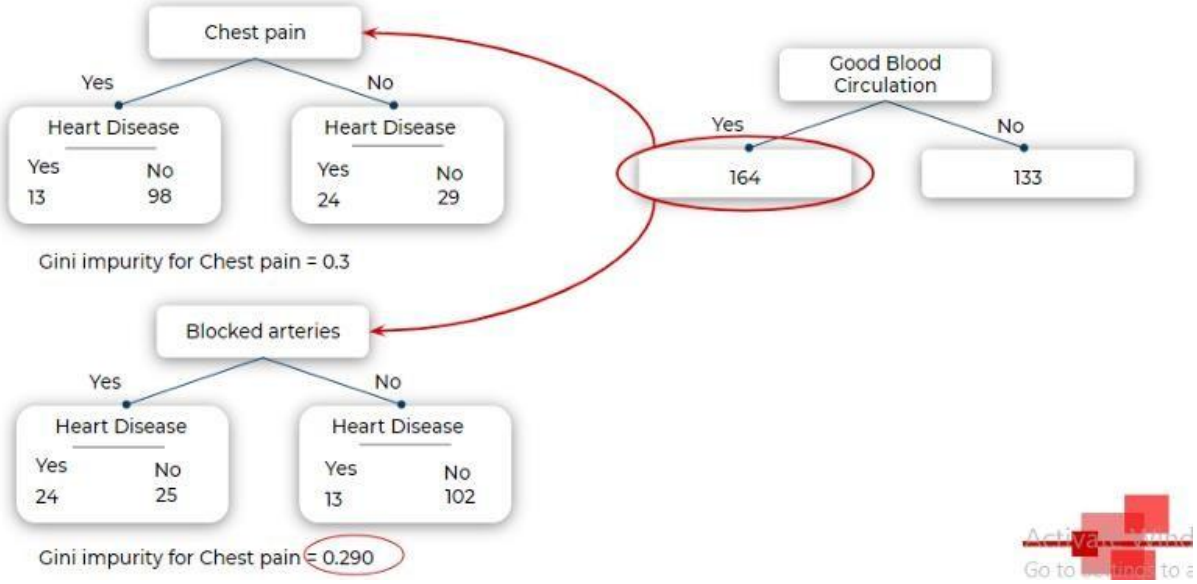


all the leaf nodes as shown below. After calculating for leaf nodes, we take its weighted average to get Gini impurity about the parent node.
We do this for all three features and select the one with the least Gini impurity as it is splitting the dataset in the best way out of three. Hence we choose good blood circulation as the root node.

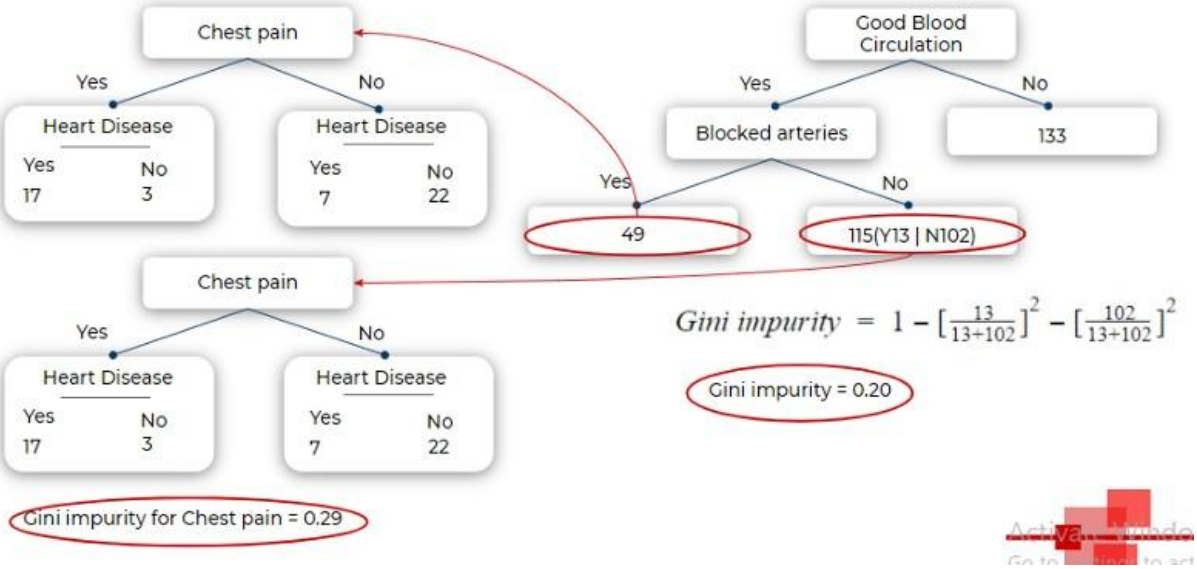


We do the same for a child node of Good blood circulation now. In the below image we will split the left child with a total of 164 sample on basis of blocked arteries as its Gini impurity is lesser than chest pain(we calculate Gini index again with the same formula as above, just a smaller subset of the sample — 164 in this case).

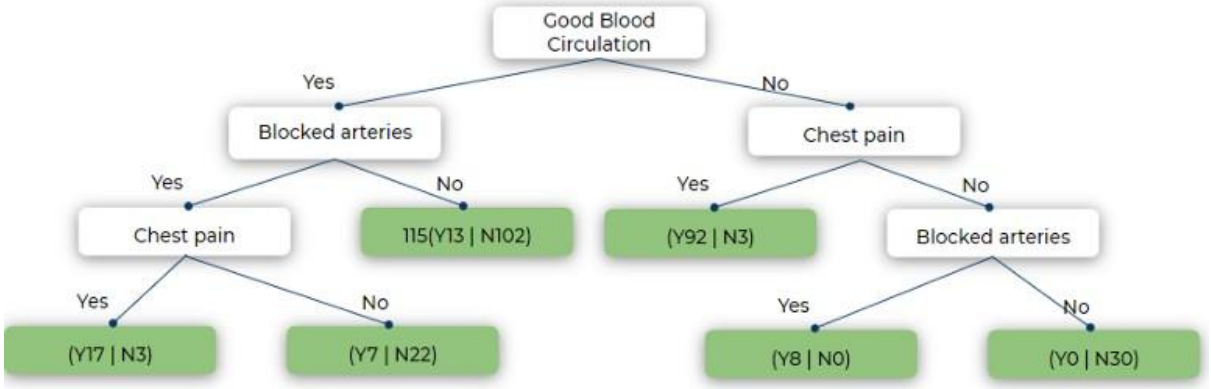
One thing to note in the below image that, when we try to split the right child of blocked arteries on basis of chest pain, the Gini index is 0.29 but the Gini impurity of the right child of the blocked tree itself, is



0.20. This means that splitting this node any further is not improving impurity. so this will be a leaf node.



We repeat the same process for all the nodes and we get the following tree. This looks a good enough fit for our training data.



This is how we create a tree from data. What should we do if we have a column with numerical values? It is simple, order them in ascending order. Calculate the mean on every two consecutive numbers. make a split on basis of that and calculate Gini impurity using the same method. We choose the split with the least Gini impurity as always.

Weight	Heart disease
155	no
167.5	
180	yes
185	
190	no
200	
220	yes
222.5	
225	yes
...	...

Gini impurity = 0.3 — Wt < 167.5

Gini impurity = 0.47 — Wt < 185

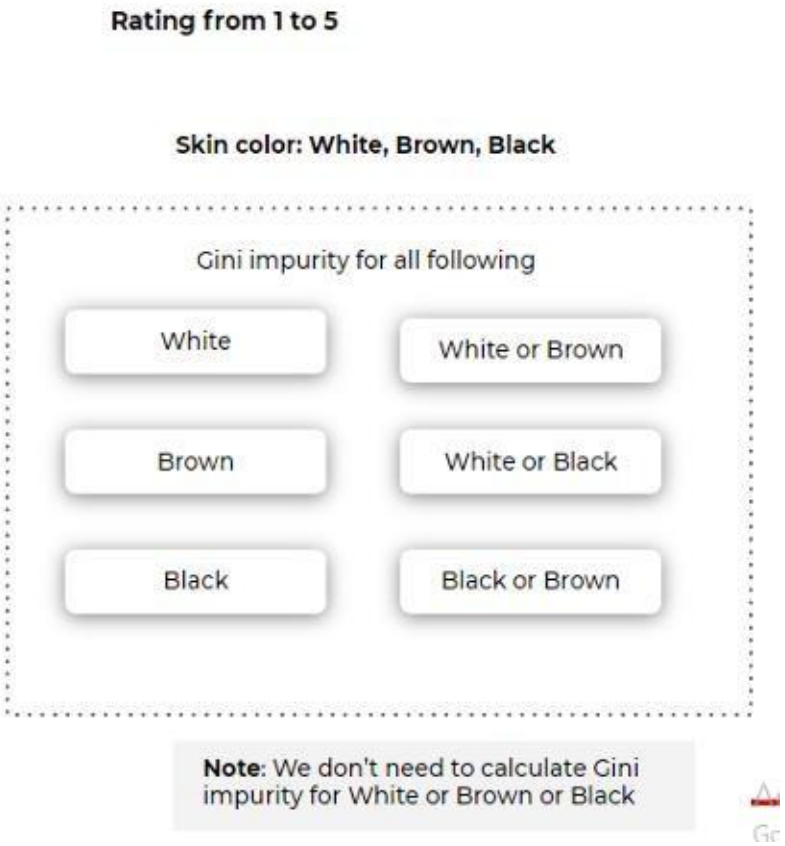
Gini impurity = 0.27 — Wt < 200

Gini impurity = 0.4 — Wt < 222.5

If we have ranked the numerical column in the dataset, we split on every rating and calculate Gini impurity

Rating	Heart disease

Skin Color	Heart disease
White	no
Brown	yes
Brown	no
Black	yes
White	yes
...	...



for each split, and select the one with the least Gini impurity.If we have categorical choices in our dataset then the split and Gini impurity calculation needs to be doneon every possible combination of choices

Avoid Overfitting in Decision Trees

Overfitting is one of the key challenges in a tree-based algorithm. If no limit is set, it will give 100% fitting, because, in the worst-case scenario, it will end up making a leaf node for each observation. Hence we need to take some precautions to avoid overfitting. It is mostly done in two ways:

1. Setting constraints on tree size
2. Tree pruning

Let's discuss both of these briefly.
Setting Constraints on tree size
Parameters play an important role in tree modeling. Overfitting can be avoided by using various parameters that are used to define a tree.

1. **Minimum samples for a node split**
    1. Defines the minimum number of observations that are required in a node to be considered for splitting. (this ensures above mentioned worst-case scenario).
    2. A higher value of this parameter prevents a model from learning relations that might be highly specific to the particular sample selected for a tree.
    3. Too high values can lead to under-fitting hence, it should be tuned properly using cross-validation.
2. *Minimum samples for a leaf node*
    1. Defines the minimum observations required in a leaf. (Again this also prevents worst-case scenarios)
    2. Generally, lower values should be chosen for imbalanced class problems as the regions in which the minority class will be in majority will be of small size.

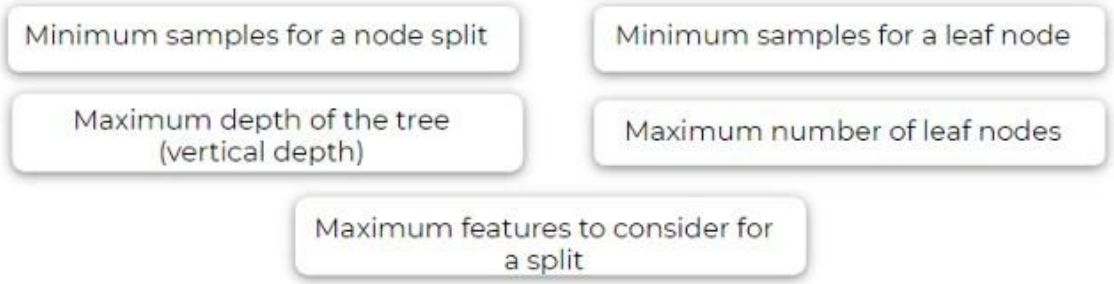
1. **Maximum depth of the tree (vertical depth)**
2. Used to control over-fitting as higher depth will allow the model to learn relations very specific to a particular sample.
3. Should be tuned properly using Cross-validation as too little height can cause underfitting.

3. Maximum number of leaf nodes

1. The maximum number of leaf nodes or leaves in a tree.
2. Can be defined in place of max_depth. Since binary trees are created, a depth of n would produce a maximum of 2^n leaves.

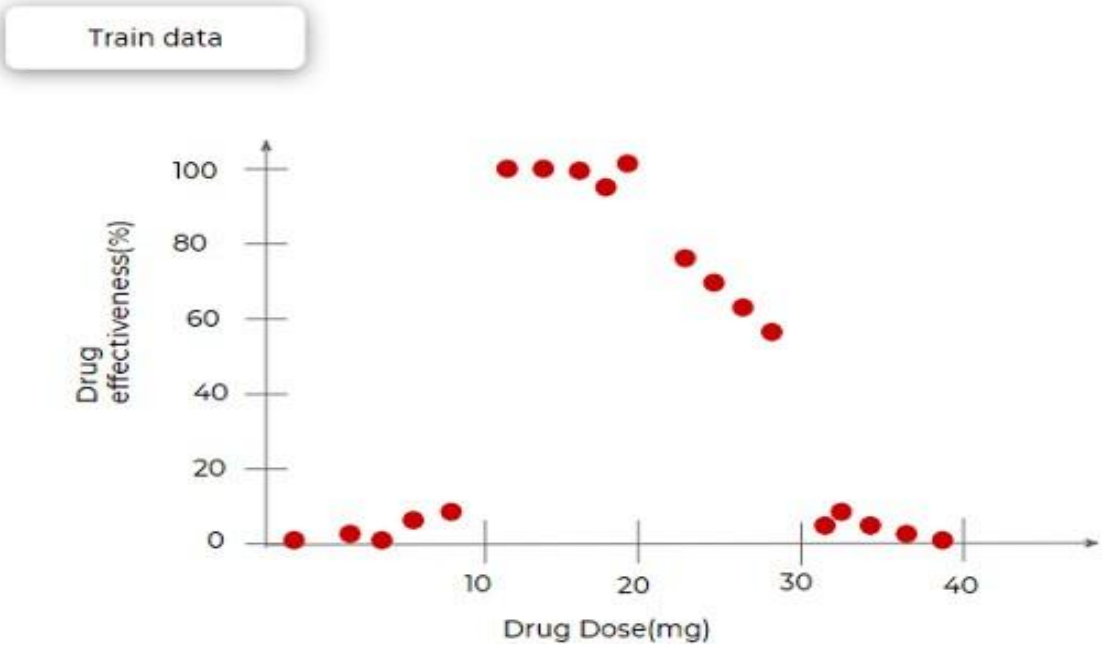
4. Maximum features to consider for a split

1. The number of features to consider while searching for the best split. These will be randomly selected.
2. As a thumb-rule, the square root of the total number of features works great but we should check up to 30–40% of the total number of features.
3. Higher values can lead to over-fitting but depend on case to case.



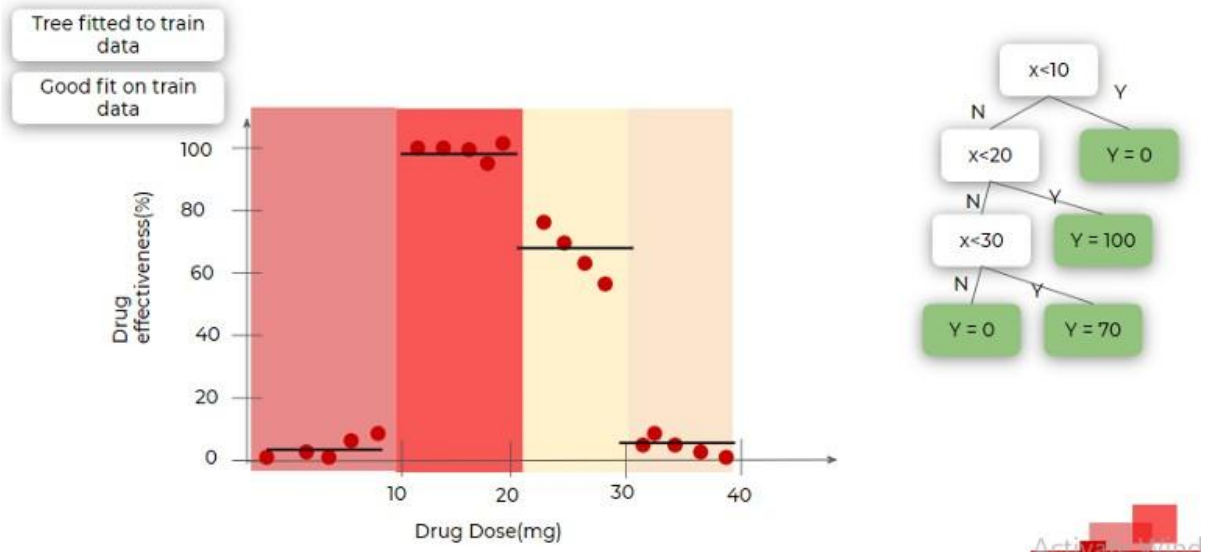
Pruning

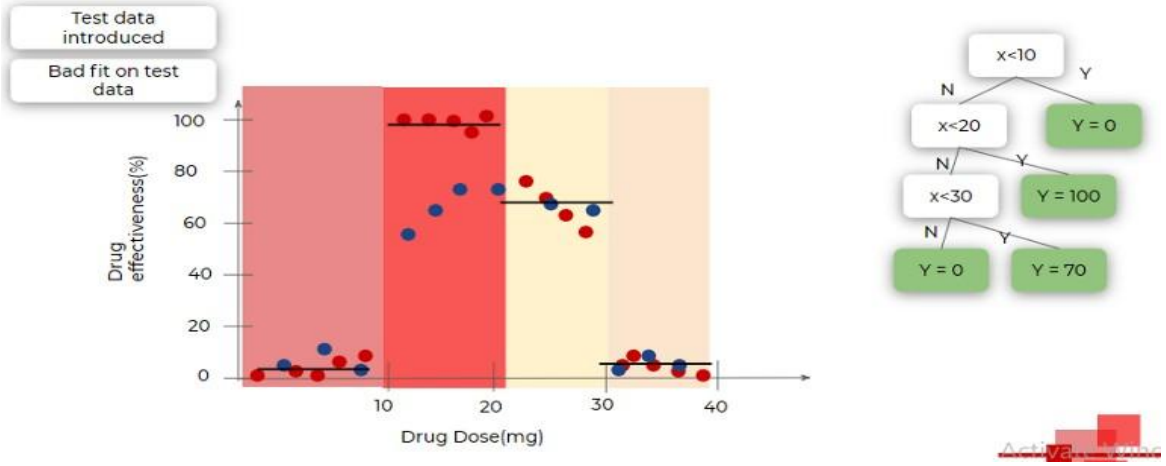
Recall from terminologies, pruning is something opposite to splitting. Now we see how exactly that is the case. Let's see the following example where drug effectiveness is plotted against drug doses. All the red points are the training dataset. We can see that for very small and very large quantities of doses, the drug effectiveness is almost negligible. Between 10–20mg, almost 100% and gradually decreasing between 20 to 30.



We modeled a tree and we got the following results. we did splitting at three places and got 4 leaf nodes which will give output as 0(y): 0–10(X), 100:10–20, 70:20–30, 0:30–40 respectively as we increase the doses. It is a great fit for the training dataset, the black horizontal lines are output given for the node.

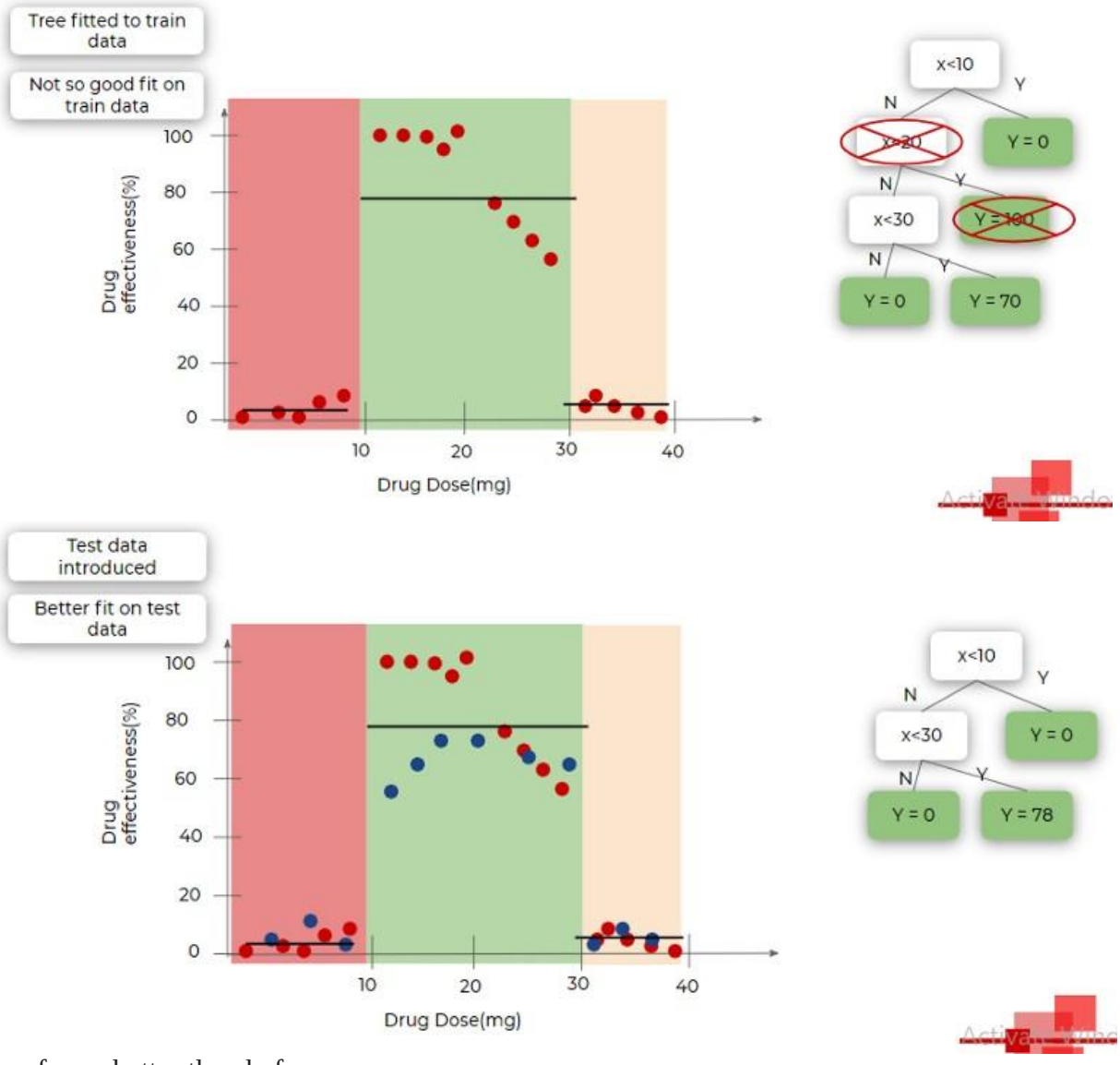
Let’s test it on the training dataset (blue dots in the following image are of the testing dataset). It is a bad fit for testing data. A good fit for training





ata and a bad fit for testing data means overfitting.To overcome the overfitting issue of our tree, we decide to merge two segments in the middle which means removing two nodes from the tree as you can see in the image below(the nodes with the red crosson them are removed). now again we fit this tree on the training dataset. Now, the tree is not that great fitfor training data.

But, when we introduce testing data, it performs better than before.



This is basically pruning. Using pruning we can avoid overfitting to the training dataset.

Advantages and disadvantages of Decision Tree
Advantages of a decision tree

- **Easy to visualize and interpret**: Its graphical representation is very intuitive to understand and it does not require any knowledge of statistics to interpret it.
- **Useful in data exploration**: We can easily identify the most significant variable and the relation between variables with a decision tree. It can help us create new variables or put some features in

one bucket.

- **Less data cleaning required**: It is fairly immune to outliers and missing data, hence less data cleaning is needed.
- **The data type is not a constraint**: It can handle both categorical and numerical data.

Disadvantages of decision tree

- **Overfitting**: single decision tree tends to overfit the data which is solved by setting constraints on model parameters i.e. height of the tree and pruning(which we will discuss in detail later in this article)
- **Not exact fit for continuous data**: It losses some of the information associated with numerical variables when it classifies them into different categories.

Implementing a decision tree using Python

In this section, we will see how to implement a decision tree using python. We will use the famous IRIS dataset for the same. The purpose is if we feed any new data to this classifier, it should be able to predict the right class accordingly.

You can get complete code for this implementation here

Importing the Required Libraries and Reading the data

First thing is to import all the necessary libraries and classes.

```
import pandas as pd
import numpy as np
```

```
#reading the data
df = sns.load_dataset('iris')
df.head()
```

```
from sklearn.model_selection import train_test_split#for decision tree object
from sklearn.tree import DecisionTreeClassifier#for checking testing results
from sklearn.metrics import classification_report, confusion_matrix#for visualizing tree
from sklearn.tree import plot_tree
```

2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

Now load the dataset. IRIS dataset is available in the seaborn library as well. you can import it with the following command

you should be able to get the above data. Here we have 4 feature columns **sepal_length, sepal_width, petal_length, and petal_width** respectively with one target column **species**.

EDA

Now performing some basic operations on it.

```
#getting information of dataset
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   sepal_length  150 non-null    float64
 1   sepal_width   150 non-null    float64
 2   petal_length  150 non-null    float64
 3   petal_width   150 non-null    float64
 4   species       150 non-null    object
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

```
df.shape
```

```
(150, 5)
```

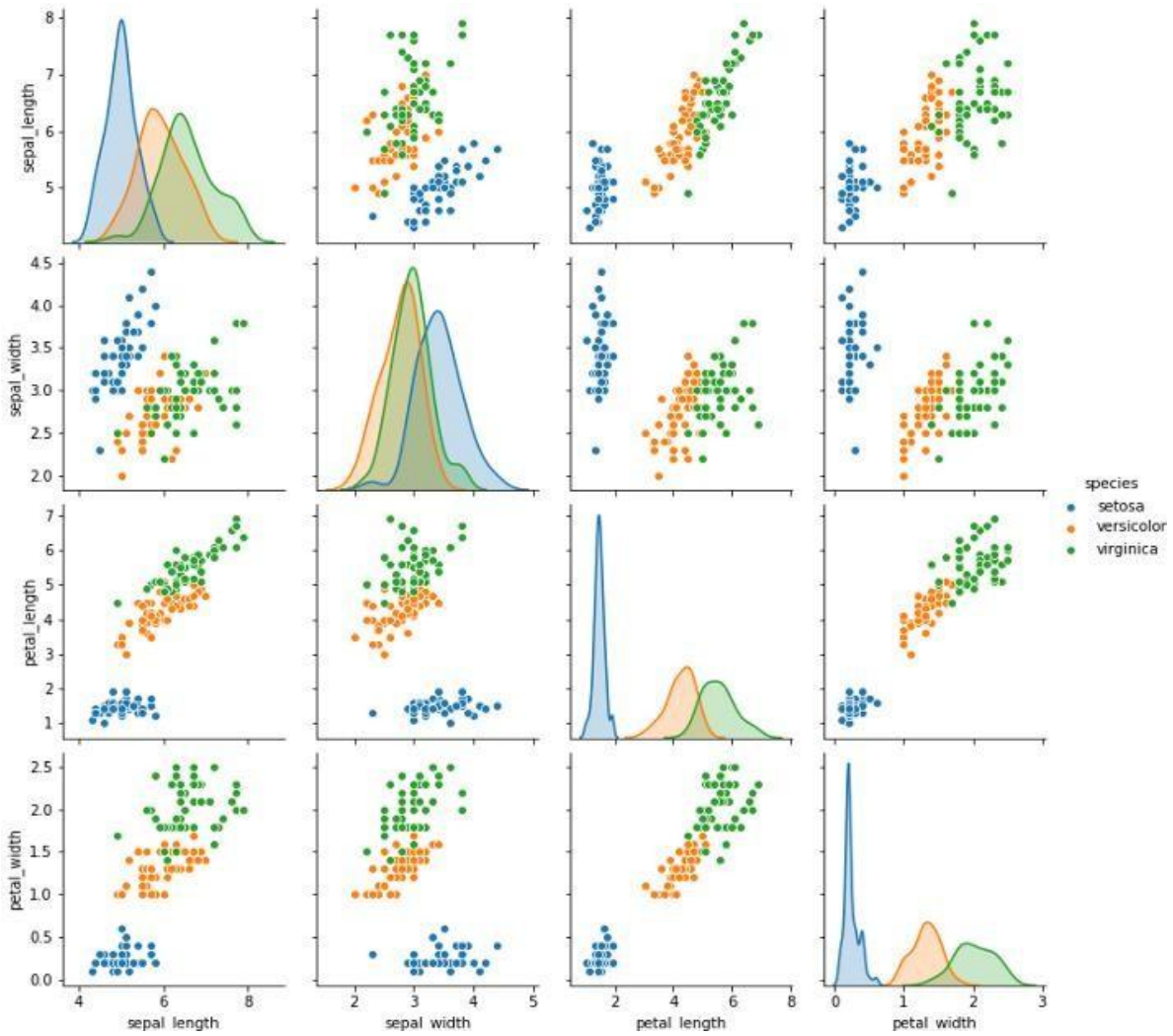
we understand that this dataset has 150 records, 5 columns with the first four of type float and last of type object str and there are no NAN values as form following command

```
df.isnull().any()
```

```
sepal_length    False
sepal_width     False
petal_length    False
petal_width     False
species         False
dtype: bool
```

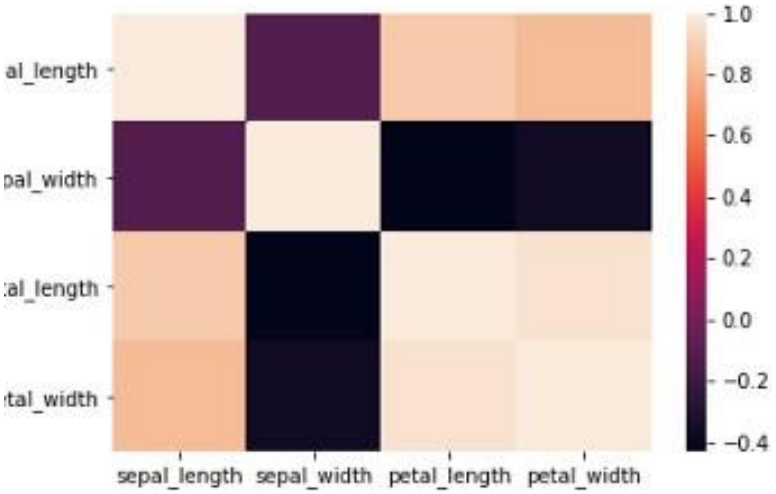
Now we perform some basic EDA on this dataset. Let’s check the correlation of all the features with each other

```
# let's plot pair plot to visualise the attributes all at once
sns.pairplot(data=df, hue = 'species')
```



We have a total of 3 species that we want to predict: **setosa, versicolor, and virginica.** We can see thatsetosa always forms a different cluster from the other two.

```
# correlation matrix
sns.heatmap(df.corr())
```



We can observe from the above two plots:

1. Setosa always forms a different cluster.
2. Petal length is highly related to petal width.
3. Sepal length is not related to sepal width.

Data Preprocessing

Now, we will separate the target variable(y) and features(X) as follows

```
target = df['species']
df1 = df.copy()
df1 = df1.drop('species', axis =1)
```

It is good practice not to drop or add a new column to the original dataset. Make a copy of it and then modify it so in case things don't work out as we expected, we have the original data to start again with a different approach.

Just for the sake of following mostly used convention, we are storing df in X

```
# Defining the attributes
X = df1
```

Now let's look at our target variable

```
target
```

```
0          setosa
1          setosa
2          setosa
3          setosa
4          setosa
          ...
145    virginica
146    virginica
147    virginica
148    virginica
149    virginica
Name: species, Length: 150, dtype: object
```

target has categorical variables stored in it we will encode it in numeric values for working.

```
#label encoding
le = LabelEncoder()
target = le.fit_transform(target)
target
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], dtype=int64)
```

We get its encoding as above, setosa:0, versicolor:1, virginica:2
Again for the sake of following the standard naming convention, naming target as y

```
y = target
```

Splitting the dataset into training and testing sets. selecting 20% records randomly for testing

```
# Splitting the data - 80:20 ratio

X_train, X_test, y_train, y_test = train_test_split(X , y, test_size = 0.2, random_state = 42)print("Training split input- ", X_train.shape)

print("Testing split input- ", X_test.shape)
```

```
Training split input-  (120, 4)
Testing split input-  (30, 4)
```

After splitting the dataset we have 120 records(rows) for training and 30 records for testing purposes.

Modeling Tree and testing it

```
# Defining the decision tree algorithmdtree=DecisionTreeClassifier()

dtree.fit(X_train,y_train)print('Decision Tree Classifier Created')
```

In the above code, we created an object of the class DecisionTreeClassifier , store its address in the variable dtree, so we can access the object using dtree. Then we fit this tree with our X_train and y_train . Finally, we print the statement Decision Tree Classifier Created after the decision tree is built.

```
# Predicting the values of test data

y_pred = dtree.predict(X_test)

print("Classification report - \n", classification_report(y_test,y_pred))
```

```
Classification report -
              precision    recall  f1-score   support

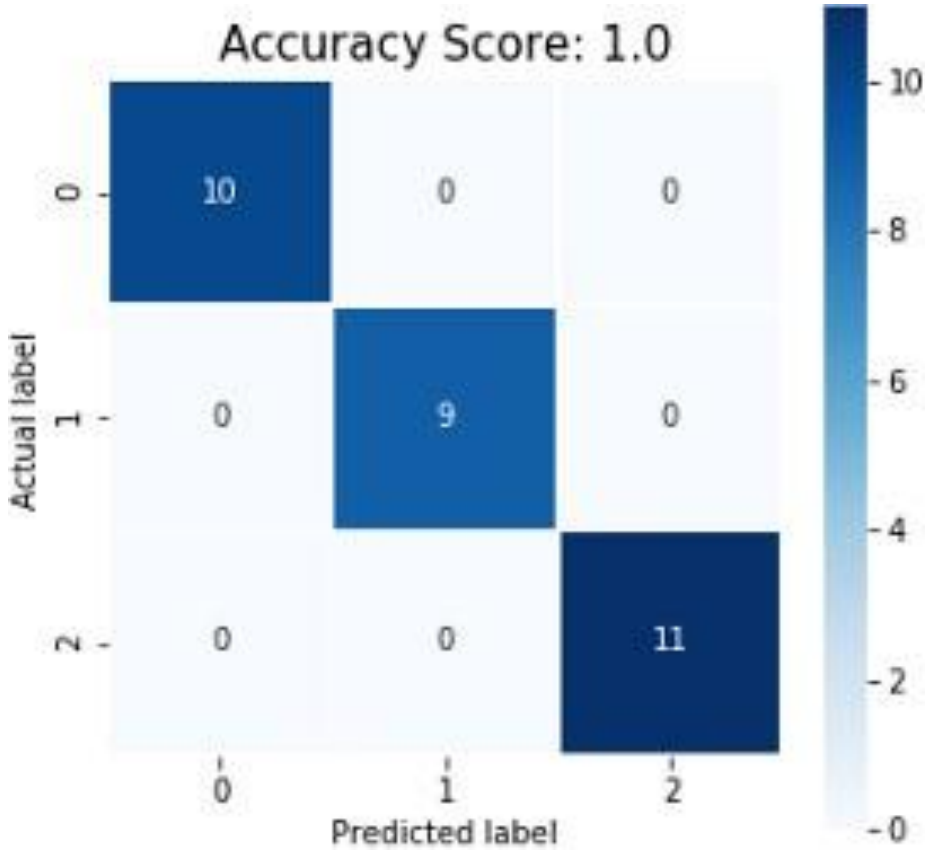
           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00         9
           2       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```

We got an accuracy of 100% on the testing dataset of 30 records.

let’s plot the confusion matrix as follows

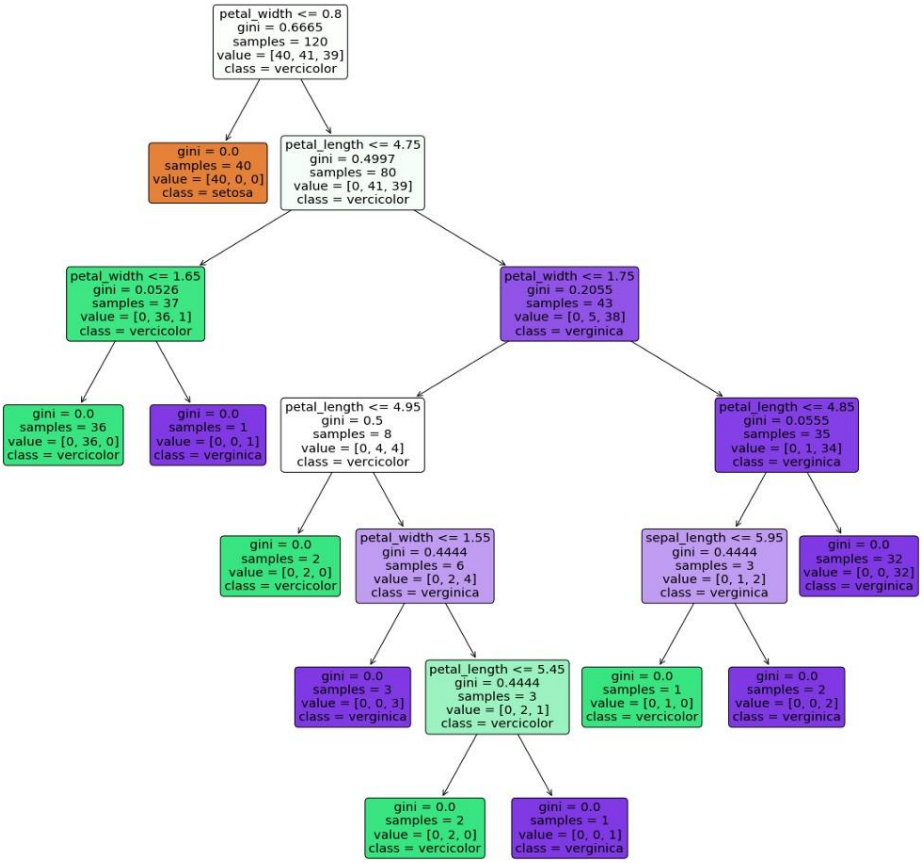
```
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(5,5))sns.heatmap(data=cm,linewidths=.5, annot=True,square = True, cmap ='Blues')plt.ylabel('Actual label')
plt.xlabel('Predicted label')all_sample_title = 'Accuracy Score: {0}'.format(dtree.score(X_test,
y_test))plt.title(all_sample_title, size = 15)
```



Visualizing the decision tree

We can directly plot the tree that we build using the following commands

```
# Visualising the graph without the use of graphvizplt.figure(figsize = (20,20))
dec_tree = plot_tree(decision_tree=dtree, feature_names = df1.columns,
                     class_names =["setosa", "vercicolor", "verginica"] , filled = True , precision = 4, rounded = True)
```



We can see how the tree is split, what are the gini for the nodes, the records in those nodes, and their labels.

All the images are from the author unless given credit explicitly.

Result:

Thus the decision tree algorithms is implemented

Ex.No. 5.IMPLEMENT BACK PROPAGATION ALGORITHM

Aim

To implement a back propagation algorithm

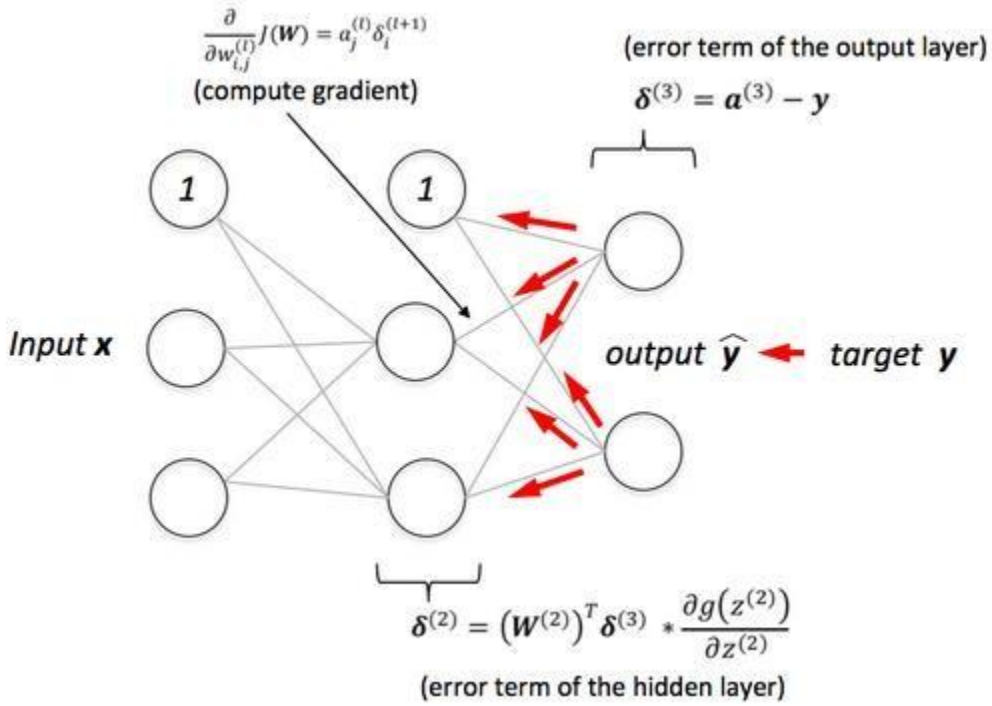
Procedure and implementation

Backpropagation neural network is used to improve the accuracy of neural network and make them capable of self-learning. Backpropagation means "backward propagation of errors". Here error is spread into the reverse direction in order to achieve better performance.

Backpropagation is an algorithm for supervised learning of artificial neural networks that uses the gradient descent method to minimize the cost function. It searches for optimal weights that optimize the mean-squared distance between the predicted and actual labels.

What is Backpropagation Neural Network(BPN)?

BPN was discovered by Rumelhart, Williams & Honton in 1986. The core concept of BPN is to backpropagate or spread the error from units of output layer to internal hidden layers in order to tune the weights to ensure lower error rates. It is considered a practice of fine-tuning the weights of neural networks in each iteration. Proper tuning of the weights will make a sure minimum loss and this will make a more robust, and generalizable trained neural network.



https://www.bogotobogo.com/python/scikit-learn/images/NeuralNetwork4-Backpropagation/Backpropagation-Diagram.png

How BPN works?

BPN learns in an iterative manner. In each iteration, it compares training examples with the actual target label. target label can be a class label or continuous value. The backpropagation algorithm works in the following steps:

- **Initialize Network:** BPN randomly initializes the weights.

- **Forward Propagate:** After initialization, we will propagate into the forward direction. In this phase, we will compute the output and calculate the error from the target output.
- **Back Propagate Error**: For each observation, weights are modified in order to reduce the error in a technique called the delta rule or gradient descent. It modifies weights in a "backward" direction to all the hidden layers.

Implementation in Python

Import Libraries

Lets import the required modules and libraries such as numpy, pandas, scikit-learn, and matplotlib.

```
# Import Libraries
import numpy as np
import pandas as pd
from sklearn datasets import load_iris
from sklearn model_selection import train_test_split
import matplotlib pyplot as plt
```

Load Dataset

Let's first load the Iris dataset using load_iris() function of scikit-learn library and seprate them in features and target labels. This data set has three classes Iris-setosa, Iris-versicolor, and Iris-virginica.

```
# Load dataset
data = load_iris()

# Get features and target
X=data data
y=data target
```

Prepare Dataset

Create dummy variables for class labels using get_dummies() function

```
# Get dummy variable
y = pd get_dummies(y) values

y[3]
```

Output:

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

       [1, 0, 0],

       [1, 0, 0]], dtype=uint8)
```

Split train and test set

To understand model performance, dividing the dataset into a training set and a test set is a good strategy. Let's split dataset by using function train_test_split(). you need to pass basically 3 parameters features, target, and test_set size. Additionally, you can use random_state in order to get the same kind of train and test set.

```
#Split data into train and test data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=20, random_state=4)
```

Initialize Hyperparameters and Weights

Lets initialize the hyperparameters such as learning rate, iterations, input size, number of hidden layers, and number of output layers.

```
# Initialize variables
learning_rate = 0.1
iterations = 5000
N = y_train size

# number of input features
input_size = 4

# number of hidden layers neurons
hidden_size = 2

# number of neurons at the output layer
output_size = 3

results = pd DataFrame columns=["mse", "accuracy"])
```

Lets initialize the weights for hidden and output layers with random values.

```
# Initialize weights
np random seed 10)

# initializing weight for the hidden layer
W1 = np random normal scale=0.5  size= input_size  hidden_size)

# initializing weight for the output layer
W2 = np random normal scale=0.5  size= hidden_size  output_size))
```

Helper Functions

Lets create helper functions such as sigmoid, mean_square_error, and accuracy.

```
def sigmoid x):
    return 1 / 1 + np exp -x))

def mean_squared_error y_pred  y_true):
    return ((y_pred - y_true **2) sum() / 2*y_pred size)

def accuracy y_pred  y_true):
    acc = y_pred argmax axis=1) == y_true argmax axis=1)
    return acc mean()
```

Backpropagation Neural Network

In this phase, we will create backpropagation neural network in three steps feedforward propagation, error calculation and backpropagation phase. Here , we will create a for loop for given number of iterations that execute the three steps(feedforward propagation, error calculation and backpropagation phase) and update the weights in each iteration.

```
for itr in range iterations):

    # feedforward propagation
    # on hidden layer
```

```
    Z1 = np.dot(x_train, W1)
    A1 = sigmoid(Z1)

    # on output layer
    Z2 = np.dot(A1, W2)
    A2 = sigmoid(Z2)


    # Calculating error
    mse = mean_squared_error(A2, y_train)
    acc = accuracy(A2, y_train)
    results=results.append({"mse":mse, "accuracy":acc}, ignore_index=True )

    # backpropagation
    E1 = A2 - y_train
    dW1 = E1 * A2 * (1 - A2)

    E2 = np.dot(dW1, W2.T)
    dW2 = E2 * A1 * (1 - A1)


    # weight updates
    W2_update = np.dot(A1.T, dW1) / N
    W1_update = np.dot(x_train.T, dW2) / N

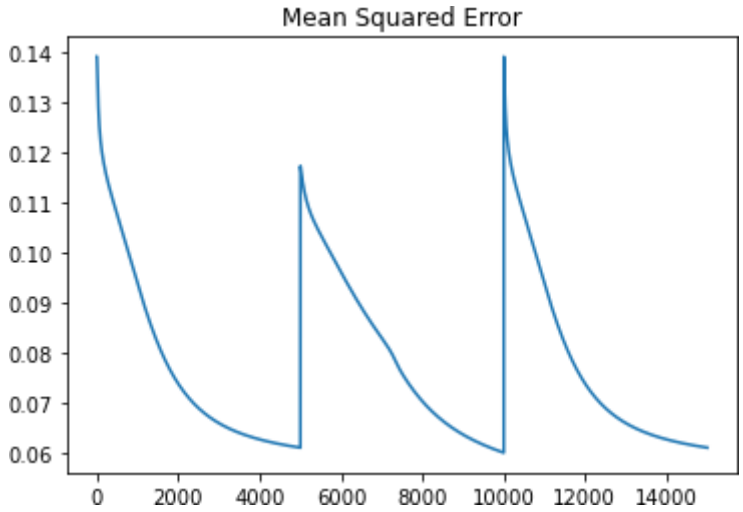
    W2 = W2 - learning_rate * W2_update
    W1 = W1 - learning_rate * W1_update
```

Plot MSE and Accuracy

Lets plot mean squared error in each iteration using pandas plot() funciton.

```
results.mse.plot(title="Mean Squared Error")
```

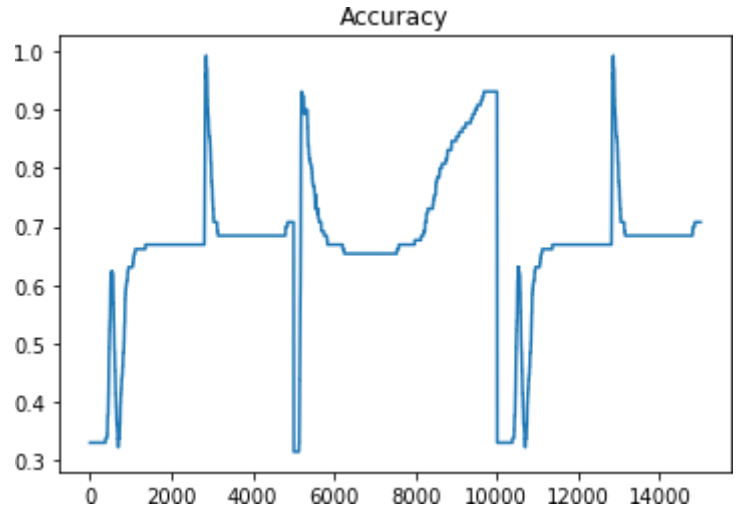
Output:



Lets plot accuracy in each iteration using pandas plot() funciton.

```
results.accuracy.plot(title="Accuracy")
```

Output:



Predict for Test Data and Evaluate the Performance

Lets make prediction for the test data and assess the performance of Backpropagation neural network.

```
# feedforward
Z1 = np dot( x_test  W1)
A1 = sigmoid( Z1)

Z2 = np dot( A1  W2)
A2 = sigmoid( Z2)

acc = accuracy( A2  y_test)
print "Accuracy: {}" format( acc)
```

Output:

```
Accuracy: 0.8
```

you can see in the above output, we are getting 80% accuracy on test dataset.

Pros and Cons

Backpropagation Neural Network is a simple and faster model compared to its earlier models. It is also a flexible and standard method. It does not need any prior knowledge for training.

BPN performance depends upon the kind of input data is used. It is quite sensitive to noisy data. We need to use a matrix-based approach instead of a mini-batch.

Conclusion

Congratulations, you have made it to the end of this tutorial!

Backpropagation neural network is a method to optimize neural networks by propagating the error or loss into a backward direction. It finds loss for each node and updates its weights accordingly in order to minimize the loss using gradient descent.

In this tutorial, you have learned What is Backpropagation Neural Network, Backpropagation algorithm working, and Implementation from scratch in python. We have also discussed the pros and cons of the Backpropagation Neural Network.

References

- Basics of Deep Learning p.13 – Implementing the Backpropagation Algorithm with NumPy: https://www.sebastian-mantey.com/theory-blog/basics-of-deep-learning-p13- implementing-the-backpropagation-algorithm-with-numpy

Result:

Thus the back propagation algorithm is implemented using python.

Ex.No: 6.IMPLEMENT SIMILARITY BASED CLUSTERING ALGORITHM

Aim:

To implement similarity based clustering algorithm.

Introduction

Have you ever dreamt of building your own Image Similarity Application but are scared that you don't know enough of Deep Learning, Convolutional Neural Network, and more? Do not worry. The following tutorial will help you get started and help you code your own Image Similarity Application with basic mathematics.

Before we jump into maths and code I would ask you a simple question. Given two reference images and one test image, which one do you think our test image belongs two :

Reference Image 1



Reference Image 2



Test Image



If you believe that our Test Image is similar to our first reference image you are right. If you do believe otherwise then let's find out together with the power of mathematics and programming.

"The future of search will be about pictures rather than keywords." — ***Ben Silbermann, Pinterest CEO.***

Image **Vector**

Every image is stored in our computer in the form of numbers and a vector of such numbers that can completely describe our image is known as an Image Vector.

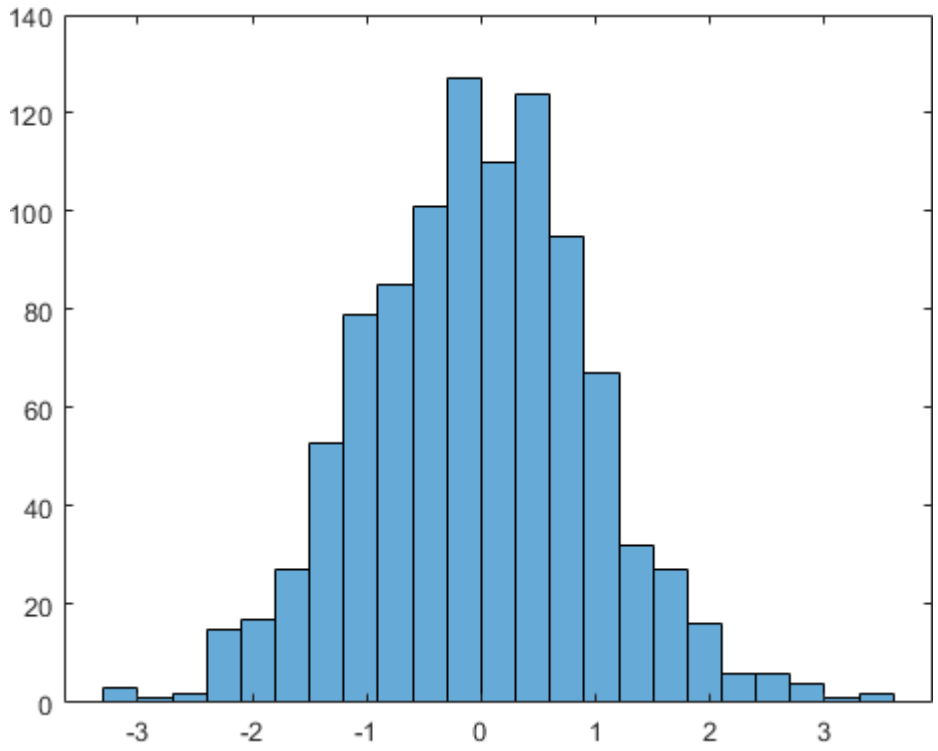
Euclidean Distance:

Euclidean Distance represents the distance between any two points in an n-dimensional space. Since we are representing our images as image vectors they are nothing but a point in an n-dimensional space and we are going to use the euclidean distance to find the distance between them.

$$d = ((p_1 - q_1)^2 + (p_2 - q_2)^2)^{1/2}$$

Histogram:

A histogram is a graphical display of numerical values. We are going to use the image vector for all three images and then find the euclidean distance between them. Based on the values returned the image with a lesser distance is more similar than the other.



To find the similarity between the two images we are going to use the following approach :

1. Read the image files as an array.
2. Since the image files are colored there are 3 channels for RGB values. We are going to flatten them such that each image is a single 1-D array.
3. Once we have our image files as an array we are going to generate a histogram for each image where for each index 0 – 255 we are going the count the occurrence of that pixel value in the image.
4. Once we have our histograms we are going to use the L2-Norm or Euclidean Distance to find the difference the two histograms.
5. Based on the distance between the histogram of our test image and the reference images we can find the image our test image is most similar to.

Coding for Image Similarity in Python

Import the dependencies we are going to use

```
from PIL import Image
from collections import Counter
import numpy as np
```

We are going to use NumPy for storing the image as a NumPy array, Image to read the image in terms of numerical values and Counter to count the number of times each pixel value (0-255) occurs in the images.

Reading the Image

```
reference_image_1 = Image.open('Reference_image1.jpg')
reference_image_arr = np.asarray(reference_image_1)
print(np.shape(reference_image_arr))
>>> (250, 320, 3)
```

We can see that out image has been successfully read as a 3-D array. In the next step, we need to flatten this 3-D array into a 1-Dimensional array.

We are going to do the same steps for the other two images. I will skip that here so that you can try your

```
flat_array_1 = array1.flatten()
print(np.shape(flat_array_1))
>>> (245760, )
```

hands on it too.

Generating the Count-Histogram-Vector :

```
RH1 = Counter(flat_array_1)
```

The following line of code returns a dictionary where the key corresponds to the pixel value and the value of the key is the number of times that pixel is present in the image.

One limitation of Euclidean distance is that it requires all the vectors to be normalized i.e both the vectors need to be of the same dimensions. To ensure that our histogram vector is normalized we are going to use a for loop from 0-255 and generate our histogram with the value of the key if the key is present in the image else we append a 0.

```
H1 = []
for i in range(256):
    if i in RH1.keys():
        H1.append(RH1[i])
```

```
    else:
        H1.append(0)
```

The above piece of code generates a vector of size (256,) where each index corresponds to the pixel value and the value corresponds to the count of the pixel in that image.

We follow the same steps for the other two images and obtain their corresponding Count-Histogram-Vectors. At this point we have our final vectors for both the reference images and the test image and all we need to do is calculate the distances and predict.

Euclidean Distance Function :

```
def L2Norm(H1,H2):
    distance =0
    for i in range(len(H1)):
        distance += np.square(H1[i]-H2[i])
    return np.sqrt(distance)
```

The above function takes in two histograms and returns the euclidean distance between them.

Evaluation :

Since we have everything we need to find the image similarities let us find out the distance between the test image and our first reference image.

```
dist_test_ref_1 = L2Norm(H1,test_H)
print("The distance between Reference_Image_1 and Test Image is : {}".format(dist_test_ref_1))
>>> The distance between Reference_Image_1 and Test Image is : 9882.175468994668
```

Let us now find out the distance between the test image and our second reference image.

```
dist_test_ref_2 = L2Norm(H2,test_H)
print("The distance between Reference_Image_2 and Test Image is : {}".format(dist_test_ref_2))
>>> The distance between Reference_Image_2 and Test Image is : 137929.0223122023
```

Conclusion

Based on the above results we can see that the distance between our test image and our first reference

image is much less than the distance between our test and our second reference image which makes sense because both the test image and our first reference image are images of a Piegon while our second reference image is of a Peacock.

In the following tutorial, we learned how to use basic mathematics and little programming to build our own Image Similarity predictor with pretty decent results.

Result:

Thus the similarity based clustering algorithm is implemented.

EX.NO:7 IMPLEMENT CLUSTERING ALGORITHM FOR ANY DATA SET.

AIM:

To implement clustering algorithm for any data set.

ALGORITHM:

1.Initialization

2.Choose the number of clusters(k)

3.Randomly initialize z cluster centroids

4.Assign data points

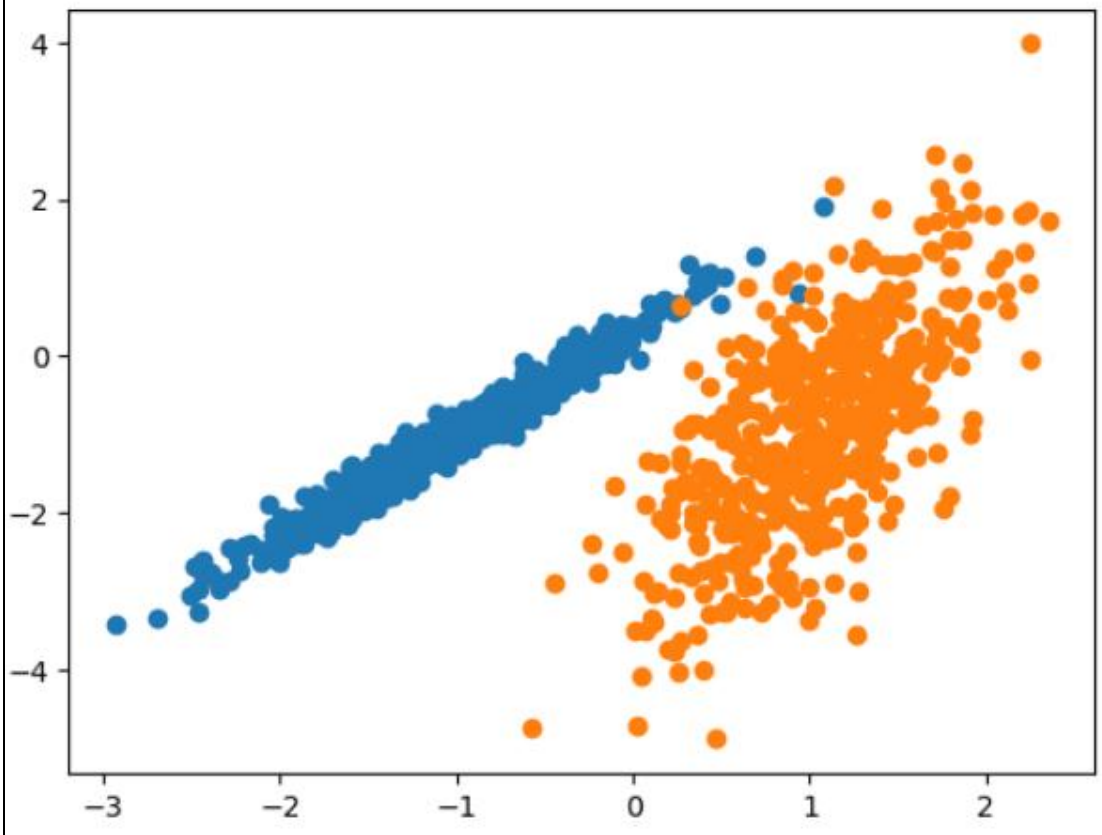
5.Update centroids

6.Repeat it until the algorithm gives the result.

PROGRAM:

```
# synthetic classification dataset
from numpy import where
from sklearn.datasets import make_classification
from matplotlib import pyplot
# define dataset
X, y = make_classification(n_samples=1000, n_features=2, n_informative=2, n_redundant=0, n_clusters_per_class=1, random_state=4)
# create scatter plot for samples from each class
for class_value in range(2):
    # get row indexes for samples with this class
    row_ix = where(y == class_value)
    # create scatter of these samples
    pyplot.scatter(X[row_ix, 0], X[row_ix, 1])
# show the plot
pyplot.show()
```

OUTPUT:



RESULT :

Thus the implementation of the clustering algorithm for any data set is executed.

Ex.No: 8.**APPLY REINFORCEMENT LEARNING**

Aim:

To apply reinforcement learning.

Introduction

One of the most fundamental question for scientists across the globe has been – "How to learn a new skill?". The desire to understand the answer is obvious – if we can understand this, we can enable human species to do things we might not have thought before. Alternately, we can train machines to do more "human" tasks and create true artificial intelligence.

While we don't have a complete answer to the above question yet, there are a few things which are clear. Irrespective of the skill, we first learn by interacting with the environment. Whether we are learning to drive a car or whether it an infant learning to walk, the learning is based on the interaction with the environment. Learning from interaction is the foundational underlying concept for all theories of learning and intelligence.

Re-inforcement Learning

Today, we will explore Reinforcement Learning – a goal-oriented learning based on interaction with environment. Reinforcement Learning is said to be the hope of true artificial intelligence. And it is rightly said so, because the potential that Reinforcement Learning possesses is immense.

Reinforcement Learning is growing rapidly, producing wide variety of learning algorithms for different applications. Hence it is important to be familiar with the techniques of reinforcement learning. If you are not familiar with reinforcement learning, I will suggest you to go through my previous article on introduction to reinforcement learning and the open source RL platforms.

Once you have an understanding of underlying fundamentals, proceed with this article. By the end of this article you will have a thorough understanding of Reinforcement Learning and its practical implementation.
P.S. *For implementation, we assume that you have basic knowledge of Python. If you don't know Python, you should first go through this tutorial*

Table of Content

1. Formulating a reinforcement learning problem
2. Comparison with other machine learning methodologies
3. Framework for solving Reinforcement learning problems
4. An implementation of Reinforcement Learning
5. Increasing the complexity
6. Peek into recent RL advancements
7. Additional Resources

1. Formulating a Reinforcement Learning Problem

Reinforcement Learning is learning what to do and how to map situations to actions. The end result is to maximize the numerical reward signal. The learner is not told which action to take, but instead must discover which action will yield the maximum reward. Let's understand this with a simple example below.

Consider an example of a child learning to walk.



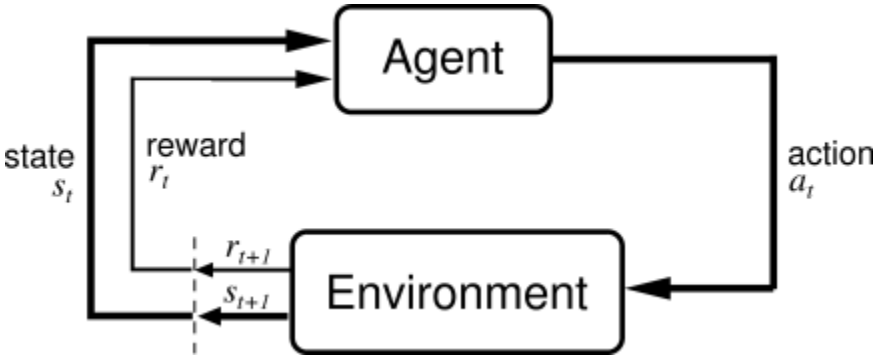
Here are the steps a child will take while learning to walk:

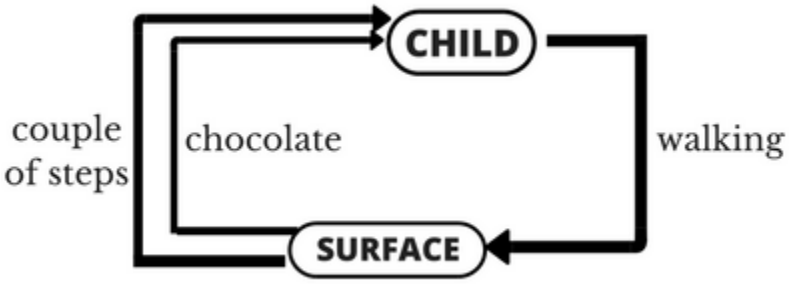
1. The first thing the child will observe is to **notice** how you are walking. You use two legs, taking a step at a time in order to walk. Grasping this concept, the child tries to replicate you.
2. But soon he/she will understand that before walking, the child has to stand up! This is a challenge that comes along while trying to walk. So now the child **attempts to get up,** staggering and slipping but still determinant to get up.
3. Then there's another challenge to cope up with. Standing up was easy, but to **remain still** is another task altogether! Clutching thin air to find support, the child manages to stay standing.
4. Now the real task for the child is to start walking. But it's easy to say than actually do it. There are so **many things to keep in mind**, like balancing the body weight, deciding which foot to put next and where to put it.

Sounds like a difficult task right? It actually is a bit challenging to get up and start walking, but you have become so use to it that you are not fazed by the task. But now you can get the gist of how difficult it is for a child.

Let's formalize the above example, the "problem statement" of the example is **to walk**, where **the child is an agent** trying to manipulate the **environment (which is the surface on which it walks)** by **taking actions (viz walking)** and he/she tries to go from one **state (viz each step he/she takes)** to another. The child gets a **reward (let's say chocolate)**

when he/she accomplishes a **submodule of the task (viz taking couple of steps)** and will not receive any chocolate **(a.k.a negative reward)** when he/she is not able towalk. This is a simplified description of a reinforcement learning problem.

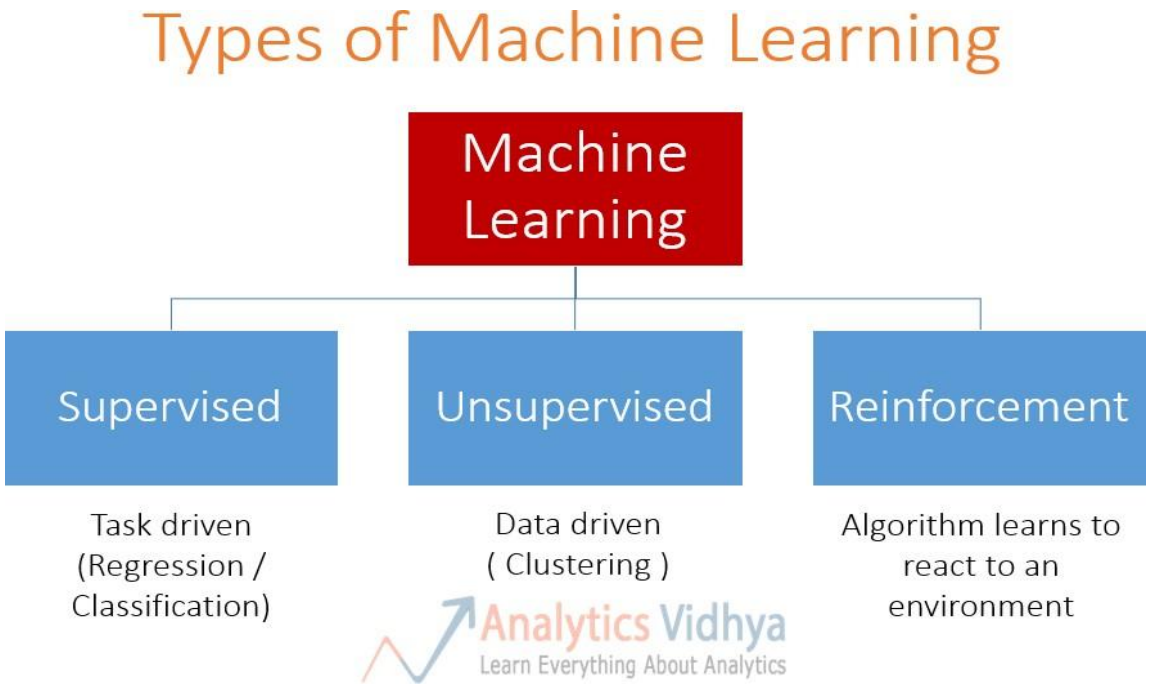




Here's a good introductory video on Reinforcement Learning.

2. Comparison with other machine learning methodologies

Reinforcement Learning belongs to a bigger class of machine learning algorithm. Below is the description of types of machine learning methodologies.



Let's see a comparison between RL and others:

- **Supervised vs Reinforcement Learning**: In supervised learning, there's an external "supervisor", which has knowledge of the environment and who shares it with the agent to complete the task. But there are some problems in which there are so many combinations of subtasks that the agent can perform to achieve the objective. So that creating a "supervisor" is almost impractical. For example, in a chess game, there are tens of thousands of moves that can be played. So creating a knowledge base that can be played is a tedious task. In these problems, it is more feasible to learn from one's own experiences and gain knowledge from them. This is the main difference that can be said of reinforcement learning and supervised learning. In both supervised and reinforcement learning, there is a mapping between input and output. But in reinforcement learning, there is a reward function which acts as a feedback to the agent as opposed to supervised learning.
- **Unsupervised vs Reinforcement Leanring**: In reinforcement learning, there's a mapping from input to output which is not present in unsupervised learning. In unsupervised learning, the main task is to find the underlying patterns rather than the mapping. For example, if the task is to suggest a news article to a user, an unsupervised learning algorithm will look at similar articles which the person has previously read and suggest anyone from them. Whereas a reinforcement learning algorithm will get constant feedback from the user by suggesting few news articles and then build a "knowledge graph" of which articles will the person like.

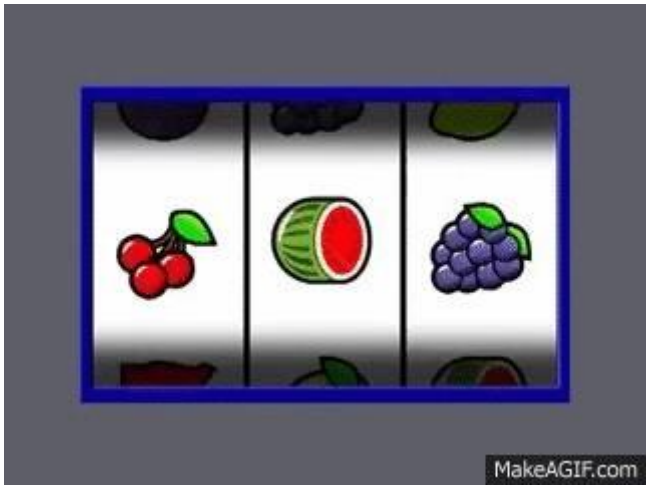
There is also a fourth type of machine learning methodology called **semi-supervised** learning, which is essentially a combination of supervised and unsupervised learning. It differs from reinforcement learning as similar to supervised and semi-supervised learning has direct mapping whereas reinforcement does not.

3. Framework for solving Reinforcement Learning Problems

To understand how to solve a reinforcement learning problem, let's go through a classic example of



reinforcement learning problem – Multi-Armed Bandit Problem. First, we would understand the fundamental problem of exploration vs exploitation and then go on to define the framework to solve RL problems.



Suppose you have many slot machines with random payouts. A slot machine would look something like this.

Now you want to do is get the maximum bonus from the slot machines as fast as possible. What would you do?

One naive approach might be to select only one slot machine and keep pulling the lever all day long. Sounds boring, but it may give you "some" payouts. With this approach, you might hit the jackpot (with a probability close to 0.00000….1) but most of the time you may just be sitting in front of the slot machine losing money. Formally, this can be defined as a **pure exploitation** approach. Is this the optimal choice? The answer is NO.

Let's look at another approach. We could pull a lever of each & every slot machine and pray to God that at least one of them would hit the jackpot. This is another naive approach which would keep you pulling levers all day long, but give you sub-optimal payouts. Formally this approach is a **pureexploration** approach.

Both of these approaches are not optimal, and we have to find a proper balance between them to get maximum reward. This is said to be **exploration vs exploitation dilemma** of reinforcement learning.

First, we formally define the framework for reinforcement learning problem and then list down the probable approaches to solve the problem.
Markov Decision Process:

The mathematical framework for defining a solution in reinforcement learning scenario is called **Markov Decision Process**. This can be designed as:

- Set of states, S
- Set of actions, A
- Reward function, R
- Policy, π
- Value, V

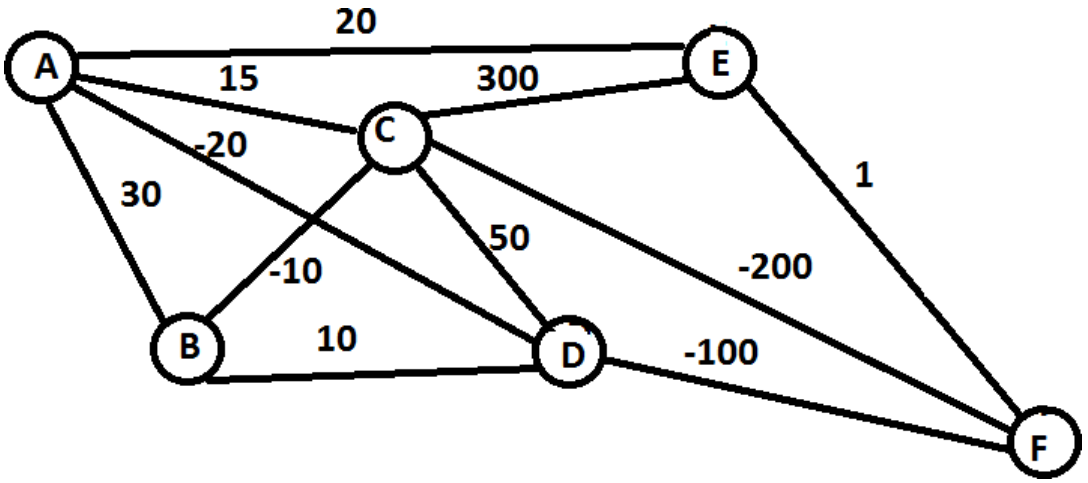
We have to take an action (A) to transition from our start state to our end state (*S*). In return getting rewards (R) for each action we take. Our actions can lead to a positive reward or negative reward.

The set of actions we took define our policy (π) and the rewards we get in return defines our value (V). Our task here is to maximize our rewards by choosing the correct policy. So we have to maximize for all possible values of *S* for a time t.

$$E(r_t \mid \pi, s_t)$$

Shortest Path Problem

Let me take you through another example to make it clear.



This is a representation of a shortest path problem. The task is to go from place A to place F, with as low cost as possible. The numbers at each edge between two places represent the cost taken to traverse the distance. The negative cost are actually some earnings on the way. We define Value is the total cumulative reward when you do a policy.

Here,

- The set of states are the nodes, viz {A, B, C, D, E, F}
- The action to take is to go from one place to other, viz {A -> B, C -> D, etc}
- The reward function is the value represented by edge, i.e. cost
- The policy is the "way" to complete the task, viz {A -> C -> F}

Now suppose you are at place A, the only visible path is your next destination and anything beyond that is not known at this stage (a.k.a observable space).

You can take a greedy approach and take the best possible next step, which is going from {A -> D} from a subset of {A -> (B, C, D, E)}. Similarly now you are at place D and want to go to place F, you can choose from {D -> (B, C, F)}. We see that {D -> F} has the lowest cost and hence we take that path.

So here, our policy was to take {A -> D -> F} and our Value is -120.

Congratulations! You have just implemented a reinforcement learning algorithm. This algorithm is known as **epsilon greedy,** which is literally a greedy approach to solving the problem. Now if you (the salesman) want to go from place A to place F again, you would always choose the same policy.
Other ways of travelling?

Can you guess which category does our policy belong to i.e. (pure exploration vs pure exploitation)?

Notice that the policy we took is not an optimal policy. We would have to "explore" a little bit to find the optimal policy. The approach which we took here is policy based learning, and our task is to find the optimal policy among all the possible policies. There are different ways to solve this problem, I'll briefly list down the major categories

- **Policy based,** where our focus is to find optimal policy
- **Value based,** where our focus is to find optimal value, i.e. cumulative reward
- **Action based,** where our focus is on what optimal actions to take at each step

I would try to cover in-depth reinforcement learning algorithms in future articles. Till then, you can refer to this paper on a survey of reinforcement learning algorithms.

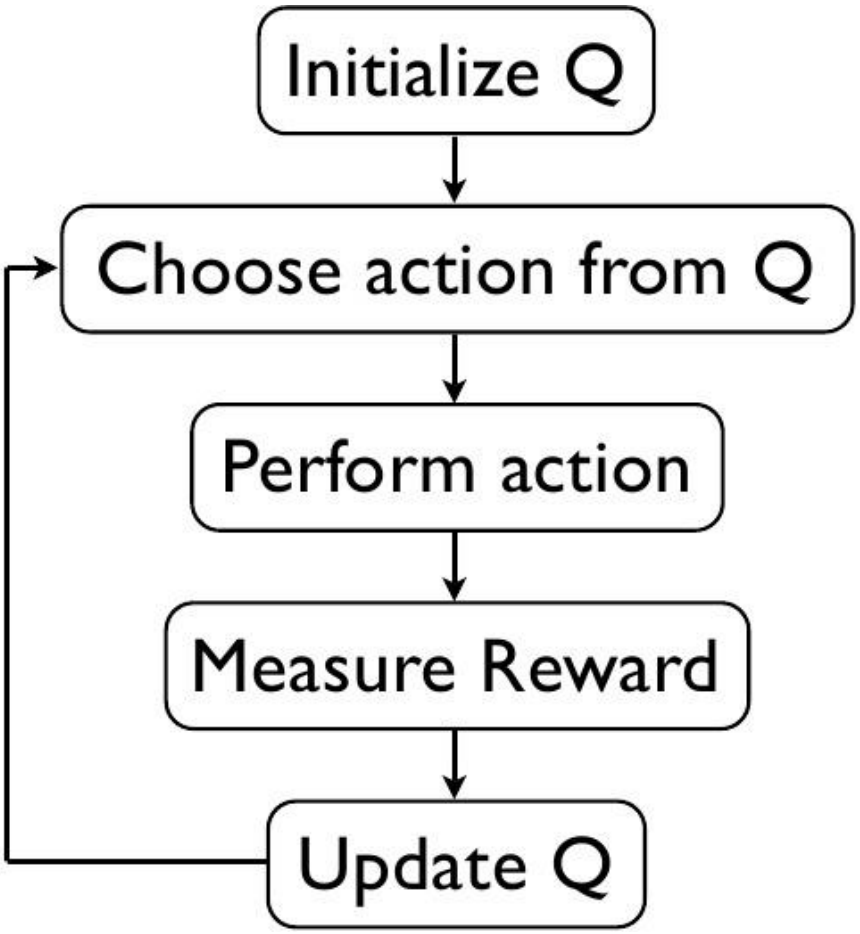
4. An implementation of Reinforcement Learning

We will be using Deep Q-learning algorithm. Q-learning is a policy based learning algorithm with the function approximator as a neural network. This algorithm was used by Google to beat humans at Atari games!

Let's see a pseudocode of Q-learning:

1. Initialize the Values table '**Q(s, a)'**.
2. Observe the current state '**s'**.
3. Choose an action **'a'** for that state based on one of the action selection policies (eg. epsilon greedy)
4. Take the action, and observe the reward **'r'** as well as the new state **'s'**.
5. Update the Value for the state using the observed reward and the maximum reward possible for the next state. The updating is done according to the formula and parameters described above.
6. Set the state to the new state, and repeat the process until a terminal state is reached.

A simple description of Q-learning can be summarized as follows:



We will first see what Cartpole problem is then go on to coding up a solution

When I was a kid, I remember that I would pick a stick and try to balance it on one hand. Me and my friends used to have this competition where whoever balances it for more time would get a "reward", a chocolate!

Here's a short video description of a real cart-pole system
Let's code it up!
To setup our code, we need to first install a few things,
Step 1: Install keras-rl library

From terminal, run the following commands:

```
git clone https://github.com/matthiasplappert/keras-rl.git
cd keras-rl
python setup.py install
```

Step 2: Install dependencies for CartPole environment

Assuming you have pip installed, you need to install the following libraries

```
pip install h5py
pip install gym
```

Step 3: lets get started!

First we have to import modules that are necessary

```
import numpy as np
import gym

from keras.models import Sequential
from keras.layers import Dense, Activation, Flatten
from keras.optimizers import Adam

from rl.agents.dqn import DQNAgent
from rl.policy import EpsGreedyQPolicy
from rl.memory import SequentialMemory
```

Then set the relevant variables

```
ENV_NAME = 'CartPole-v0'
```

```
# Get the environment and extract the number of actions available in the Cartpole problem
env = gym.make(ENV_NAME)
np.random.seed(123)
env.seed(123)
nb_actions = env.action_space.n
```

Next, we build a very simple single hidden layer neural network model.

```
model = Sequential()
model.add(Flatten(input_shape=(1,) + env.observation_space.shape))
model.add(Dense(16))
model.add(Activation('relu'))
model.add(Dense(nb_actions))
model.add(Activation('linear'))
print(model.summary())
```

Next, we configure and compile our agent. We set our policy as Epsilon Greedy and we also set our memory as Sequential Memory because we want to store the result of actions we performed and the rewards we get for each action.

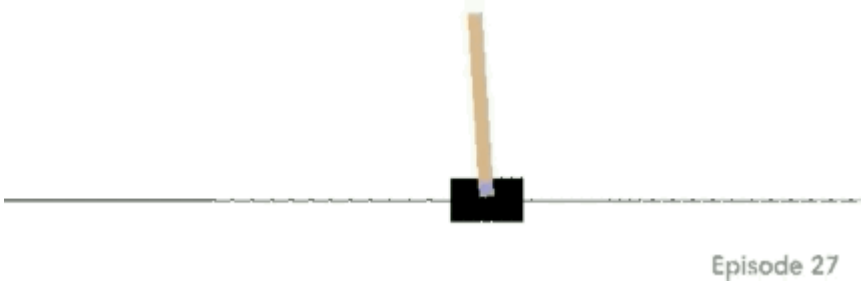
Now we test our reinforcement learning model

```
dqn.test(env, nb_episodes=5, visualize=True)
```

```
memory = SequentialMemory(limit=50000, window_length=1)
dqn = DQNAgent(model=model, nb_actions=nb_actions, memory=memory, nb_steps_warmup=10,
target_model_update=1e-2, policy=policy)
dqn.compile(Adam(lr=1e-3), metrics=['mae'])


# Okay, now it's time to learn something! We visualize the training here for show, but this slows down training quite a lot.
dqn.fit(env, nb_steps=5000, visualize=True, verbose=2)
```

This will be the output of our model:



And Voila! You have just built a reinforcement learning bot!

5. Increasing the complexity

Now that you have seen a basic implementation of Re-inforcement learning, let us start moving towards a few more problems, increasing the complexity little bit every time.

Problem – Towers of Hanoi



For those, who don't know the game – it was invented in 1883 and consists of 3 rods along with a number of sequentially-sized disks (3 in the figure above) starting at the leftmost rod. The objective is to move all

the disks from the leftmost rod to the rightmost rod **with the least number of moves**. (You can read more on wikipedia)

If we have to map this problem, let us start with states:

- **Starting state** – All 3 disks in leftmost rod (in order 1, 2 and 3 from top to bottom)
- **End State** – All 3 disks in rightmost rod (in order 1, 2 and 3 from top to bottom)

All possible states:

Here are our 27 possible states:

All disks in a rod	One disk in a Rod	(13) disks in a rod	(23) disks in a rod	(12) disks in a rod
(123)**	321	(13)2*	(23)1*	(12)3*
(123)	312	(13)*2	(23)*1	(12)*3
**(123)	231	2(13)*	1(23)*	3(12)*
	132	*(13)2	*(23)1	*(12)3
	213	2*(13)	1*(23)	3*(12)
	123	*2(13)	*1(23)	*3(12)

Where (12)3* represents disks 1 and 2 in leftmost rod (top to bottom) 3 in middle rod and * denotes an empty rightmost rod

Numerical Reward:

Since we want to solve the problem in least number of steps, we can attach a reward of -1 to each step.

Policy:
Now, without going in any technical details, we can map possible transitions between above states. For example (123)** -> (23)1* with reward -1. It can also go to (23)*1

If you can now see a parallel, each of these 27 states mentioned above can represent a graph similar to that of shortest path algorithm above and we can find the most optimal solutions by experimenting various states and paths.

Problem – 3 x 3 Rubix Cube

While I can solve this for you as well, I would want you to do this by yourself. Follow the same line of thought I used above and you should be good.

Start by defining the Starting state and the end state. Next, define all possible states and their transitions along with reward and policy. Finally, you should be able to create a solution for solving a rubix cube using the same approach.

6. A Peek into Recent Advancements in Reinforcement Learning

As you would realize that the complexity of this Rubix Cube is many folds higher than the Towers of Hanoi. You can also understand how the possible number of options have increased in number. Now, think of number of states and options in a game of Chess and then in Go! Google DeepMind recently created a deep reinforcement learning algorithm which defeated Lee Sedol!

With the recent success in Deep Learning, now the focus is slowly shifting to applying deep learning to solve reinforcement learning problems. The news recently has been flooded with the defeat of Lee Sedol by a deep reinforcement learning algorithm developed by Google DeepMind. Similar breakthroughs are being seen in video games, where the algorithms developed are achieving human-level accuracy and



beyond. Research is still at par, with both industrial and academic masterminds working together to accomplish the goal of building better self-learning robots

Source

Some major domains where RL has been applied are as follows:

- Game Theory and Multi-Agent Interaction
- Robotics
- Computer Networking
- Vehicular Navigation
- Medicine and
- Industrial Logistic.

There are so many things unexplored and with the current craze of deep learning applied to reinforcement learning, there certainly are breakthroughs incoming!

https://www.analyticsvidhya.com/blog/2017/01/introduction-to-reinforcement-learning-implementation/

Result:

Thus the reinforcement learning is applied.

Ex.No: 9.IMPLEMENT AND DEMONSTRATE THE CANDIDATE-ELIMINATION ALGORITHM

Aim:

To implement and demonstrate the candidate-elimination algorithm.

Candidate Elimination Algorithm Machine Learning

```
For each training example d, do:
    If d is positive example
        Remove from G any hypothesis h inconsistent with d
        For each hypothesis s in S not consistent with d:
            Remove s from S
            Add to S all minimal generalizations of s consistent with d and having a generalization in G
            Remove from S any hypothesis with a more specific h in S
    If d is negative example
        Remove from S any hypothesis h inconsistent with d
        For each hypothesis g in G not consistent with d:
            Remove g from G
            Add to G all minimal specializations of g consistent with d and having a specialization in S
            Remove from G any hypothesis having a more general hypothesis in G
```

Python Program to Implement and Demonstrate FIND-S Algorithm

```
import numpy as np
import pandas as pd


data = pd.read_csv(path+'/enjoysport.csv')
```

```
concepts = np.array(data.iloc[:,0:-1])
print("\nInstances are:\n",concepts)
target = np.array(data.iloc[:,-1])
print("\nTarget Values are: ",target)


def learn(concepts, target):
    specific_h = concepts[0].copy()
    print("\nInitialization of specific_h and genearal_h")
    print("\nSpecific Boundary: ", specific_h)
    general_h = [["?" for i in range(len(specific_h))] for i in range(len(specific_h))]
    print("\nGeneric Boundary: ",general_h)


    for i, h in enumerate(concepts):
        print("\nInstance", i+1 , "is ", h)
        if target[i] == "yes":
            print("Instance is Positive ")
            for x in range(len(specific_h)):
                if h[x]!= specific_h[x]:
                    specific_h[x] ='?'
                    general_h[x][x] ='?'
```

```
        if target[i] == "no":
            print("Instance is Negative ")
            for x in range(len(specific_h)):
                if h[x]!= specific_h[x]:
                    general_h[x][x] = specific_h[x]
                else:
                    general_h[x][x] = '?'

        print("Specific Bundary after ", i+1, "Instance is ", specific_h)
        print("Generic Boundary after ", i+1, "Instance is ", general_h)
        print("\n")

    indices = [i for i, val in enumerate(general_h) if val == ['?', '?', '?', '?', '?', '?']]
    for i in indices:
        general_h.remove(['?', '?', '?', '?', '?', '?'])
    return specific_h, general_h


s_final, g_final = learn(concepts, target)


print("Final Specific_h: ", s_final, sep="\n")
print("Final General_h: ", g_final, sep="\n")
```

Output:

Instances are:
[[‘sunny’ ‘warm’ ‘normal’ ‘strong’ ‘warm’ ‘same’]
[‘sunny’ ‘warm’ ‘high’ ‘strong’ ‘warm’ ‘same’]

Dataset:

EnjoySport Dataset is saved as .csv (comma separated values) file in the current working directory otherwise use the complete path of the dataset set in the program:

See also **Advantages and Disadvantages of Regression Model**

sky	airtemp	humidity	wind	water	forcast	enjoysport
sunny	warm	normal	strong	warm	same	yes
sunny	warm	high	strong	warm	same	yes
rainy	cold	high	strong	warm	change	no
sunny	warm	high	strong	cool	change	yes

[‘rainy’ ‘cold’ ‘high’ ‘strong’ ‘warm’ ‘change’]
[‘sunny’ ‘warm’ ‘high’ ‘strong’ ‘cool’ ‘change’]]
Target Values are: [‘yes’ ‘yes’ ‘no’ ‘yes’] Initialization
of specific_h and genearal_h
Specific Boundary: [‘sunny’ ‘warm’ ‘normal’ ‘strong’ ‘warm’ ‘same’]

Generic Boundary: [[‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’]]

Instance 1 is [‘sunny’ ‘warm’ ‘normal’ ‘strong’ ‘warm’ ‘same’] Instance is Positive Specific
Bundary after 1 Instance is [‘sunny’ ‘warm’ ‘normal’ ‘strong’ ‘warm’ ‘same’]
Generic Boundary after 1 Instance is [[‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’]]

Result:
Thus the candidate-elimination algorithm is implemented.

Ex.No: 10.IMPLEMENT K-NEAREST NEIGHBOUR ALGORITHM

Aim:

To implement K-Nearest neighbour algorithm.

K-Nearest Neighbors

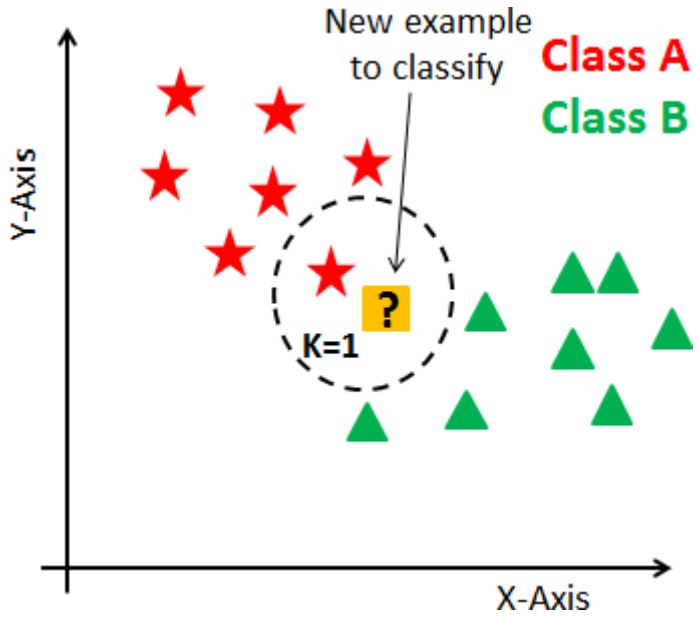
KNN is a non-parametric and lazy learning algorithm. Non-parametric means there is no assumption for underlying data distribution. In other words, the model structure determined from the dataset. This will be very helpful in practice where most of the real-world datasets do not follow mathematical theoretical assumptions. The lazy algorithm means it does not need any training data points for model generation.

All training data used in the testing phase. This makes training faster and the testing phase slower and costlier. The costly testing phase means time and memory. In the worst case, KNN needs more time to scan all data points, and scanning all data points will require more memory for storing training data.

For more such tutorials, projects, and courses visit DataCamp:

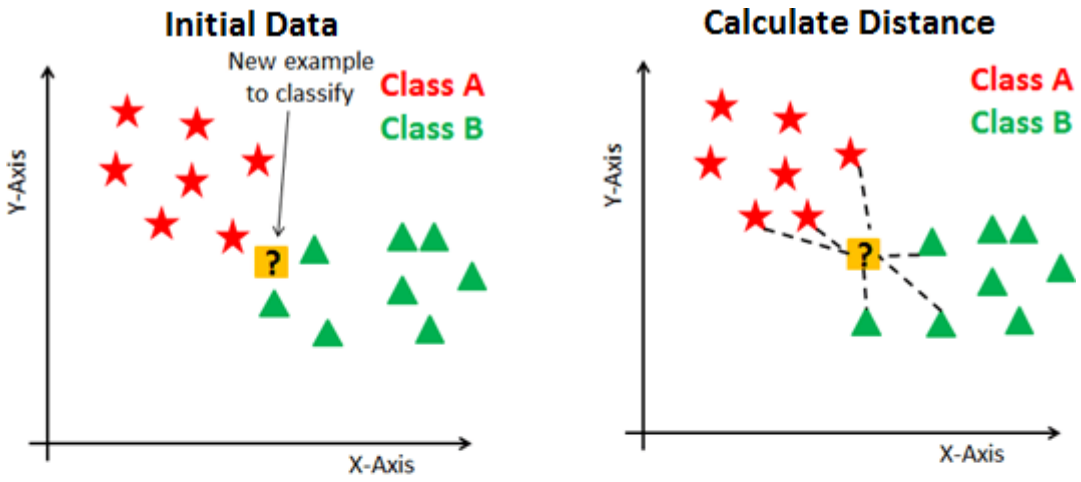
How does the KNN algorithm work?

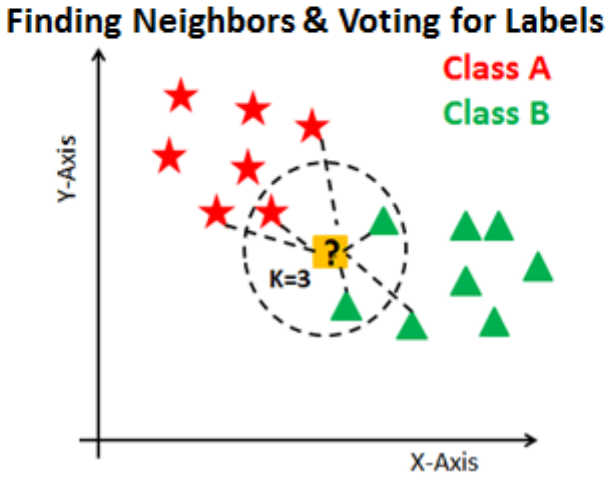
In KNN, K is the number of nearest neighbors. The number of neighbors is the core deciding factor. K isgenerally an odd number if the number of classes is 2. When K=1, then the algorithm is known as the nearest neighbor algorithm. This is the simplest case. Suppose P1 is the point, for which label needs to predict. First, you find the closest point to P1 and then the label of the nearest point assigned to P1.



Suppose P1 is the point, for which label needs to predict. First, you find the k closest point to P1 and thenclassify points by majority vote of its k neighbors. Each object votes for their class and the class with themost votes is taken as the prediction. For finding closest similar points, you find the distance between points using distance measures such as Euclidean distance, Hamming distance, Manhattan distance, and Minkowski distance. KNN has the following basic steps:

- Calculate distance
- Find closest neighbors
- Vote for labels





Eager Vs. Lazy Learners

Eager learners mean when given training points will construct a generalized model before performing prediction on given new points to classify. You can think of such learners as being ready, active, and eager to classify unobserved data points.

Lazy Learning means there is no need for learning or training of the model and all of the data points used at the time of prediction. Lazy learners wait until the last minute before classifying any data point. Lazy learners stores merely the training dataset and waits until classification needs to perform. Only when it sees the test tuple does it perform generalization to classify the tuple based on its similarity to the stored training tuples. Unlike eager learning methods, lazy learners do less work in the training phase and more work in the testing phase to make a classification. Lazy learners are also known as instance-based learners because lazy learners store the training points or instances, and all learning is based on instances.

Curse of Dimensionality

KNN performs better with a lower number of features than a large number of features. You can say that when the number of features increases than it requires more data. An increase in dimension also leads to the problem of overfitting. To avoid overfitting, the needed data will need to grow exponentially as you increase the number of dimensions. This problem of higher dimension is known as the Curse of Dimensionality.

To deal with the problem of the curse of dimensionality, you need to perform principal component analysis before applying any machine learning algorithm, or you can also use the feature selection approach.

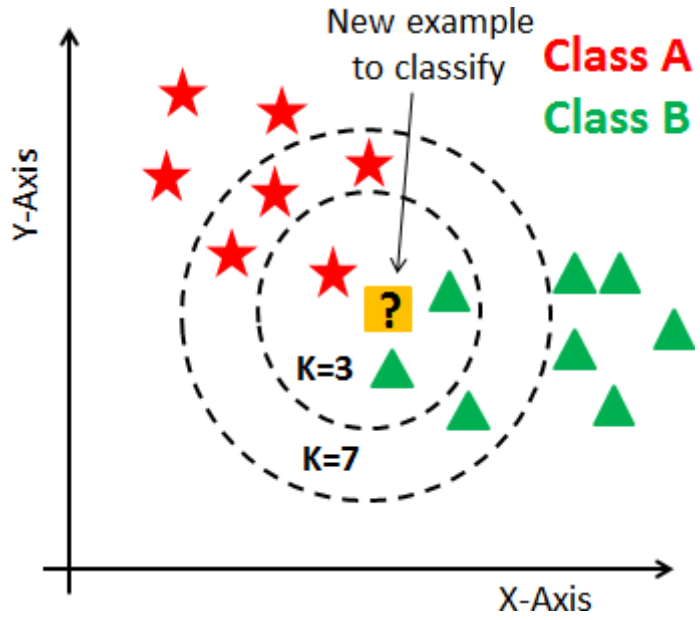
Research has shown that in large dimensions Euclidean distance is not useful anymore. Therefore, you can prefer other measures such as cosine similarity, which get decidedly less affected by a high dimension.

How do you decide the number of neighbors in KNN?

Now, you understand the KNN algorithm working mechanism. At this point, the question arises that How to choose the optimal number of neighbors? And what are its effects on the classifier? The number of neighbors(K) in KNN is a hyperparameter that you need to choose at the time of model building. You can think of K as a controlling variable for the prediction model.

Research has shown that no optimal number of neighbors suits all kinds of data sets. Each dataset has its own requirements. In the case of a small number of neighbors, the noise will have a higher influence on the result, and a large number of neighbors make it computationally expensive. Research has also shown that a small number of neighbors are the most flexible fit which will have low bias but the high variance and a large number of neighbors will have a smoother decision boundary which means lower variance but higher bias.

Generally, Data scientists choose an odd number if the number of classes is even. You can also check by generating the model on different values of k and check their performance. You can also try the Elbow method here.



Building KNN Classifier in Scikit-learn

Defining dataset

Let's first create your own dataset. Here you need two kinds of attributes or columns in your data: Feature and label. The reason for the two types of columns is the "supervised nature of KNN algorithm".

```
# Assigning features and label variables
# First Feature
weather= 'Sunny','Sunny','Overcast','Rainy','Rainy','Rainy','Overcast','Sunny','Sunny',
'Rainy','Sunny','Overcast','Overcast','Rainy'
# Second Feature
```

```
temp=['Hot','Hot','Hot','Mild','Cool','Cool','Cool','Mild','Cool','Mild','Mild','Mild','Hot','Mild'] # Label or target varible
play=['No','No','Yes','Yes','Yes','No','Yes','No','Yes','Yes','Yes','Yes','Yes','No']
```

In this dataset, you have two features (weather and temperature) and one label(play).

Encoding data columns

Various machine learning algorithms require numerical input data, so you need to represent categorical columns in a numerical column.

In order to encode this data, you could map each value to a number. e.g. Overcast:0, Rainy:1, and Sunny:2. This process is known as label encoding, and sklearn conveniently will do this for you using Label Encoder.

```
# Import LabelEncoder
from sklearn import preprocessing

#creating labelEncoder
le = preprocessing.LabelEncoder()

# Converting string labels into numbers.
weather_encoded=le.fit_transform(weather)
print(weather_encoded)
```

Output: [2 2 0 1 1 1 0 2 2 1 2 0 0 1]

Here, you imported the preprocessing module and created the Label Encoder object. Using this LabelEncoder object, you can fit and transform the "weather" column into the numeric column.

Similarly, you can encode temperature and label it into numeric columns.

```
# converting string labels into numbers
temp_encoded=le.fit_transform(temp)

label=le.fit_transform(play)
```

Combining Features

Here, you will combine multiple columns or features into a single set of data using "zip" function

```
#combinig weather and temp into single listof tuples
features=list(zip(weather_encoded,temp_encoded))
```

Generating Model

Let's build the KNN classifier model.

First, import the KNeighborsClassifier module and create a KNN classifier object by passing the argument number of neighbors in KNeighborsClassifier() function.

Then, fit your model on the train set using fit() and perform prediction on the test set using predict().

```
from sklearn.neighbors import KNeighborsClassifier
# Create KNN Classifier
model = KNeighborsClassifier(n_neighbors=3)
# Train the model using the training sets
```

```
model fit features label

#Predict Output
predicted= model predict [ 0 2 ]

# 0:Overcast, 2:Mild
print predicted
```

Output: [1]

In the above example, you have given input [0,2], where 0 means Overcast weather and 2 means Mild temperature. The model predicts [1], which means play.

KNN with Multiple Labels

Till now, you have learned How to create a KNN classifier for two in python using scikit-learn. Now you will learn about KNN with multiple classes.

In the model the building part, you can use the wine dataset, which is a very famous multi-class classification problem. This data is the result of a chemical analysis of wines grown in the same region in Italy using three different cultivars. The analysis determined the quantities of 13 constituents found in each of the three types of wines.

The dataset comprises 13 features (‘alcohol’, ‘malic_acid’, ‘ash’, ‘alcalinity_of_ash’, ‘magnesium’, ‘total_phenols’, ‘flavanoids’, ‘nonflavanoid_phenols’, ‘proanthocyanins’, ‘color_intensity’, ‘hue’, ‘od280/od315_of_diluted_wines’, ‘proline’) and a target (type of cultivars).

This data has three types of cultivar classes: ‘class_0’, ‘class_1’, and ‘class_2’. Here, you can build a model to classify the type of cultivar. The dataset is available in the scikit-learn library, or you can also download it from the UCI Machine Learning Library.

Loading Data

Let’s first load the required wine dataset from scikit-learn datasets.

```
#Import scikit-learn dataset library
from sklearn import datasets#Load dataset
wine = datasets load_wine
```

Exploring Data

After you have loaded the dataset, you might want to know a little bit more about it. You can check features and target names.

```
# print the names of the features
print wine feature_names

# print the label species(class_0, class_1, class_2)
print wine target_names
```

Output:

['alcohol', 'malic_acid', 'ash', 'alcalinity_of_ash', 'magnesium', 'total_phenols', 'flavanoids', 'nonflavanoid_phenols', 'proanthocyanins', 'color_intensity', 'hue', 'od280/od315_of_diluted_wines', 'proline']

['class_0' 'class_1' 'class_2']

Let’s check top 5 records of the feature set.

```
# print the wine data (top 5 records)
print wine data[0:5])
```

Output:

```
[[ 1.42300000e+01 1.71000000e+00 2.43000000e+00 1.56000000e+01
   1.27000000e+02  2.80000000e+00  3.06000000e+00  2.80000000e-01
   2.29000000e+00  5.64000000e+00   1.04000000e+00   3.92000000e+00
   1.06500000e+03]
 [ 1.32000000e+01 1.78000000e+00 2.14000000e+00 1.12000000e+01
   1.00000000e+02  2.65000000e+00  2.76000000e+00  2.60000000e-01
   1.28000000e+00  4.38000000e+00   1.05000000e+00   3.40000000e+00
   1.05000000e+03]
 [ 1.31600000e+01 2.36000000e+00 2.67000000e+00 1.86000000e+01
   1.01000000e+02  2.80000000e+00  3.24000000e+00  3.00000000e-01
   2.81000000e+00  5.68000000e+00   1.03000000e+00   3.17000000e+00
   1.18500000e+03]
 [ 1.43700000e+01 1.95000000e+00 2.50000000e+00 1.68000000e+01
   1.13000000e+02  3.85000000e+00  3.49000000e+00  2.40000000e-01
   2.18000000e+00   7.80000000e+00   8.60000000e-01   3.45000000e+00
   1.48000000e+03]
 [ 1.32400000e+01 2.59000000e+00 2.87000000e+00 2.10000000e+01
   1.18000000e+02  2.80000000e+00  2.69000000e+00  3.90000000e-01
   1.82000000e+00  4.32000000e+00   1.04000000e+00   2.93000000e+00
   7.35000000e+02]]
```

Let’s check the records of the target set.

```
# print the wine labels (0:Class_0, 1:Class_1, 2:Class_3)
print wine target
```

Output:

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2]
```

Let’s explore it for a bit more. You can also check the shape of the dataset using shape.

```
# print data(feature)shape
print wine data shape

# print target(or label)shape
print wine target shape
```

(178, 13)

```
(178,)
```

Splitting Data

To understand model performance, dividing the dataset into a training set and a test set is a good strategy.
Let's split dataset by using function train_test_split(). You need to pass 3 parameters features, target, and test_set size. Additionally, you can use random_state to select records randomly.

```
# Import train_test_split function
from sklearn model_selection import train_test_split

# Split the dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(wine data, wine target, test_size=0.3) # 70% training and 30% test
```

Generating Model for K=5

Let's build the KNN classifier model for k=5.

```
#Import knearest neighbors Classifier model
from sklearn neighbors import KNeighborsClassifier

# Create KNN Classifier
knn = KNeighborsClassifier(n_neighbors=5)

# Train the model using the training sets
knn.fit(X_train, y_train)

# Predict the response for test dataset
y_pred = knn.predict(X_test)
```

Model Evaluation for k=5

Let's estimate, how accurately the classifier or model can predict the type of cultivars.
Accuracy can be computed by comparing actual test set values and predicted values.

```
#Import scikit-learn metrics module for accuracy calculation
from sklearn import metrics

# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

Output:

```
Accuracy: 0.685185185185
```

Well, you got a classification rate of 68.51%, considered as good accuracy.
For further evaluation, you can also create a model for a different number of neighbors.
Re-generating Model for K=7
Let's build the KNN classifier model for k=7.

```
#Import knearest neighbors Classifier model
from sklearn neighbors import KNeighborsClassifier

#Create KNN Classifier
knn = KNeighborsClassifier(n_neighbors=7)

#Train the model using the training sets
```

```
knn fit(X_train, y_train)

#Predict the response for test dataset
y_pred = knn.predict(X_test)
```

Model Evaluation for k=7

Let's again estimate, how accurately the classifier or model can predict the type of cultivars for k=7.

```
#Import scikit-learn metrics module for accuracy calculation
from sklearn import metrics

# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

Output:

Accuracy: 0.777777777778

Well, you got a classification rate of 77.77%, considered as good accuracy.

Here, you have increased the number of neighbors in the model, and accuracy got increased. But, this is not necessary for each case that an increase in many neighbors increases the accuracy. For a more detailed understanding of it, you can refer to the section "How to decide the number of neighbors?" of this tutorial.

Result:

Thus the knn algorithm is implemeted

Ex.No: 11.IMPLEMENT THE NON-PARAMETRIC LOCALLY WEIGHTED REGRESSION ALGORITHM

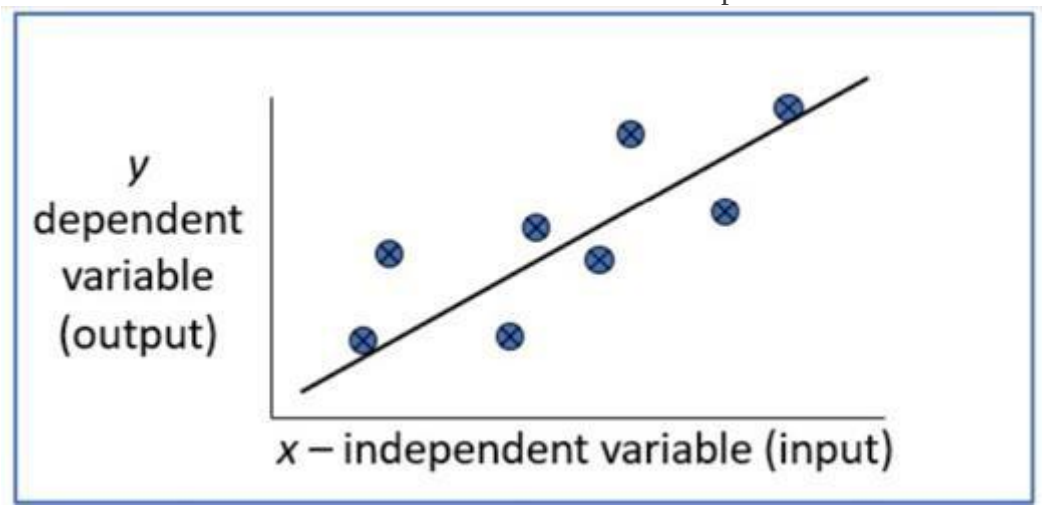
Aim:

Python Program to Implement the Locally Weighted Regression Algorithm

Locally Weighted Regression Algorithm

Regression:

- Regression is a technique from statistics that are used to predict values of the desired target quantity when the target quantity is continuous.
    - In regression, we seek to identify (or estimate) a continuous variable y associated with a given input vector x.
        - y is called the dependent variable.
        - x is called the independent variable.



Loess/Lowess Regression:

Loess regression is a nonparametric technique that uses local weighted regression to fit a smooth curve through points in a scatter plot.



Lowess Algorithm:

Locally weighted regression is a very powerful nonparametric model used in statistical learning.

Given a dataset X, y, we attempt to find a model parameter β(x) that minimizes residual sum of weighted squared errors.

The weights are given by a kernel function (k or w) which can be chosen arbitrarily

See also Advantages and Disadvantages of Regression Model

Algorithm

1. Read the Given data Sample to X and the curve (linear or non linear) to Y

2. Set the value for Smoothening parameter or Free parameter say τ

3. Set the bias /Point of interest set x0 which is a subset of X

4. Determine the weight matrix using :

$$w(x, x_o) = e^{-\frac{(x-x_o)^2}{2\tau^2}}$$

5. Determine the value of model term parameter β using:

$$\hat{\beta}(x_o) = (X^T W X)^{-1} X^T W y$$

6. Prediction = x0*β

Python Program to Implement and Demonstrate Locally Weighted Regression Algorithm

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


def kernel(point, xmat, k):
    m,n = np.shape(xmat)
    weights = np.mat(np1.eye((m)))
    for j in range(m):
        diff = point - X[j]
        weights[j,j] = np.exp(diff*diff.T/(-2.0*k**2))
    return weights


def localWeight(point, xmat, ymat, k):
    wei = kernel(point,xmat,k)
    W = (X.T*(wei*X)).I*(X.T*(wei*ymat.T))
    return W


def localWeightRegression(xmat, ymat, k):
    m,n = np.shape(xmat)
    ypred = np.zeros(m)
```

```
    for i in range(m):
        ypred[i] = xmat[i]*localWeight(xmat[i],xmat,ymat,k)
    return ypred


# load data points
data = pd.read_csv('10-dataset.csv')
bill = np.array(data.total_bill)
tip = np.array(data.tip)


#preparing and add 1 in bill
mbill = np.mat(bill)
mtip = np.mat(tip)


m= np.shape(mbill)[1]
one = np.mat(np1.ones(m))
X = np.hstack((one.T,mbill.T))


#set k here
ypred = localWeightRegression(X,mtip,0.5)
SortIndex = X[:,1].argsort(0)
xsort = X[SortIndex][:,0]
```

```
fig = plt.figure()

ax = fig.add_subplot(1,1,1)

ax.scatter(bill,tip, color='green')

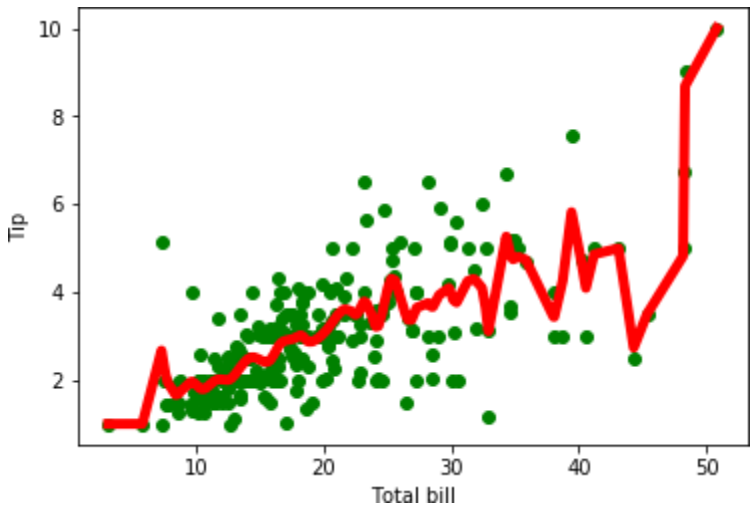
ax.plot(xsort[:,1],ypred[SortIndex], color = 'red', linewidth=5)

plt.xlabel('Total bill')

plt.ylabel('Tip')

plt.show();
```

Output



Click here to download restaurant bill dataset
https://www.vtupulse.com/machine-learning/locally-weighted-regression-algorithm-in-python/

Result :

Thus the algorithm is implemented