



REFERENCE MANUAL

संदर्भ संहिता

Short Course on
“Python for Artificial
Intelligence in
Agriculture”
(02 – 11 February 2023)

लघु पाठ्यक्रम
कृषि में कृत्रिम बुद्धिमत्ता के लिए
पायथन

Training Coordinators:

Dr Sudeep

Dr Sanchita Naha

Dr Md Ashraful Haque



भा. कृ. अनु. प. - भारतीय कृषि सांख्यिकी अनुसंधान संस्थान,
लाइब्रेरी एवेन्यू, पूसा, नई दिल्ली - ११००१२



ICAR – INDIAN AGRICULTURAL STATISTICS RESEARCH INSTITUTE,
LIBRARY AVENUE, PUSA, NEW DELHI – 110012



कृषि में कृत्रिम बुद्धिमत्ता के लिए पायथन
Python for Artificial Intelligence in Agriculture

Under the aegis of
Agricultural Education Division, ICAR

02 – 11 February, 2023

Reference Manual

Course Coordinators:
Dr. Sudeep, Dr. Sanchita Naha, Dr. Md Ashraful Haque

ICAR- Indian Agricultural Statistics Research Institute
Library Avenue, New Delhi – 110012

Editors:

Dr. Sudeep

Dr. Sanchita Naha

Dr. Md Ashraful Haque

Disclaimer: The information contained in this reference manual has been taken from various web resources. Respective URLs are mentioned in the content.

Citation:

Sudeep, Naha, S., Haque, M. A. (2023). **Python for Artificial Intelligence in Agriculture**. Reference Manual, ICAR-Indian Agricultural Statistics Research Institute, New Delhi.


आमुख

भा.कृ.अनु.प.- भारतीय कृषि सांख्यिकी अनुसंधान संस्थान (भा.कृ.अनु.प.- भा.कृ.सां.अनु.सं.) सांख्यिकीय विज्ञान, कृषि सांख्यिकी, संगणक अनुप्रयोग और जैव सूचना विज्ञान) में प्रासंगिकता का एक प्रमुख संस्थान है और कृषि अनुसंधान और सूचित नीति निर्णय लेने की गुणवत्ता को समृद्ध करने के लिए कृषि विज्ञान में उनका विवेकपूर्ण संलयन है। 1930 में अपनी स्थापना के बाद से, तत्कालीन इंपीरियल काउंसिल ऑफ एग्रीकल्चरल रिसर्च के एक छोटे से सांख्यिकीय खंड के रूप में, संस्थान का कद बढ़ा है और इसने राष्ट्रीय और अंतरराष्ट्रीय स्तर पर अपनी उपस्थिति दर्ज कराई है। संस्थान बहुत सक्रिय रूप से सलाहकार सेवा का अनुसरण कर रहा है जिसने संस्थान को राष्ट्रीय कृषि अनुसंधान और शिक्षा प्रणाली (एन.ए.आर.ई.एस.) और राष्ट्रीय कृषि सांख्यिकी प्रणाली (एन.ए.एस.एस.) दोनों में अपनी उपस्थिति दर्ज कराने में सक्षम बनाया है। कृषि अनुसंधान और नीति नियोजन की गुणवत्ता बढ़ाने के लिए उन्नत सांख्यिकीय तकनीकों और संस्थान में विकसित प्रयोगों के कुशल रचना का व्यापक रूप से उपयोग किया गया है। भा.कृ.अनु.प.- भा.कृ.सां.अनु.सं. कृषि में आधुनिक, अत्याधुनिक सूचना और संचार प्रौद्योगिकी (आई.सी.टी.) के हस्तक्षेप से संबंधित अनुसंधान, शिक्षण और प्रशिक्षण गतिविधियों में भी सक्रिय रूप से सम्मिलित है। राष्ट्रीय कृषि नवाचार परियोजना; एन.ए.आई.पी. के तहत भा.कृ.अनु.प., यानी भा.कृ.अनु.प. डेटा सेंटर; भा.कृ.अनु.प. डी.सी.), भा.कृ.अनु.प.- एन.ए.ए.आर.एम., हैदराबाद में डेटा रिकवरी सेंटर (कृषि मेघ) के तहत राष्ट्रीय कृषि उच्च शिक्षा परियोजना (एन.ए.एच.ई.पी.)।

कृत्रिम बुद्धिमत्ता (ए.आई.) की हालिया लहर ने डेटा साइंस के क्षेत्र में एक नया आयाम जोड़ा है। इन हालिया ए.आई. हस्तक्षेपों का लाभ उठाते हुए कृषि भी उसी गति से आगे बढ़ रही है। भा.कृ.अनु.प.- भा.कृ.सां.अनु.सं. अपने कई उच्च गुणवत्ता वाले परिणामों और उच्च प्रभाव वाले प्रकाशनों के साथ एन.ए.आर.ई.एस. के भीतर ए.आई. अनुसंधान करने वाले अग्रणी संस्थानों में से एक है। ए.आई. आधारित उपकरणों और तकनीकों को विकसित करने के लिए पायथन वैज्ञानिक समुदाय में सबसे लोकप्रिय प्रोग्रामिंग भाषा के रूप में उभरा है। इसलिए, इस कार्यक्रम को "कृषि में कृत्रिम बुद्धिमत्ता के लिए पाइथन" पर प्रशिक्षण प्रदान करने के लिए डिज़ाइन किया गया है ताकि प्रतिभागियों को आंकड़ों का विश्लेषण, आंकड़ों का प्रतिरूपण, उन्नत मशीन शिक्षा और गहन शिक्षा कलन विधि वेब ए पी आई आदि विकसित करने में सक्षम बनाया जा सके।

इस पाठ्यक्रम के संकाय में संगणक अनुप्रयोग और कृषि सांख्यिकी के क्षेत्र में सुस्थापित वैज्ञानिक शामिल हैं। संदर्भ पुस्तिका में व्याख्यान टिप्पणियां विषय की व्याख्या प्रदान करते हैं। मुझे उम्मीद है कि संदर्भ पुस्तिका प्रतिभागियों के लिए काफी उपयोगी होगी। मैं इस अवसर पर पूरे संकाय को एक अद्भुत काम करने के लिए धन्यवाद देता हूँ। मैं डॉ. सुदीप, डॉ. संचिता नाहा और डॉ. मोहम्मद अशरफुल हक, इस प्रशिक्षण कार्यक्रम के पाठ्यक्रम समन्वयक, को समय पर इस मूल्यवान दस्तावेज को लाने के लिए बधाई देना चाहते हैं। हम इस संदर्भ पुस्तिका को बेहतर बनाने के लिए प्रत्येक भागीदार के सुझावों की प्रतीक्षा कर रहे हैं।

दिनांक: 02 फरवरी 2023
पूसा, नई दिल्ली


डॉ. राजेंद्र प्रसाद
निदेशक, भा.कृ.अनु.प.-भा.कृ.सां.अनु.सं.


FOREWORD

ICAR-Indian Agricultural Statistics Research Institute (ICAR- IASRI) is a premier Institute of relevance in Statistical Sciences (Statistics, Computer Applications and Bioinformatics) and their judicious fusion in agricultural sciences for enriching quality of agricultural research and informed policy decision making. Ever since its inception in 1930, as a small Statistical Section of the then Imperial Council of Agricultural Research, the Institute has grown in stature and made its presence felt both nationally and internationally. The Institute has been very actively pursuing advisory service that has enabled the institute to make its presence felt both in National Agricultural Research and Education System (NARES) and National Agricultural Statistics System (NASS). The advanced statistical techniques and efficient design of experiments developed at the institute have been widely used for enhancing the quality of agricultural research and policy planning. ICAR-IASRI is also actively engaged in research, teaching and training activities related to modern, cutting-edge Information and Communication Technology (ICT) interventions in agriculture. The institute has many achievements in terms of establishing several IT infrastructures for ICAR, i.e., ICAR Data Centre (ICAR DC) under National Agricultural Innovation Project (NAIP), ICAR Data Recovery centre (Krishi Megh) in ICAR-NAARM, Hyderabad under National Agricultural Higher Education Project (NAHEP).

The recent wave of Artificial Intelligence (AI) has added a new dimension to the field of data science. Agriculture is also keeping up at the same pace taking advantage of these recent AI interventions. ICAR-IASRI is one of the pioneer institutes in conducting AI research within NARES with many of its high-quality outcomes and high impact publications. Python has emerged as the most popular programming language in the scientific community for developing AI- based tools and techniques. Therefore, this programme has been designed to impart training on “**Python for Artificial Intelligence in Agriculture**” to enable participants to perform data analysis, data modelling, develop advanced machine learning and deep learning algorithms, web APIs etc.

The faculty for this course comprises well-established scientists in the discipline of Computer Applications and Agricultural Statistics. The lecture notes in the reference manual provide an exposition of the subject. I hope that the reference manual will be quite useful to the participants. I take this opportunity to thank the entire faculty for doing a wonderful job. I wish to complement Dr. Sudeep, Dr. Sanchita Naha and Dr. Md Ashraful Haque, Course Coordinators of this training program, for bringing out this valuable document in time. We look forward to suggestions from each participant in improving this reference manual.

Date: 02.02.2023
New Delhi


Dr. Rajender Parsad
Director, ICAR-IASRI

प्रस्तावना

भा.कृ.अनु.प.-भारतीय कृषि सांख्यिकी अनुसंधान संस्थान सांख्यिकीय विज्ञान (कृषि सांख्यिकी, संगणक अनुप्रयोग, और जैव सूचना विज्ञान) में प्रासंगिकता का एक प्रमुख संस्थान है और कृषि अनुसंधान की गुणवत्ता को समृद्ध करने एवं नीतिगत निर्णय लेने के लिए कृषि विज्ञान में उनका विवेकपूर्ण संलयन है। संस्थान ने कृषि अनुसंधान के लिए उपयोगी विभिन्न सॉफ्टवेयर, वेब एप्लिकेशन, मोबाइल ऐप और अब कृत्रिम बुद्धिमत्ता (एआई) आधारित उपकरण, तकनीक और कार्यप्रणाली विकसित करने का नेतृत्व किया है।

एआई के हालिया उदय के बाद से, डेटा वैज्ञानिकों और अन्य डोमेन विशेषज्ञों को एआई प्रशिक्षण और संवेदीकरण की स्पष्ट रूप से आवश्यकता है। संस्थान के मुख्य अनुसंधान हितों में से एक कृषि के लिए एआई-आधारित उपकरणों और पद्धतियों का विकास है। इसके अलावा, आर, पायथन आदि जैसे सॉफ्टवेयर विभिन्न कृषि प्रयोगों से प्राप्त आंकड़ों के विश्लेषण में प्रमुख भूमिका निभा रहे हैं। भा.कृ.अनु.प.-भा.कृ.सां.अनु.सं. फरवरी 02-11, 2023 के दौरान भा.कृ.अनु.प. या एस.एयू./सी.एयू./आई.सी.ए.आर. वित्तपोषित केवीके के सभी वैज्ञानिक कर्मचारियों के लिए "कृषि में कृत्रिम बुद्धिमत्ता के लिए पाइथन" पर एक लघु पाठ्यक्रम आयोजित कर रहा है। इस लघु पाठ्यक्रम का उद्देश्य प्रतिभागियों को कृत्रिम बुद्धिमत्ता (ए.आई.) और मशीन लर्निंग (एम.एल.) अनुप्रयोगों को विकसित करने में उपयोग किए जाने वाले पायथन प्रोग्रामिंग से परिचित कराना है। हमारा उद्देश्य प्रतिभागियों को विभिन्न एआई और एमएल उपकरणों और तकनीकों के बारे में अभ्यास सत्रों से परिचित कराना है, जिससे वे पाइथन में अपने स्वयं के अनुप्रयोगों को विकसित कर सकें।

हमने एआई अनुप्रयोगों के विकास पर सिद्धांत और व्यावहारिक दोनों कक्षाओं की पेशकश करने के लिए इस पाठ्यक्रम को डिजाइन किया है। इस पाठ्यक्रम के अंतर्गत आने वाले विषयों में एआई का परिचय और कृषि में इसका दायरा, पाइथन का बुनियादी सिंटैक्स, पाइथन में व्यावहारिक सत्र, प्रोग्रामिंग की वस्तु-उन्मुख अवधारणा, डेटा हैंडलिंग और विजुअलाइज़ेशन तकनीक, एम.एल. की तकनीक और उपयुक्त पैकेज, डिजिटल इमेज, प्रसंस्करण, वेब विकास ढांचा आदि शामिल हैं।

हम इस अवसर पर संस्थान के शिक्षकों को धन्यवाद देना चाहते हैं जिन्होंने इस पाठ्यक्रम को सफल बनाने में अपना बहुमूल्य समय दिया है। उनके सहयोग के बिना इस नियमावली को समय पर पूरा करना संभव नहीं होता। हम इस प्रशिक्षण कार्यक्रम में अपने कर्मचारियों को प्रतिनियुक्त करने के लिए विभिन्न भा.कृ.अनु.प. संस्थानों और राज्य कृषि विश्वविद्यालयों के भी आभारी हैं। हम डॉ. राजेंद्र प्रसाद, निदेशक, भा.कृ.अनु.प.-भा.कृ.सां.असं के उनके बहुमूल्य मार्गदर्शन और पाठ्यक्रम के सुचारू संचालन के लिए सभी आवश्यक सुविधाएं उपलब्ध कराने के लिए आभारी हैं। हम उन सभी के आभारी हैं जिन्होंने इस प्रशिक्षण नियमावली को तैयार करने में प्रत्यक्ष व अप्रत्यक्ष रूप से हमारी सहायता की है।


02.02.23

(डॉ. संचिता नाहा)
पाठ्यक्रम संयोजक


अशरफुल हक

(डॉ. मो. अशरफुल हक)
पाठ्यक्रम संयोजक



(डॉ. सुदीप)
पाठ्यक्रम निदेशक


PREFACE

ICAR- Indian Agricultural Statistics Research Institute (ICAR-IASRI) is a premier Institute of relevance in Statistical Sciences (Statistics, Computer Applications and Bioinformatics) and their judicious fusion in agricultural sciences for enriching quality of agricultural research and informed policy decision making. The Institute has taken a lead in developing various Software, Web Applications, Mobile apps and now Artificial Intelligence (AI) based tools, techniques, and methodologies useful for Agricultural Research.

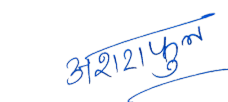
Since the recent rise of AI, data scientists and other domain experts are clearly in need of AI training and sensitization. One of the institute's main research interests is the development of AI-based tools and methodologies for agriculture. Moreover, software like R, Python etc. are playing major roles in analysing data emerged from various agricultural experiments. ICAR-IASRI is organizing a Short Course on "Python for Artificial Intelligence in Agriculture" for all the scientific staff of ICAR or SAUs/CAUs/ICAR funded KVKs during February 02-11, 2023 (10 days). The aim of this short course is to familiarize participants with Python programming used in developing Artificial Intelligence (AI) and Machine Learning (ML) applications. Our objective is to acquaint the participants about various AI and ML tools and techniques with hands on practice sessions enabling them to develop their own applications in python.

We have designed this course to offer both theory and practical classes on developing AI applications. The topics covered under this course include introduction to AI and its scope in agriculture, basic syntax of python, hands on practical sessions in python, object-oriented concepts of programming, data handling and visualization techniques, machine learning techniques and suitable packages, digital image processing, Web development framework etc.


We would like to take this opportunity to thank the faculty of the institute who have spared their valuable time in making this course successful. Without their cooperation timely completion of this manual would not have been possible. We are also thankful to the various ICAR Institutes and State Agricultural Universities for deputing their employees in this training programme. We are grateful to Dr. Rajender Parsad, Director, ICAR-IASRI for his valuable guidance and making all necessary facilities available for smooth conduct of the course. We are thankful to each and every one who has supported us directly or indirectly for preparing this training manual.


(Dr Sanchita Naha)

Course Coordinator


(Dr Md Ashraf Haque)

Course Coordinator


(Dr Sudeep)

Course Director

**DIVISION OF COMPUTER APPLICATIONS
ICAR - INDIAN AGRICULTURAL STATISTICS RESEARCH INSTITUTE
LIBRARY AVENUE, PUSA, NEW DELHI - 110012**

Short Course on

“Python for Artificial Intelligence in Agriculture”

02 – 11 February 2023

COURSE SCHEDULE

Date & Day	09:15-09:30	09:30 – 11:15	11:15 – 12:30	12:30-13:30	14:30 – 15:45	16:00 – 17:30
02.02.2023 (Thursday)	Registration	Introduction to Artificial Intelligence (Sudeep Marwaha)	Inauguration	Introductory Python (Madhu)	Introduction to Python, Basic Syntax, variables and Operators (Madhu)	Practice session I
Date & Day	Time					
	09:15-11:00		11:15 – 13:00		14:00 – 15:45	
03.02.2023 (Friday)	Data Structures, Control Structures and Loops (Md. Ashraf Haque)		Practice session II		Functions, Module, File Handling, (Akshay Dheeraj)	
04.02.2023 (Saturday) HOLIDAY						
05.02.2023 (Sunday) HOLIDAY						
Date & Day	Time					
	09:15-11:00		11:15 – 13:00		14:00 – 15:45	
06.02.2023 (Monday)	OOps concepts and Exception handling (Soumen Pal)		Practice session IV (Madhu)		Data Handling and Visualization using NumPy, Pandas, Matplotlib (Sanchita Naha)	
07.02.2023 (Tuesday)	Introduction to Machine Learning and Scikit-Learn (Ramasubramanian V)		Regression Analysis using Scikit-Learn (Kanchan Sinha)		Support Vector Machines using Scikit-Learn (Mrinmoy Ray)	
08.02.2023 (Wednesday)	Concepts of Artificial Neural Network: Perceptron, RNN, LSTM (Anshu Bharadwaj)		Practical session of ANN: Tensorflow and Keras (Chandan Kumar Deb)		Deep Learning and Convolutional Neural Networks (Md. Ashraf Haque)	
09.02.2023 (Thursday)	Introduction to Digital Image Processing (Alka Arora)		Digital Image Processing using OpenCV (Practical) (Chandan Kumar Deb)		Ensemble Techniques (Shashi Dahiya)	
10.02.2023 (Friday)	Development of APIs using Python (Sanchita Naha)		AI-DISC Demonstration (S N Islam)		Feedback & Evaluation	
11.02.2023 (Saturday)	Doubt Clearing Session				Practical Session VI	

Tea Break:

11:00 – 11:15 & 15:45 – 16:00

Lunch:

13:00 – 14:00

Note: All instructors are supposed to be present in the practical, Review & problem-solving sessions. Technical staff will assist the instructors in the theory as well as practical sessions.

CONTENTS

Sl. No.	Topic	Author	Page No.
1.	Introduction to Artificial Intelligence	Sudeep	1-10
2.	Introduction to Python, Basic Syntax, Variables and Operators	Madhu	11-26
3.	Data Structures, Control Structures and Loops	Md. Ashraful Haque	27-34
4.	Functions, Module, File Handling	Akshay Dheeraj	35-58
5.	Object-Oriented Programming	Madhu	59-88
6.	Data Handling and Visualization using NumPy, Pandas and Matplotlib	Sanchita Naha	89-108
7.	Introduction to Machine Learning and Scikit-Learn	Ramasubramanian V	109-118
8.	Regression Analysis using Scikit-Learn	Kanchan Sinha	119-130
9.	Support Vector Machine using Scikit-learn Library	Mrinmoy Ray	131-138
10.	Random Forest using Scikit-learn Library	Upendra Kumar Pradhan	139-144
11.	Concepts of Artificial Neural Network (ANN): Perceptron, RNN, LSTM	Anshu Bharadwaj	145-156
12.	Hands on Artificial Neural Network using Scikit-learn and Keras	Chandan Kumar Deb	157-162
13.	Deep Learning and Convolutional Neural Networks	Md. Ashraful Haque and Akshay Dheeraj	163-174
14.	Image Classification using CNN: Tensorflow and Keras	Akshay Dheeraj and Md. Ashraful Haque	175-180
15.	Introduction to Digital Image Processing using OpenCV	Alka Arora and Chandan Kumar Deb	181-188
16.	Ensemble Techniques	Shashi Dahiya	189-198
17.	Introduction to Big Data Analytics in Agriculture	Samarth Godara	199-204
18.	Web Development and API binding using Flask Framework	Sanchita Naha	205-212

Introduction to Artificial Intelligence

Sudeep

ICAR-Indian Agricultural Statistics Research Institute, New Delhi - 110 012

sudeep@icar.gov.in

1. Introduction

The Artificial Intelligence is a very old field of study and has a rich history. Modern AI was formalized by John McCarthy, considered as father of AI. It is a branch of computer science, founded around early 1950's. Primarily, the term Artificial Intelligence (or AI) refers to a group of technique that enables a computer or a machine to mimic the behaviour of humans in problem solving tasks. Formally, AI is described as “the study of how to make the computers do things at which, at the moment, people are better” (Rich and Knight, 1991; Rich et al., 2009). The main aim of AI is to program the computer for performing certain tasks in humanly manner such as knowledgebase, reasoning, learning, planning, problem solving etc. The Machine Learning (ML) techniques are the subset of AI which makes the computers/machines/programs the capable of learning and performing tasks without being explicitly programmed. The ML techniques are not just the way of mimicking human behaviour but the way of mimicking how humans learn things. The main characteristics of machine learning is ‘learning from experience’ for solving any kind of problem. The methods of learning can be categorized into three types: (a) supervised learning algorithm is given with labelled data and the desired output whereas (b) unsupervised learning algorithm is given with unlabelled data and identifies the patterns from the input data and (c) reinforcement learning algorithm allows the ML techniques to capture the learnable things on the basis of rewards or reinforcement. Now, the Deep Learning (DL) technique are the advanced version of machine learning algorithms gained much popularity in the area of image recognition and computer vision. The artificial neural networks (ANNs) clubbed with representation learning are the backbone of the deep learning concepts. These techniques allows a machine to learn patterns in the dataset with multiple levels of abstractions. The DL models are composed of a series of non-linear layers where each of the layer has the capability of transforming the low-level representations into higher-level representations i.e. into a more abstract representations (LeCun *et al.*, 2015). There are several DL algorithms available now-a-days such as Deep Convolutional Neural Networks, Deep Recurrent Neural networks, Long Short-term Memory (LSTM) networks that are being applied to different areas of engineering, bioinformatics, agriculture, medical science and many more (Fusco et al., 2021).

2. Applications of Artificial Intelligence in Agriculture:

In present scenario, AI techniques are being exponentially applied in the various areas of the agricultural domain. These areas can be categorized into the following groups:

Soil and water management, Crop Health Management, Crop Phenotyping, Recommender-based systems for crops, Semantic web and Ontology driven expert systems for crops and Geo-AI. The application of AI, ML and DL based techniques on these areas are discussed in the following sections.

2.1 Soil and Irrigation Management:

Soil and irrigation are the most viable components of agriculture. The soil and irrigation are the determinant factors for the optimum crop yield. In order to obtain enhanced crop yield and to maintain the soil properties, there is a requirement of appropriate knowledge about the soil resources. The management of irrigation becomes crucial when there are scares of water availability. Therefore, the soil and irrigation related issues should be managed properly and cautiously to ensure a potential yield in crops. In this regards, AI and ML based techniques have shown potential ability to resolve soil and irrigation related issues in crops. A range of machine learning models such as linear regression, support vector machines (or regressors), Artificial neural networks, random forest algorithm and so on are being used. Many researchers have used remote-sensed data with the machine learning techniques for determining soil health parameters. In this section, few significant works in this field are highlighted below:

A. Soil Management:

Besalatpour et al., (2011), Aitkenhead et al., (2012) and Sirsat et al., (2017) used different machine learning techniques such as linear regression, support vector machine, random forests for the prediction of the physical and chemical properties of soil. Rivera et al. (2020) and Azizi et al., (2020) worked on estimation and classification of aggregate stability of the soils using conventional machine learning techniques as well as deep learning models. Jha et al., (2018) worked on prediction of microbial dynamics in soils using regression-based techniques. Patil and Dekha (2016) and Mehdizadeh et al (2017) worked on predicting the evapotranspiration rate in crops using several machine learning techniques. Researchers worked on mapping the soil properties digitally using the remote sensing data with the help of machine learning and deep learning models (Taghizadeh-Mehrjardi et al. 2016; Kalambukattu et al., (2018; Padarian et al. 2019; Taghizadeh-Mehrjardi et al., 2020).

B. Irrigation management:

Zema et al. 2018 applied Data Envelopment Analysis (DEA) with Multiple Regression analysis to improve the irrigation performance Water Users Associations. Ramya et al. 2020 and Glória et al., 2021 worked on IoT based smart irrigation systems with machine learning models. Agastya et al, 2021 and Zhang et al. 2018 used deep learning-based CNN models for detection of irrigations using remote sensing data. Jimenez et al. 2021 worked on estimating the irrigation based on soil matric potential.

2.2 Crop Health Management:

Every year a significant amount of yield is damaged due to attack of disease causing pathogens and insect-pest infestation. In order to manage the spread of the diseases

and insect-pests, proper management practices should be applied at the earliest. Therefore, there is requirement of automatic diseases, pest identification system. In this regard, image-based diagnosis of diseases and pests have become de facto standard of automatic stress identification. This kind of automated detection methodology use sophisticated deep learning-based AI techniques that reduces the intervention of the human experts. There are several attempts have been done to diagnose the diseases as well as insects-pests in crops using deep learning techniques. In this section, some of the significant works in this field have been discussed briefly.

A. Disease identification:

Mohanty et al. 2016 worked on disease diagnosis problem using deep CNN models. They used an open-source dataset named PlantVillage (Hughes and Salathe, 2016) containing 54,306 digital images of 26 diseases from 14 crops. Ferentinos, 2018 worked on developing deep CNN-based models for recognising 56 diseases from different crops. Barbedo, 2019 applied transfer learning approach for diagnosis of diseases of 12 different crops. Too et al. 2019, applied pre-trained deep CNN models for identification of diseases of 18 crops using the PlantVillage data. Chen et al. 2020 applied a pretrained VGGNet network for classifying the diseases of Rice and Maize crop. Chen et. al. 2020 and Rahman et al. 2020 worked on identifying the major diseases of Rice crop. Lu et al., 2017; Johannes et al. 2017; Picon et al. 2019 and Nigam et al. 2021 applied deep CNN models for recognising the diseases of wheat crop. Priyadharshini et al. 2019; Sibiya & Sumbwanyambe, 2019; Haque et al. 2021 used deep learning models for identifying diseases of maize crop.

B. Pest Identification:

Pest Identification problem is inherently different from disease detection. As compared to disease detection there are less number of work has found in the literature. Some of the research of pest identification has been discussed in the following section.

Cheeti et al. (2021) developed a model for pest detection and classification of peat using YOLO(You look only once) and CNN. YOLO algorithm is used for detection of pest in an image and Alex net CNN is used for pest classification. Chen et al. (2021) propose an AI-based pest detection system for solving the specific issue of detection of scale pests based on pictures. Deep-learning-based object detection models, such as faster region-based convolutional networks (Faster R-CNNs), single-shot multibox detectors (SSDs), and You Only Look Once v4 (YOLO v4), are employed to detect and localize scale pests in the picture. Taiwan Agricultural Research Institute, Council of Agriculture, has collected images of the three types of pests from the actual fields for decades. Fuentes et al. (2017) address disease and pest identification by introducing the application of deep meta-architectures and feature extractors. They proposed a robust deep-learning-based detector for real-time tomato diseases and pests recognition. The system introduces a practical and applicable solution for detecting the class and location of diseases in tomato plants, which in fact represents

a main comparable difference with traditional methods for plant diseases classification. Karnik et al. (2021) image pre-processing and data augmentation techniques has been performed to get better image.yolov3 classification for classifying plant leaf disease of pepper bell, potato and tomato. This proposed in divided into two stage part first classifier and second stage classifier where in first classifier it will preprocess of median filter and data augmentation is used and trained in yolov3 algorithm and in second stage classifier it will perform the extract plant leaf image output using Resnet50 based. So, it two step classification approach. Based on this research work we achieved 94% accuracy of detection lead diseases. Experiments showed [Li et al. (2020)] that our system with the custom backbone was more suitable for detection of the untrained rice videos than VGG16, ResNet-50, ResNet-101 backbone system and YOLOv3 with our experimental environment. Liu et al.2020 used Yolo V3 model is a little inadequate in the scale when recognizing tomato disease spots and pests.

2.3 Plant Phenomics:

Non-destructive phenotypic measurement with high throughput imaging technique becoming extremely popular. High throughput imaging system produces a large number of images. Deduction of the phenotypic characteristics through image analysis is quick and accurate. A wide range of phenotypic study can be done using phenomics analysis. High throughput imaging system coupled with sophisticated AI technology like deep learning make this field more efficient and accurate. Phenomics is has been used for study of several phenotypic characters like spike detection and counting, yield forecasting, quantification of the senescence in the plant, leaf weight and count, plant volume, convex hull, water stress and many more.

2.4 Recommender Systems:

Recommender systems (RSs) help online users in decision making regarding products among a pile of alternatives. In general, these systems are software solutions which predict liking of a user for unseen items. RSs have been mainly designed to help users in decision making for areas where one is lacking enough personal experience to evaluate the overwhelming number of alternative items that a website has to offer [Resnick & Varian, 1997]. Recommender systems have proved its worth in many different applications like e-commerce, e-library, e-tourism, e-learning, e-business, e-resource services etc. by suggesting suitable products to users [Lu *et al.*, 2015]. RSs are used to introduce new/unseen items to users, to increase user satisfaction etc. Recommendations are generated by processing large amount of historical data on the users and the products to be suggested. Most popular way of gathering users liking on a particular product is in terms of rating either in numerical scale (1 to 5) or ordinal scale (strongly agree, agree, neutral, disagree, strongly disagree). Other techniques of more knowledge-based recommendation are the use of Ontologies [Middleton *et al.*, 2002] of user profiles or item descriptions etc. The core task of a recommendation system is to predict the usefulness of an item to an individual user based on the earlier history of that item or by evaluating the earlier choices of the user. Collaborative way

of user modelling [Konstan *et al.*, 1997] is where ratings are predicted for $\langle user, item \rangle$ pair, $\bar{R}_{\langle u, i \rangle}$ based on a large number of ratings previously gathered by the system on individual $\langle user, item \rangle$ pairs. Another way of recommendation is to suggest items that are similar to the ones previously liked by the user, called Content based filtering [Wang *et al.*, 2018; Smyth, 2007]. In a hybrid method of prediction, limitations by the earlier mentioned processes are tackled in various ways.

Agriculture has used recommender systems since 2015 and continues to do so. RSs have been explored to develop crop recommendation strategies based on soil and weather parameters, crop rotation practices, water management, suggestion on suitable varieties, recommendations for management practices etc. It is absolutely essential for the farmers to receive recommendations on the best crop for cultivation. Kamatchi and parvati, 2019 proposed a hybrid RS in combination with Collaborative Filtering, Case-based Reasoning and Artificial Neural Networks (ANN) to predict future climatic conditions and recommendation of crops based on the predicted climate. Crop recommendations have been developed based on season and productivity [Vaishnavi *et al.*, 2021], area and soil type [Pande *et al.*, 2021] by using several machine learning algorithms like Support vector Machine (SVM), Random forest (RF), Multivariate Linear regression (MLR), K- Nearest neighbour (KNN), ANN etc. Ensemble techniques have been used to develop a collaborative system of crop rotation, crop yield prediction, forecasting and fertilizer recommendation [Archana *et al.*, 2020]; to classify soil types into recommended crop types Kharif or Rabi based on specific physical and chemical characteristics, average rainfall and surface temperature [Kulkarni *et al.*, 2018]. Naha and Marwaha, 2020 presented an Ontology driven context aware RS that can recommend land preparation methods, sowing time, seed rate, fertilizer management, irrigation scheduling and harvesting methods to Maize cultivators. Application of RSs has also penetrated in the e-agriculture domain by suggesting parts of agricultural machineries in online ordering [Ballesteros *et al.*, 2021].

2.5 Semantic web, Knowledgebase and Natural Language processing:

Agriculture is vast source of resources and so it is also a vast source of information. The problem with this information is most of the information are unstructured. That unstructured knowledge is merely understandable for machine. It is also has low accessibility for human too. The main objectives of the semantic web and knowledge base system is to make unstructured data into structured one. Semantic web and the knowledgebase mainly facilitated by the ontology in the back end. Ontology is a formal, explicit specification of a shared conceptualization (Gruber, 1993). Making of Ontology that facilitated the semantic web and knowledge base can be made across the agricultural domain to make the unstructured data into structured one. Many ontology has already been developed in accordance with the Bedi and Marwaha, 2004 in the agricultural domain. Saha *et al.*, (2011) developed an ontology on dynamic maize variety selection in different climatic condition, Sahiram *et.al.*, (2012) developed a ontology on rapeseed and mustard for identification of the variety in

multiple languages, Das *et. al.*, (2011) developed a ontology for USDA soil taxonomy and ontology was extended by Deb *et. al.*, (2012), Biswas *et. al.*, (2012) developed a ontology on microbial taxonomy and was extended by Karn *et. al.*, (2014).

2.6 GIS and Remote sensing coupled with AI:

GIS and Remote sensing is helping agricultural community since long. The land use planning, land cover analysis, forest distribution, water distribution, water use pattern, crop rotation and crop calendar analysis can be done by GIS and remote sensing. But when the AI and machine learning coupled with these technology it become more powerful. Machine learning and AI efficiently used for correct land classification and phenological change detection. From Digital soil mapping to yield forecasting, from phenology detection to leaf area index a vast range of the area in agriculture can be handled by GIS and Remote sensing.

References:

- Agastya, C., Ghebremusse, S., Anderson, I., Vahabi, H., & Todeschini, A. (2021). Self-supervised Contrastive Learning for Irrigation Detection in Satellite Imagery. arXiv preprint arXiv:2108.05484.
- Ahila Priyadharshini, R., Arivazhagan, S., Arun, M., & Mirnalini, A. (2019). Maize leaf disease classification using deep convolutional neural networks. *Neural Computing and Applications*, 31(12), 8887-8895.
- Aitkenhead, M. J., Coull, M. C., Towers, W., Hudson, G., & Black, H. I. J. (2012). Predicting soil chemical composition and other soil parameters from field observations using a neural network. *Computers and Electronics in Agriculture*, 82, 108-116.
- Archana, K., & Saranya, K. G. (2020). Crop Yield Prediction, Forecasting and Fertilizer Recommendation using Voting Based Ensemble Classifier. *Seventh Sense Research Group (SSRG) International Journal of Computer Science and Engineering*, 7(5), 1-4.
- Azizi, A., Gilandeh, Y. A., Mesri-Gundoshmian, T., Saleh-Bigdeli, A. A., & Moghaddam, H. A. (2020). Classification of soil aggregates: A novel approach based on deep learning. *Soil and Tillage Research*, 199, 104586.
- Ballesteros J. M., Cartujano, A. R., Evaldez, D., Macutay, J., (2021). Online ordering and recommender system of combine harvester parts and equipment with 3D modelling and augmented reality brochure for BLAZE equifarm and general merchandise. *11th International Workshop on Computer Science and Engineering (WCSE 2021)*, 174-179.
- Barbedo, J. G. A. (2019). Plant disease identification from individual lesions and spots using deep learning. *Biosystems Engineering*, 180, 96-107.

- Besalatpour, A., Hajabbasi, M. A., Ayoubi, S., Gharipour, A., & Jazi, A. Y. (2012). Prediction of soil physical properties by optimized support vector machines. *International Agrophysics*, 26(2).
- Biswas, S., Marwaha, S., Malhotra, P. K., Wahi, S. D., Dhar, D. W., & Singh, R. (2013). Building and querying microbial ontology. *Procedia Technology*, 10, 13-19.
- Cheeti, S., Kumar, G. S., Priyanka, J. S., Firdous, G., & Ranjeeva, P. R. (2021). Pest Detection and Classification Using YOLO AND CNN. *Annals of the Romanian Society for Cell Biology*, 15295-15300.
- Chen, J. W., Lin, W. J., Cheng, H. J., Hung, C. L., Lin, C. Y., & Chen, S. P. (2021). A smartphone-based application for scale pest detection using multiple-object detection methods. *Electronics*, 10(4), 372.
- Chen, J., Zhang, D., Nanehkaran, Y. A., & Li, D. (2020). Detection of rice plant diseases based on deep transfer learning. *Journal of the Science of Food and Agriculture*, 100(7), 3246-3256.
- Das, B. *et al.* (2017a) “Comparison of different uni-and multi-variate techniques for monitoring leaf water status as an indicator of water-deficit stress in wheat through spectroscopy,” *Biosystems Engineering*, 160, pp. 69–83.
- Deb, C. K., Marwaha, S., Malhotra, P. K., Wahi, S. D., & Pandey, R. N. (2015, March). Strengthening soil taxonomy ontology software for description and classification of USDA soil taxonomy up to soil series. In *2015 2nd International Conference on Computing for Sustainable Global Development (INDIACom)* (pp. 1180-1184). IEEE.
- Ferentinos, K. P. (2018). Deep learning models for plant disease detection and diagnosis. *Computers and electronics in agriculture*, 145, 311-318.
- Fuentes, A., Yoon, S., Kim, S. C., & Park, D. S. (2017). A robust deep-learning-based detector for real-time tomato plant diseases and pests recognition. *Sensors*, 17(9), 2022.
- Fusco, R., Grassi, R., Granata, V., Setola, S. V., Grassi, F., Cozzi, D., ... & Petrillo, A. (2021). Artificial Intelligence and COVID-19 Using Chest CT Scan and Chest X-ray Images: Machine Learning and Deep Learning Approaches for Diagnosis and Treatment. *Journal of Personalized Medicine*, 11(10), 993.
- Glória, A.; Cardoso, J.; Sebastião, P. Sustainable irrigation system for farming supported by machine learning and real-time sensor data. *Sensors* 2021, 21, 3079.
- Janik, L. J., Forrester, S. T., & Rawson, A. (2009). The prediction of soil chemical and physical properties from mid-infrared spectroscopy and combined partial

- least-squares regression and neural networks (PLS-NN) analysis. *Chemometrics and Intelligent Laboratory Systems*, 97(2), 179-188.
- Jimenez, A.F.; Ortiz, B.V.; Bondesan, L.; Morata, G.; Damianidis, D. Long short-term memory neural network for irrigation management: A case study from southern Alabama, USA. *Precis. Agric.* 2021, 22, 475–492
- Johannes, A., Picon, A., Alvarez-Gila, A., Echazarra, J., Rodriguez-Vaamonde, S., Navajas, A. D., & Ortiz-Barredo, A. (2017). Automatic plant disease diagnosis using mobile capture devices, applied on a wheat use case. *Computers and electronics in agriculture*, 138, 200-209.
- Kalambukattu, J. G., Kumar, S., & Raj, R. A. (2018). Digital soil mapping in a Himalayan watershed using remote sensing and terrain parameters employing artificial neural network model. *Environmental earth sciences*, 77(5), 1-14.
- Kamatchi, S., B. & Parvathi, R. (2019). Improvement of crop production using recommender system by weather forecasts. *Procedia Computer Science*, 165, 724–732.
- Karnik, J., & Suthar, A. (2021). Agricultural Plant Leaf Disease Detection Using Deep Learning Techniques. Available at SSRN 3917556.
- Konstan, J. A., Miller, B. N., Maltz, D., Herlocker, J. L., Gordon, L. R., & Riedl, J. (1997). Grouplens: Applying collaborative filtering to usenet news. *Communications of the ACM*, 40(3), 77-87.
- LeCun, Y., Bengio, Y. & Hinton, G. (2015). Deep learning. *Nature*, **521(7553)**, 436-444.
- Li, D., Wang, R., Xie, C., Liu, L., Zhang, J., Li, R., ... & Liu, W. (2020). A recognition method for rice plant diseases and pests video detection based on deep convolutional neural network. *Sensors*, 20(3), 578.
- Liu, J., & Wang, X. (2020). Tomato diseases and pests detection based on improved Yolo V3 convolutional neural network. *Frontiers in plant science*, 11, 898.
- Lu, J., Hu, J., Zhao, G., Mei, F., & Zhang, C. (2017). An in-field automatic wheat disease diagnosis system. *Computers and electronics in agriculture*, 142, 369-379.
- Lu, J., Wu, D., Mao, M., Wang, W., and Zhang, G., (2015). Recommender system application developments: a survey. *Decision Support Systems*, 74, 12- 32.
- Manoranjan, D., Malhotra, P. K., Sudeep, M., & Pandey, R. N. (2012). Building and querying soil ontology. *Journal of the Indian society of agricultural statistics*, 66(3), 459-464.
- Mohanty, S. P., Hughes, D. P., & Salathé, M. (2016). Using deep learning for image-based plant disease detection. *Frontiers in plant science*, 7, 1419.

- Naha, S. and Marwaha, S. (2020). Context-Aware Recommender System for Maize Cultivation. *Journal of Community Mobilization and Sustainable Development*, 15(2), 485-490.
- Nigam, S., Jain, R., Marwaha, S., & Arora, A. (2021). Wheat rust disease identification using deep learning. De Gruyter.
- Padarian, J., Minasny, B., & McBratney, A. B. (2019). Using deep learning for digital soil mapping. *Soil*, 5(1), 79-89.
- Pande, S. M., Ramesh, P. K., Anmol, A., Aishwarya, B. R., Rohilla, K., & Shaurya, K. (2021). Crop recommender system using machine learning approach. *In 2021 5th International Conference on Computing Methodologies and Communication (ICCMC)*, 1066-1071. IEEE.
- Patil, A. P., & Deka, P. C. (2016). An extreme learning machine approach for modelling evapotranspiration using extrinsic inputs. *Computers and electronics in agriculture*, 121, 385-392.
- Resnick, P., & Varian, H. R. (1997). Recommender systems. *Communications of the ACM*, 40(3), 56- 58.
- Rich, E., & Knight, K. (1991). *Artificial Intelligence. 2nd Edn.* New York, NY, United States: McGraw-Hill.
- Rich, E., Knight, K., and Nair, S. B. (2009). *Artificial Intelligence. 3rd Edn.* New Delhi, India: Tata McGraw-Hill.
- Rivera, J. I., & Bonilla, C. A. (2020). Predicting soil aggregate stability using readily available soil properties and machine learning techniques. *Catena*, 187, 104408.
- Sibiya, M., & Sumbwanyambe, M. (2019). A computational procedure for the recognition and classification of maize leaf diseases out of healthy leaves using convolutional neural networks. *AgriEngineering*, 1(1), 119-131.
- Sirsat, M. S., Cernadas, E., Fernández-Delgado, M., & Khan, R. (2017). Classification of agricultural soil parameters in India. *Computers and electronics in agriculture*, 135, 269-279.
- Smyth, B. (2007). Case-based recommendation. In *The adaptive web*. Springer, Berlin, Heidelberg, 342-376.
- Taghizadeh-Mehrjardi, R., Ayoubi, S., Namazi, Z., Malone, B. P., Zolfaghari, A. A., & Sadrabadi, F. R. (2016). Prediction of soil surface salinity in arid region of central Iran using auxiliary variables and genetic programming. *Arid Land Research and Management*, 30(1), 49-64.

- Too, E. C., Yujian, L., Njuki, S., & Yingchun, L. (2019). A comparative study of fine-tuning deep learning models for plant disease identification. *Computers and Electronics in Agriculture*, 161, 272-279.
- Vaishnavi, S., Shobana, M., Sabitha, R., & Karthik, S. (2021). Agricultural Crop Recommendations based on Productivity and Season. *In 2021 7th International Conference on Advanced Computing and Communication Systems (ICACCS)*. 1, 883-886. IEEE.
- Wang, D., Liang, Y., Xu, D., Feng, X., & Guan, R. (2018). A content-based recommender system for computer science publications. *Knowledge-Based Systems*, 157, 1-9.
- Zema, D.A.; Nicotra, A.; Mateos, L.; Zimbone, S.M. Improvement of the irrigation performance in water users associations integrating data envelopment analysis and multi-regression models. *Agric. Water Manag.* 2018, 205, 38–49.

Introduction to Python: Basic Syntax, Variables and Operators

Madhu

ICAR-Indian Agricultural Statistics Research Institut, New Delhi - 110 012

madhu.dahiya@icar.gov.in

Introduction to Python:

Python is a very popular general-purpose interpreted, interactive, object-oriented, and high-level programming language. Python is dynamically-typed and garbage-collected programming language. It was created by Guido van Rossum during 1985-1990. Like Perl, Python source code is also available under the GNU General Public License (GPL).

Characteristics of Python: Following are important characteristics of Python Programming-

- It supports functional and structured programming methods as well as OOP.
- It can be used as a scripting language or can be compiled to byte-code for building large applications.
- It provides very high-level dynamic data types and supports dynamic type checking.
- It supports automatic garbage collection.

It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java.

Basic Python Syntax:

The basic syntax of the python programming language is as follows:

```
print('India is my country.')
```

Variables in Python:

```
x=2
y="India"
print(x)
print(y)
```

Type Casting

```
x=str(5)
y=int(5.0)
z=float(5)
print(x)
print(y)
print(z)
print(type(x))
```



```
print(type(y))
```

Single & Double Quotes

```
x="india"  
y='india'  
print(x)  
print(y)
```

Multiline Strings

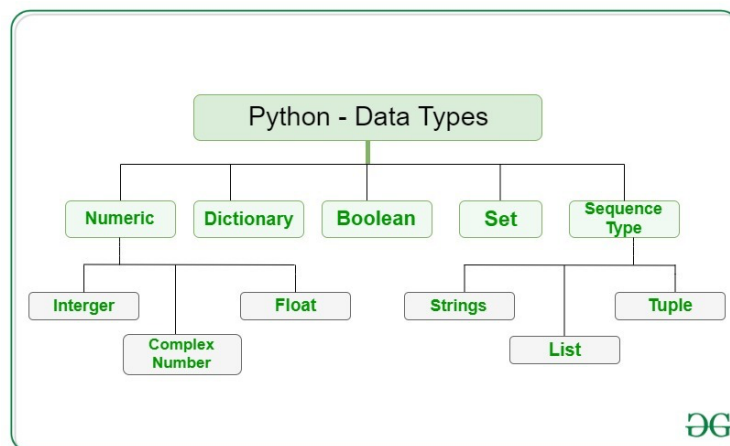
```
a = """hello,  
good morning,  
h r u,  
all."""  
print(a)
```

```
a = '''hello,  
good morning,  
h r u  
all.'''  
print(a)
```

Python Data Types:

Data types are the classification or categorization of data items. It represents the kind of value that tells what operations can be performed on a particular data. Since everything is an object in Python programming, data types are actually classes and variables are instance (object) of these classes. Following are the standard or built-in data type of Python:

- Numeric
- Sequence Type
- Boolean
- Set
- Dictionary



Numeric: In Python, numeric data type represents the data which has numeric value. Numeric value can be integer, floating number or even complex numbers. These values are defined as int, float and complex class in Python.

- **Integers** – This value is represented by int class. It contains positive or negative whole numbers (without fraction or decimal). In Python there is no limit to how long an integer value can be.
- **Float** – This value is represented by float class. It is a real number with floating point representation. It is specified by a decimal point. Optionally, the character e or E followed by a positive or negative integer may be appended to specify scientific notation.
- **Complex Numbers** – Complex number is represented by complex class. It is specified as *(real part) + (imaginary part)j*. For example $-2+3j$

Note – type() function is used to determine the type of data type.

Float

```
x = 1.10
y = 1.0
z = -35.59
print(type(x))
print(type(y))
print(type(z))
```

int

```
x = 1
y = 3565
z = -3255522
print(type(x))
print(type(y))
print(type(z))
```

complex

```
x = 3+5j
y = 5j
z = -5j
print(type(x))
print(type(y))
print(type(z))
```

Sequence Type: In Python, sequence is the ordered collection of similar or different data types. Sequences allows to store multiple values in an organized and efficient fashion. There are several sequence types in Python –

- **String**
- **List**
- **Tuple**

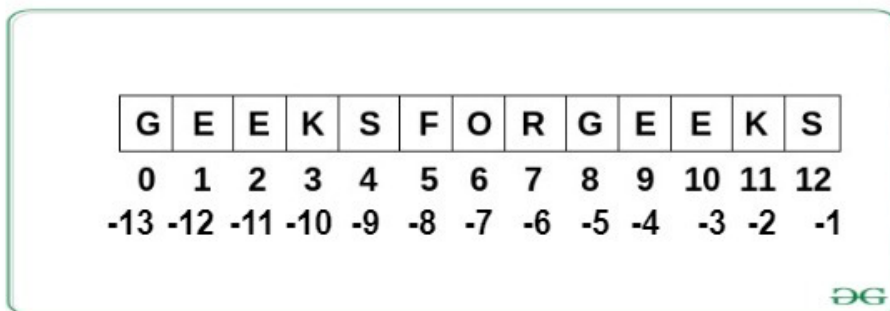
1) **String:** In Python, Strings are arrays of bytes representing Unicode characters. A string is a collection of one or more characters put in a single quote, double-quote or triple quote. In python there is no character data type, a character is a string of length one. It is represented by str class.

Creating String

Strings in Python can be created using single quotes or double quotes or even triple quotes.

Accessing elements of String

In Python, individual characters of a String can be accessed by using the method of Indexing. Indexing allows negative address references to access characters from the back of the String, e.g. -1 refers to the last character, -2 refers to the second last character and so on.



```
print("Hello")
print('Hello')
a = "Hello"
print(a)
```

Strings are Arrays

```
a = "Hello, World!" # 0...n
print(a[1])
print(a[2])
print(a[7])
```

Looping Through a String

```
for x in "banana":
    print(x)
```

String Slicing

```
b = "Hello, World!"
print(b[2:5])
b = "Hello, World!"
print(b[:5])
b = "Hello, World!"
print(b[2:])
```

```
b = "Hello, World!"
print(b[-5:-2])
print(b[-1])
```

Strings Functions

```
a = "hello, World!"
print(a.upper())           #Converts a string into upper case
print(a.capitalize())     #Converts the first character to upper
case
print(a.casefold())       #Converts string into lower case
print(a.split())          #Splits the string at the specified
separator, and returns a list
print(a.lower())          #Converts a string into lower case
print(a.strip())          # Returns a trimmed version of the
string
#returns "Hello, World!"
print(a.replace("h", "J")) #Returns a string where a specified
value is replaced with a specified value
print(a.isdigit())        #Returns True if all characters in the
string are digits
print(a.isupper())        #Returns True if all characters in the
string are upper case
print(len(a))             #len() function returns the length of a string
```

String Concatenation

```
a = "Hello"
b = "World"
c = a + b
print(c)
a = "Hello"
b = "World"
c = a + " " + b
print(c)
"""we cannot combine strings and numbers"""
age = 36
txt = "My name is John, I am " + age
print(txt)
```

we can combine strings and numbers by using the **format() method!** The format() method takes the passed arguments, formats them, and places them in the string where the placeholders {}

```
age = 36
txt = "My name is John, and I am {}"
print(txt.format(age))
```

The format() method takes unlimited number of arguments, and are placed into the respective placeholders:

You can use index numbers {0} to be sure the arguments are placed in the correct placeholders.

```
quantity = 3
itemno = 567
price = 49.95
myorder = "I want {} pieces of item {} for {} dollars."
print(myorder.format(quantity, itemno, price))
quantity = 3
itemno = 567
price = 49.95
myorder = "I want to pay {2} dollars for {0} pieces of item {1}."
print(myorder.format(quantity, itemno, price))
```

Boolean: Data type with one of the two built-in values, True or False. Boolean objects that are equal to True are truthy (true), and those equal to False are falsy (false). But non-Boolean objects can be evaluated in Boolean context as well and determined to be true or false. It is denoted by the class bool.

Note – True and False with capital ‘T’ and ‘F’ are valid boolean otherwise python will throw an error.

Booleans represent one of two values: True or False.

```
print(10 > 9)
print(10 == 9)
print(10 < 9)
print(bool("Hello"))           #Almost any value is evaluated to True if it
has some sort of content.
print(bool(15))                #Any string is True, except empty strings.
print(bool(0))                 # Any number is True, except 0.
print(bool(""))
print(bool(()))
print(bool([]))
print(bool({}))                # Any list, tuple, set, and dictionary are
True, except empty ones.
print(bool(False))
print(bool(None))
```

Python Arrays

Array in Python can be created by importing array module. array(data_type, value_list) is used to create an array with data type and value list specified in its arguments.

import array as arr

```
# creating an array with integer type
a = arr.array('i', [1, 2, 3])
# printing original array
print ("The new created array is : ", end = " ")
for i in range (0, 3):
    print (a[i], end = " ")
print()
```

Accessing Python Array Elements

```
import array as arr
a = arr.array('i', [2, 4, 6, 8])
print("First element:", a[0])
print("Second element:", a[1])
print("Last element:", a[-1])
```

Slicing Python Arrays

```
import array as arr
numbers_list = [2, 5, 62, 5, 42, 52, 48, 5]
numbers_array = arr.array('i', numbers_list)
print(numbers_array[2:5]) # 3rd to 5th
print(numbers_array[:-5]) # beginning to 4th
print(numbers_array[5:]) # 6th to end
print(numbers_array[:]) # beginning to end
```

Changing and Adding Elements

```
import array as arr
numbers = arr.array('i', [1, 2, 3, 5, 7, 10])
# changing first element
numbers[0] = 0
print(numbers) # Output: array('i', [0, 2, 3, 5, 7, 10])
# changing 3rd to 5th element
numbers[2:5] = arr.array('i', [4, 6, 8])
print(numbers) # Output: array('i', [0, 2, 4, 6, 8, 10])
import array as arr
numbers = arr.array('i', [1, 2, 3])
numbers.append(4)
print(numbers) # add one item to the array using the append() method
numbers.extend([5, 6, 7])
print(numbers) # add several items using the extend() method
import array as arr
odd = arr.array('i', [1, 3, 5])
even = arr.array('i', [2, 4, 6])
numbers = arr.array('i') # concatenate two arrays using + operator
numbers = odd + even
```

```
print(numbers)
import array as arr
number = arr.array('i', [1, 2, 3, 3, 4])
del number[2] # removing third element
print(number) # Output: array('i', [1, 2, 3, 4])
del number # deleting entire array
print(number) # Error: array is not defined
import array as arr
numbers = arr.array('i', [10, 11, 12, 12, 13])
numbers.remove(12) # remove() method to remove the given item
print(numbers)
print(numbers.pop(2)) # pop() method to remove an item at the given
index
print(numbers)
```

Python Operators

Operators are special symbols in Python that carry out arithmetic or logical computation. The value that the operator operates on is called the operand.

1) Arithmetic operators: Arithmetic operators are used to perform mathematical operations like addition, subtraction, multiplication, etc.

```
+ Add two operands or unary plus x + y+ 2:
- Subtract right operand from the left or unary minus x - y- 2
* Multiply two operands x * y
/ Divide left operand by the right one (always results into float) x /
y
% Modulus - remainder of the division of left operand by the right x %
y (remainder of x/y)
// Floor division - division that results into whole number adjusted to
the left in the number line x // y
** Exponent - left operand raised to the power of right x**y (x to the
power y)
"""
x = 3
y = 2
print('x + y =',x+y)
print('x - y =',x-y)
print('x * y =',x*y)
print('x / y =',x/y)
print('x // y =',x//y)
print('x ** y =',x**y)
```

2) Comparison operators: Comparison operators are used to compare values. It returns either True or False according to the condition.

```
**>** Greater than - True if left operand is greater than the right x >
y
< Less than - True if left operand is less than the right x < y
```

```
== Equal to - True if both operands are equal x == y
!= Not equal to - True if operands are not equal x != y

**>=** Greater than or equal to - True if left operand is greater than or
equal to the right x >= y
<= Less than or equal to - True if left operand is less than or equal to
the right x <= y
"""
x = 5
y = 10
print('x > y is',x>y)
print('x < y is',x<y)
print('x == y is',x==y)
print('x != y is',x!=y)
print('x >= y is',x>=y)
print('x <= y is',x<=y)
```

3) Logical operators: Logical operators are the and, or, not operators.

- *and*: True if both the operands are true x and y
- *or*: True if either of the operands is true x or y
- *not*: True if operand is false (complements the operand) not x

```
x = True
y = False
print('x and y is',x and y)    #true true
print('x or y is',x or y)     #either true
print('not x is',not x)
```

4) Assignment operators: Assignment operators are used in Python to assign values to variables. $a = 5$ is a simple assignment operator that assigns the value 5 on the right to the variable a on the left.

```
**=**      x = 5          x = 5

**+=**     x += 5        x = x + 5

**-=**     x -= 5        x = x - 5

*=         x *= 5         x = x * 5

**/=**     x /= 5        x = x / 5

a = 21
b = 10
c = 0
c = a + b
print ("Value of c is ", c)
c += a
print ("Value of c is ", c)
```



```
c *= a
print ("Value of c is ", c)
```

```
c /= a
print ("Value of c is ", c)
```

```
c = 2
c %= a
print ("Value of c is ", c)
```

```
c **= a
print ("Value of c is ", c)
```

```
c //= a
print ("Value of c is ", c)
```

5) Bitwise Operators: Bitwise operators are used to compare (binary) numbers:

- **& AND** Sets each bit to 1 if both bits are 1
- **| OR** Sets each bit to 1 if one of two bits is 1
- **^ XOR** Sets each bit to 1 if only one of two bits is 1
- **~ NOT** Inverts all the bits

```
a = 10      #1010   0101
b = 4       #0100
```

```
# Print bitwise AND operation
print("a & b =", a & b)      #0000
```

```
# Print bitwise OR operation
print("a | b =", a | b)     #1110
```

```
# Print bitwise NOT operation
print("~a =", ~a)          # ~a = ~1010
                           #   = -(1010 + 1)
                           #   = -(1011)
                           #   = -11 (Decimal)
```

```
# print bitwise XOR operation
print("a ^ b =", a ^ b)    # Returns 1 if one of the bits is 1 and
                           # the other is 0 else returns false.
```

6) Shift Operators: These operators are used to shift the bits of a number left or right thereby multiplying or dividing the number by two respectively.

Bitwise right shift: Shifts the bits of the number to the right and fills 0 on voids left(fills 1 in the case of a negative number) as a result.

Bitwise left shift: Shifts the bits of the number to the left and fills 0 on voids right as a result.

```
a = 10
b = -10

# print bitwise right shift operator
print("a >> 1 =", a >> 1)
print("b >> 1 =", b >> 1)

a = 5
b = -10

# print bitwise left shift operator
print("a << 1 =", a << 1)
print("b << 1 =", b << 1)
```

7) Identity operators: *is* and *is not* are the identity operators in Python. They are used to check if two values (or variables) are located on the same part of the memory.

```
x1 = 5
y1 = 5
x2 = 'Hello'
y2 = 'Hello'
x3 = [1,2,3]
y3 = [1,2,3]
print(x1 is not y1)      # Output: False
print(x2 is y2)         # Output: True
print(x3 is y3)         # Output: False
```

8) Membership operators: *in* and *not in* are the membership operators in Python. They are used to test whether a value or variable is found in a sequence (string, list, tuple, set and dictionary).

```
x = 'Hello world'
y = {1:'a',2:'b'}
print('H' in x)         # Output: True
print('hello' not in x) # Output: True
print(1 in y)           # Output: True
print('a' in y)         # Output: False
```

Functions

You use functions in programming to bundle a set of instructions that you want to use repeatedly. That means that a function is a piece of code written to carry out a specified task.

There are three types of functions in Python:

- **User-Defined Functions (UDFs)**, which are functions that users create to help them out.
- **Anonymous functions**, which are also called ****lambda functions**** because they are not declared with the standard `def` keyword.
- **Built-in functions**, such as `help()` to ask for help, `min()` to get the minimum value, `print()` to print an object to the terminal.

Creating a Function

```
def my_function():
    print("Hello from a function")

""**Calling** **a** **Function** ""

def my_function():
    print("Hello from a function")

my_function()

def my_function(fname):
    #A parameter is the variable
    #listed inside the parentheses in the function definition.
    print(fname + " Refsnes")

my_function("Emil")
#An argument is the value that
#is sent to the function when it is called.
my_function("Tobias")
my_function("Linus")
```

Number of Arguments

```
def my_function(fname, lname):
    print(fname + " " + lname)

my_function("Emil", "Refsnes")

def my_function(*kids):
    #do not know how many
    #arguments that will be passed into your function, add a * before the
    #parameter name in the function definition
    print("The youngest child is " + kids[2])

my_function("Emil", "Tobias", "Linus")

def my_function(child3, child2, child1):
    print("The youngest child is " + child3)

my_function(child1 = "Emil", child2 = "Tobias", child3 = "Linus")
```

```
#send arguments with the key = value syntax

def my_function(**kid):          #number of keyword arguments is
unknown, add a double ** before the parameter name
    print("His last name is " + kid["lname"])

my_function(fname = "Tobias", lname = "Refsnes")
```

Default Parameter Value

```
def my_function(country = "Norway"):
    print("I am from " + country)

my_function("Sweden")
my_function("India")
my_function()      #If we call the function without argument, it uses the
default value
my_function("Brazil")
```

Passing a List as an Argument

```
def my_function(food):
    for x in food:
        print(x)

fruits = ["apple", "banana", "cherry"]

my_function(fruits)
```

Return Values

```
def my_function(x):
    return 5 * x
print(my_function(3))
print(my_function(5))
print(my_function(9))
```

Python Lambda

A lambda function is a small anonymous function. A lambda function can take any number of arguments, but can only have one expression.

```
x = lambda a : a + 10
print(x(5))
Max = lambda a, b : a if(a > b) else b          # Example of lambda
function using if-else
print(Max(1, 2))
```

Difference Between Lambda functions and def defined function

```
def cube(y):
    return y*y*y
lambda_cube = lambda y: y*y*y
print(cube(5))
print(lambda_cube(5))
```

Python Built-In Functions

all(): The python all() function accepts an iterable object (such as list, dictionary, etc.). It returns true if all items in passed iterable are true. Otherwise, it returns False. If the iterable object is empty, the all() function returns True.

```
k = [1, 3, 4, 6]          # all values true
print(all(k))

k = [0, False]          # all values false
print(all(k))

k = [1, 3, 7, 0]        # one false value
print(all(k))

k = [0, False, 5]      ## one true value
print(all(k))

k = []                  # empty iterable
print(all(k))

test1 = []
print(test1,'is',bool(test1))
test1 = [0]
print(test1,'is',bool(test1))
test1 = 0.0
print(test1,'is',bool(test1))
test1 = None
print(test1,'is',bool(test1))
test1 = True
print(test1,'is',bool(test1))
test1 = 'Easy string'
print(test1,'is',bool(test1))

x = 10
print('Absolute value of -40 is:', abs(x))    #abs() function is used to
return the absolute value of a number
floating = -20.83
print('Absolute value of -20.83 is:', abs(floating))
y = bin(x) #bin() function is used to return the binary representation
of a specified integer.
print (y)
```

Python for Artificial Intelligence in Agriculture

```
s = sum([1, 2,4 ])          #sum() function is used to get the sum
of numbers of an iterable, i.e., list.
print(s)
print(float(9))            # float() function change into float
number
print(complex(9))          # complex() function change into complex
number
```

Data Structures, Control Structures and Loops in Python

Md. Ashraful Haque

ICAR-Indian Agricultural Statistics Research Institute, New Delhi - 110 012

ashraful.haque@icar.gov.in

Data structures in python:

Data Structures are the way of organizing, storing, manipulating, and accessing data in better way. The data structures enable us to can be access and update data in a more efficient manner depending upon the situation. Data Structures are fundamentals of any programming language around which a program is built. There are mainly four types of built-in data structures in python. Python helps to learn the fundamental of these data structures in a simpler way as compared to other programming languages. These data structures are-

- List
- Tuple
- Set
- Dictionary

1. List Data Structures:

List are used to store more than one data in single variable. In python lists are flexible i.e. it can store multiple data type in a single list.

The characteristics of Lists Data Structures in python are:

- Items are indexed (starting from 0)
- Items are ordered
- Items are changeable
- Lists allow duplicate values of items

Creation of List: Lists are created by placing the comma separated items inside the square brackets.

```
## creation of lists
list1 = ["apple", "banana", "cherry"]
list2 = [1, 5, 7, 9, 3]
list3 = [True, False, False]
list4 = [1, 2, 3, "GFG", 2.3]
list5 = [1,2,3,4,4,]
```

Accessing Items from List: Items of the lists can be access by mentioning the index or indices inside the square brackets.

```
## accessing items
```



```
list1 = ["apple", "banana", "cherry"]
list2 = [1, 5, 7, 9, 3, 6, 9, 2, 1, 10]
x = list1[0]
print(x)
y = list2[1:4]
print(y)
## new list
new_list = [1, 2, 3, 'example', 3.132, 10, 30]
#access all elements
print(new_list)
#access index 3 element
print(new_list[3])
#access elements from 0 to 1 and exclude 2
print(new_list[0:2])
#access elements in reverse
print(new_list[::-1])
```

Updating the list: Items in the list at particular position can be updated by mentioning the values in the left-hand side of the assignment operator.

```
list2 = [1, 5, 7, 9, 3, 6, 9, 2, 1, 10]
list2[2] = 34
print(list2)
```

Remove items: Items in the list at particular position can be deleted by del statement.

```
list2 = [1, 5, -12, 9, 3, 6, 9, 2, 1, 10]
del list2[2] print(list2)
```

Some common functions operate on list data structures:

```
## append(): adds an items or a list of items in at the end of a list
list1.append(list2)
## insert(): adds an items at a particular location of a list
list1.insert(1,'mango')
## remove(): deletes an item by its value from a list
list1.remove('banana')
## clear(): deletes all the elements from the
list list1.clear()
## index(): finds the index of the given element in the list
list1.index('mango')
## finds the count of the given element present in the list
list1.count('mango')
#sorted(): temporarily sorts the elements of the list
sorted(list1)
#sort(): permanantly sorts the elements of the list
list1.sort(reverse=True)
```

Some basic operations on list data structures:

```
## Get number of items in a list
n = len(list1)
## Concatenate two lists together
list_new = list1 + list2
## check membership of an item in a list
100 in list2 # (gives true or false)
```

2. Tuple Data Structure:

Tuples are sequence of immutable objects in python. Tuples can store more than one datatype in a single instance of tuple.

The characteristics of Tuple Data Structures in python are:

- Items are indexed (starting from 0)
- Items are ordered
- Items are non-changeable
- Tuples allow duplicate values for the items

Creation of tuples: Tuples are created by placing the comma separated items inside the round brackets or parenthesis.

```
tuple1 = ("apple", "banana", "cherry")
tuple2 = (1, 5, 7, 9, 3)
tuple3 = (True, False, False)
```

Accessing items from Tuple: Items of the tuple can be access by mentioning the index or indices inside the square brackets.

```
## accessing items
tuple1 = ("apple", "banana", "cherry")
tuple2 = (1, 5, 7, 9, 3, 6, 9, 2, 1, 10)
x = tuple1[0]
print(x)
y = tuple2[1:4]
print(y)
```

Updating the tuple: Items in the tuples can't be changes once the tuple is created.

```
tuple2 = (1, 5, 7, 9, 3, 6, 9, 2, 1, 10)
tuple2[2]= 34 ## will raise an error print(tuple2)
```

Remove items: Items in the tuple can't be deleted as tuples are immutable. However, del statement can be used to delete whole tuple instead.

```
tup = ('physics', 'chemistry', 1997, 2000)
print(tup)
del tup
print(tup) ## will raise an error
```

Some basic operations on tuple data structures:

```
## Get number of items in a tuple
n = len(tuple1)
```

```
tuple_new = tuple1 + tuple2
## check membership of an item in a tuple
100 in tuple2 # (gives true or false)
```

3. Set Data Structure:

Mathematically, a set is a collection of items in any order. The sets in python are typically used for mathematical operations like union, intersection, difference and complement etc.

The characteristics of Set Data Structures in python are:

- Items are unindexed
- Items are unordered
- Items are non-changeable.
- Sets doesn't allow duplicate values

Creation of Sets: Sets are created by placing the comma separated items inside curly brackets.

```
set1 = {"apple", "banana", "cherry"}
set2 = {1, 5, 7, 9, 3}
set3 = {True, False, False}
```

Accessing items in Sets: Items in the sets can't be access by mentioning the index number. For accessing the items in the Sets one can use any loop structure.

```
Days=set(["Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"])
for d in Days:
    print(d)
```

Adding and deleting items: In Sets, a new item can be added using *add()* function and an existing item can be deleted by *discard()* function.

```
Days=set(["Mon", "Tue", "Wed", "Thu", "Fri", "Sat"])
Days.add("Sun")
print(Days)
Days.discard("Mon")
print(Days)
```

Different set operations:

Union of Sets: The union operation on two sets produces a new set containing all the distinct elements from both the sets. In the below example the element “Wed” is present in both the sets. Here, pipe (|) operator is used.

```
DaysA = set(["Mon", "Tue", "Wed"])
DaysB = set(["Wed", "Thu", "Fri", "Sat", "Sun"])
AllDays = DaysA|DaysB
print(AllDays)
```

Intersection of Sets: The intersection operation on two sets produces a new set containing only the common elements from both the sets. Here, ampersand (&) operator is used.

```
DaysA = set(["Mon", "Tue", "Wed"])
DaysB = set(["Wed", "Thu", "Fri", "Sat", "Sun"])
AllDays = DaysA & DaysB
print(AllDays)
```

Difference of Sets: The difference operation on two sets produces a new set containing only the elements from the first set and none from the second set. Here, minus (-) operator is used.

```
DaysA = set(["Mon", "Tue", "Wed"])
DaysB = set(["Wed", "Thu", "Fri", "Sat", "Sun"])
AllDays = DaysA - DaysB
print(AllDays)
```

Compare Sets: We can check if a given set is a subset or superset of another set.

```
DaysA = set(["Mon", "Tue", "Wed"])
DaysB = set(["Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"])
SubsetRes = DaysA <= DaysB
SupersetRes = DaysB >= DaysA
print(SubsetRes)
print(SupersetRes)
```

4. Dictionary Data Structure:

Dictionaries are the type of data structure that are used to store data in key:value pair. In Dictionary, each key is separated from its value by a colon (:), the items are separated by commas, and the whole thing is enclosed in curly braces.

The characteristics of Dictionaries Data Structures in python are:

- Items are ordered
- Items are changeable
- Dictionary doesn't allow duplicate values

Creation of dictionaries: Dictionaries are created by placing the comma separated key:values pairs inside curly brackets.

```
dictionary1 = {"brand": "Ford",
               "model": "Mustang",
               "year": 1964}
x = dictionary1 ["model"]
print(x)
```

Accessing items: Items can be accessed by mentioning the key name inside the square bracket.

```
dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}
print ("dict['Name']: ", dict['Name'])
```

```
print ("dict['Age']: ", dict['Age'])
```

Updating Dictionary: One can update a dictionary by adding a new entry or a key-value pair, modifying an existing entry, or deleting an existing entry.

```
dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}
dict['Age'] = 8;
# update existing entry
dict['School'] = "DPS School"
# Add new entry
print ("dict['Age']: ", dict['Age'])
print ("dict['School']: ", dict['School'])
```

Delete Dictionary Elements: One can either remove individual dictionary elements or clear the entire contents of a dictionary.

```
dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}
del dict['Name'] # remove entry with key 'Name'
dict.clear() # remove all entries in dict
del dict # delete entire dictionary
print ("dict['Age']: ", dict['Age'])
print ("dict['School']: ", dict['School'])
```

Control Structures in Python:

There is mainly one control structure that is *if...else*. The *if...else* structures are used to implement the logical conditions of the program and allow the program to branch based on the evaluation of an expression.

General syntax of *if...else* :

```
if expression :
    statement 1
    statement 2
    ...
statement n else:
    statement 1
    statement 2
    ...
statement always executed
```

N.B. Indentation in the control and loop structures are very crucial in case of python programming language.

```
## examples of if..else
## if statement value = 5
threshold= 4
print("value is", value, "threshold is ",threshold)
if value > threshold :
    print(value, "is bigger than ", threshold)
## if..else statement a = 330 b = 200 if b > a:
```

```
print("b is greater than a") else: print("error")
```

Nested control structures: The *if..else* structures can be used in nested manner by using *elif* statement.

```
## nested if.. statements
### if ... elif ... else ...
a = 5
b = 4
print("a = ", a, "and b = ", b)
if a > b :
    print(a, " is greater than ", b)
elif a == b :
    print(a, " equals ", b)
else :
    print(a, " is less than ", b)
```

Loop Structures in Python:

In python generally two types of loop structures are used: while loop and for loop.

1. while loop:

With the ‘while’ loop, a set of statements can be executed repeatedly long as a condition is true . For the loop to terminate, there has to be some termination criteria mentioned in the code which will potentially change the condition and stop the iteration.

```
## Simple example
i=1
while i < 6:
    print(i)
    i = i + 1
## sum of n numbers using a while loop
n = 10
cur_sum = 0
i = 1
while i <= n :
    cur_sum = cur_sum + i
    i = i + 1
print("The sum of the numbers from 1 to", n, "is ", cur_sum)
```

Points to note:

- Here, the conditional clause ($i \leq n$) in the *while* statement can be anything which would return a boolean value of either *True* or *False* upon execution.
- Initially *i* has been set to 1 (before the start of the loop) and therefore the condition is *True*.
- The clause can be made more complex by using parentheses, *and* and *or* operators amongst others

- The statements after the *while* clause are only executed if the condition evaluates as *True*.
- Within the statements after the *while* clause there should be something which potentially will make the condition evaluate as *False* next time around. If not the loop will never end.
- In this case the last statement in the loop changes the value of *i* which is part of the condition clause, so hopefully the loop will end.

2. *for* loop:

A *for* loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string) for executing a set of statements. The difference between *while* and *for* loop is that in *for* loop we know that at the outset how often the statements in the loop will be executed, we don't have to rely on a variable being changed within the looping statements as in *while* loop.

General syntax of *for* loop:

```
for variable_name in some_sequence :
    statement1
    statement2
    ...
    statementn

## simple example
for i in [1,2,3] :
    print(i)
print("\nExample 1\n")
fruits = ["apple", "banana", "cherry"]
for x in fruits:
    print(x)

print("\nExample 2\n")
for x in "banana":
    print(x)
print("\nExample 3\n")
for name in ["Tom", 42, 3.142] :
    print(name)
print("\nExample 4\n")
for i in range(10) :
    print(i)
print("\nExample 5\n")
longString = "The quick brown fox jumped over the lazy sleeping do"
for word in longString.split() :
    print(word)
```

Functions, Module, File Handling in Python

Akshay Dheeraj

ICAR-Indian Agricultural Statistics Research Institute, New Delhi - 110 012

akshay.dhreeraj@icar.gov.in

Introduction to Functions

In Python, the **function is a block of code defined with a name**. We use functions whenever we need to perform the same task multiple times without writing the same code again. It can take arguments and returns the value.

Python has a DRY principle like other programming languages. DRY stands for Don't Repeat Yourself. Consider a scenario where we need to do some action/task many times. We can define that action only once using a function and call that function whenever required to do the same activity.

Function improves efficiency and reduces errors because of the reusability of a code. Once we create a function, we can call it anywhere and anytime. The benefit of using a function is reusability and modularity.

Types of Functions

Python support two types of functions

1. Built-in function
2. User-defined function

Built-in function

The functions which are come along with Python itself are called a built-in function or **predefined function**. Some of them are listed below. range(), type(), input(), eval() etc.

Example: Python range() function generates the immutable sequence of numbers starting from the given start integer to the stop integer.

```
for i in range(1, 10):  
    print(i, end=' ')  
# Output 1 2 3 4 5 6 7 8 9
```

User-defined function

Functions which are created by programmer explicitly according to the requirement are called a user-defined function.

Creating a Function

Use the following steps to define a function in Python.

- Use the def keyword with the function name to define a function.
- Next, pass the number of parameters as per your requirement. (Optional).
- Next, define the function body with a **block of code**. This block of code is nothing but the action you want to perform.

In Python, no need to specify curly braces for the function body. The only **indentation** is essential to separate code blocks. Otherwise, you will get an error.

Syntax of creating a function

```
def function_name(parameter1, parameter2):  
    # function body  
    # write some action  
return value
```

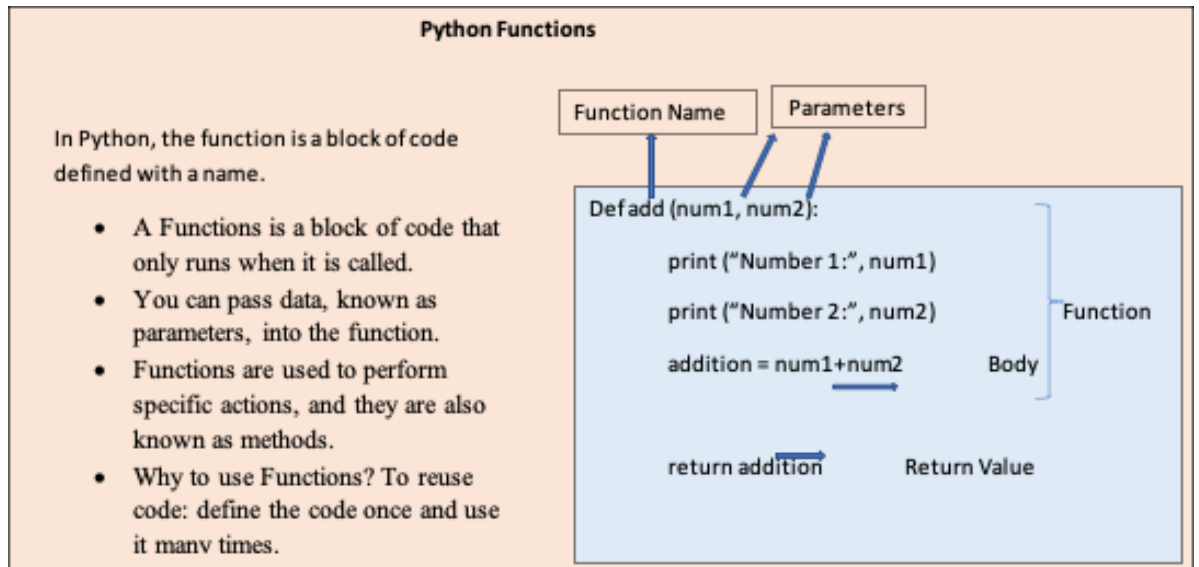
Here,

- **function_name:** Function name is the name of the function. We can give any name to function.
- **parameter:** Parameter is the value passed to the function. We can pass any number of parameters. Function body uses the parameter's value to perform an action
- **function_body:** The function body is a block of code that performs some task. This block of code is nothing but the action you wanted to accomplish.
- **return value:** Return value is the output of the function.

Note: While defining a function, we use two keywords, def (mandatory) and return (optional).

Creating a function without any parameters

```
# function  
def message():  
    print("Welcome participants to ICAR-IASRI")  
  
# call function using its name  
message()
```



Creating a function with parameters

Let's create a function that takes two parameters and displays their values.

In this example, we are creating function with two parameters ' name' and 'age'.

```
# function  
def training_func(name, training_name):  
    print("Hello", name, "Welcome to ICAR-IASRI")  
    print("Your training title is", training_name)  
  
# call function  
training_func ('XYZ', 'Python')
```

Creating a function with parameters and return value

Functions can return a value. The return value is the output of the function. Use the return keyword to return value from a function.

```
# function  
def calculator(a, b):  
    add = a + b  
    # return the addition  
    return add  
  
# call function  
# take return value in variable  
res = calculator(20, 5)  
  
print("Addition :", res)  
# Output Addition : 25
```

Calling a function

Once we defined a function or finalized structure, we can call that function by using its name. We can also call that function from another function or program by importing it.

To call a function, use the name of the function with the parenthesis, and if the function accepts parameters, then pass those parameters in the parenthesis.

```
# function
def even_odd(n):
    # check number is even or odd
    if n % 2 == 0:
        print('Even number')
    else:
        print('Odd Number')

# calling function by its name
even_odd(19)
# Output Odd Number
```

Calling a function of a module

You can take advantage of the built-in module and use the functions defined in it. For example, Python has a random module that is used for generating random numbers and data. It has various functions to create different types of random data.

Let's see how to use functions defined in any module.

- First, we need to use the import statement to import a specific function from a module.
- Next, we can call that function by its name.

```
# import randint function
from random import randint

# call randint function to get random number
print(randint(10, 20))
# Output 14
```

Docstrings

In Python, the documentation string is also called a **docstring**. It is a descriptive text (like a comment) written by a programmer to let others know what block of code does. We write docstring in source code and define it immediately after module, class, function, or method definition

It is being declared using triple single quotes (" ") or triple-double quote("""" """).

We can access docstring using doc attribute (`__doc__`) for any object like list, tuple, dict, and user-defined function, etc.

Single-Line Docstring

The single-line docstring is a docstring that fits in one line. We can use the triple single or triple-double quotes to define it. The Opening and closing quotes need to be the same. By convention, we should use to use the triple-double quotes to define docstring.

```
def factorial(x):
    """This function returns the factorial of a given number."""
    return x
# access doc string
print(factorial.__doc__)
```

When you use the help function to get the information of any function, it returns the docstring.

```
# pass function name to help() function
print(help(factorial))
```

Help on function factorial in module **main**:

```
factorial(x)
    This function returns the factorial of a given number.
None
```

Multi-Line Docstring

A multi-line Docstrings is the same single-line Docstrings, but it is followed by a single blank line with the descriptive text.

The general format of writing a multi-line Docstring is as follows:

Example

```
def any_fun(parameter1):
    """
    Description of function

    Arguments:
    parameter1(int):Description of parameter1

    Returns:
    int value
    """
print(any_fun.__doc__)
```

Output

Description of function

Arguments

parameter1(int):Description of parameter1

Returns:

int value

Return Value From a Function

In Python, to return value from the function, a return statement is used. It returns the value of the expression following the returns keyword.

Syntax of return statement

```
def fun():  
    statement-1  
    statement-2  
    statement-3  
    .  
    .  
    return [expression]
```

The return value is nothing but a outcome of function.

- The return statement ends the function execution.
- For a function, it is not mandatory to return a value.
- If a return statement is used without any expression, then the None is returned.
- The return statement should be inside of the function block.

```
def is_even(list1):  
    even_num = []  
    for n in list1:  
        if n % 2 == 0:  
            even_num.append(n)  
    # return a list  
    return even_num  
  
# Pass list to the function  
even_num = is_even([2, 3, 42, 51, 62, 70, 5, 9])  
print("Even numbers are:", even_num)
```

Output

Even numbers are: [2, 42, 62, 70]

Return Multiple Values

You can also return multiple values from a function. Use the return statement by separating each expression by a comma.

Example: –

In this example, we are returning three values from a function. We will also see how to process or read multiple return values in our code.

```
def arithmetic(num1, num2):
    add = num1 + num2
    sub = num1 - num2
    multiply = num1 * num2
    division = num1 / num2
    # return four values
    return add, sub, multiply, division

# read four return values in four variables
a, b, c, d = arithmetic(10, 2)

print("Addition: ", a)
print("Subtraction: ", b)
print("Multiplication: ", c)
print("Division: ", d)
```

Scope and Lifetime of Variables

When we define a function with variables, then those variables' scope is limited to that function. In Python, the scope of a variable is an area where a variable is declared. It is called the variable's local scope.

We cannot access the local variables from outside of the function. Because the scope is local, those variables are not visible from the outside of the function.

Note: The inner function does have access to the outer function's local scope.

When we are executing a function, the life of the variables is up to running time. Once we return from the function, those variables get destroyed. So function does no need to remember the value of a variable from its previous call.

The following code shows the scope of a variable inside a function.

Example

```
global_lang = 'DataScience'

def var_scope_test():
    local_lang = 'Python'
```

```
print(local_lang)

var_scope_test()
# Output 'Python'

# outside of function
print(global_lang)
# Output 'DataScience'

# NameError: name 'local_lang' is not defined
print(local_lang)
```

In the above example, we print the local and global variable values from outside of the function. The global variable is accessible with its name `global_lang`.

But when we try to access the local variable with its name `local_lang`, we got a `NameError`, because the local variable is not accessible from outside of the function.

Example

```
def function1():
    # local variable
    loc_var = 888
    print("Value is :", loc_var)

def function2():

    print("Value is :", loc_var)

function1()
function2()
```

Output

```
Value is : 888
print("Value is :", loc_var) # gives error,
NameError: name 'loc_var' is not defined
```

Global Variable in function

A Global variable is a variable that is declared outside of the function. The scope of a global variable is broad. It is accessible in all functions of the same module.

Example

```
global_var = 999

def function1():
    print("Value in 1nd function :", global_var)

def function2():
```

```
print("Value in 2nd function :", global_var)

function1()
function2()
```

Output

```
Value in 1nd function : 999
Value in 2nd function : 999
```

Global Keyword in Function

In Python, `global` is the keyword used to access the actual global variable from outside the function. we use the `global` keyword for two purposes:

1. To declare a global variable inside the function.
2. Declaring a variable as `global`, which makes it available to function to perform the modification.

Let's see what happens when we don't use `global` keyword to access the global variable in the function

```
# Global variable
global_var = 5

def function1():
    print("Value in 1st function :", global_var)

def function2():
    # Modify global variable
    # function will treat it as a local variable
    global_var = 555
    print("Value in 2nd function :", global_var)

def function3():
    print("Value in 3rd function :", global_var)
```

```
function1()
function2()
function3()
```

Output

```
Value in 1st function : 5
Value in 2nd function : 555
Value in 3rd function : 5
```


As you can see, function2() treated global_var as a new variable (local variable). To solve such issues or access/modify global variables inside a function, we use the global keyword.

```
# Global variable
x = 5

# defining 1st function
def function1():
    print("Value in 1st function :", x)

# defining 2nd function
def function2():
    # Modify global variable using global keyword
    global x
    x = 555
    print("Value in 2nd function :", x)

# defining 3rd function
def function3():
    print("Value in 3rd function :", x)

function1()
function2()
function3()
```

Output

```
Value in 1st function : 5
Value in 2nd function : 555
Value in 3rd function : 555
```

Python Function Arguments

The argument is a value, a variable, or an object that we pass to a function or method call. In Python, there are four types of arguments allowed.

1. Positional arguments
2. keyword arguments
3. Default arguments
4. Variable-length arguments

Positional Arguments

Positional arguments are arguments that are pass to function in **proper positional order**. That is, the 1st positional argument needs to be 1st when the function is called.

The 2nd positional argument needs to be 2nd when the function is called, etc. See the following example for more understanding.

Example

```
def add(a, b):  
    print(a - b)  
  
add(50, 10)  
# Output 40  
add(10, 50)  
# Output -40
```

Keyword Arguments

A keyword argument is an argument value, passed to function preceded by the variable name and an equals sign.

Example

```
def message(name, surname):  
    print("Hello", name, surname)  
  
message(name="A", surname="B")  
message(surname="C", name="D")
```

Output

```
Hello A B  
Hello C D
```

In keyword arguments order of argument is not matter, but the number of arguments must match. Otherwise, we will get an error.

While using keyword and positional argument simultaneously, we need to pass 1st arguments as positional arguments and then keyword arguments. Otherwise, we will get Syntax Error. See the following example.

Example

```
def message(first_nm, last_nm):  
    print("Hello..!", first_nm, last_nm)  
  
# correct use  
message("A", "B")  
message("A", last_nm="B")  
  
# Error  
# SyntaxError: positional argument follows keyword argument  
message(first_nm="A", "B")
```

Default Arguments

Default arguments take the default value during the function call if we do not pass them. We can assign a default value to an argument in function definition using the = assignment operator.

Example

```
# function with default argument
def message(name="Guest"):
    print("Hello", name)

# calling function with argument
message("John")

# calling function without argument
message()
```

Output
Hello John
Hello Guest

Variable-length Arguments

In Python, sometimes, there is a situation where we need to pass multiple numbers of arguments to the function. Such types of arguments are called **variable-length arguments**. We can declare a variable-length argument with the * (**asterisk**) symbol.

```
def fun(*var):
    function body
```

We can pass any number of arguments to this function. Internally all these values are represented in the form of a **tuple**.

Example

```
def addition(*numbers):
    total = 0
    for no in numbers:
        total = total + no
    print("Sum is:", total)

# 0 arguments
addition()

# 5 arguments
```

```
addition(10, 5, 2, 5, 4)
```

```
# 3 arguments  
addition(78, 7, 2.5)
```

Output

```
Sum is: 0  
Sum is: 26  
Sum is: 87.5
```

Recursive Function

A recursive function is a function that calls itself, again and again.

Consider, calculating the factorial of a number is a repetitive activity, in that case, we can call a function again and again, which calculates factorial.

```
def factorial(no):  
    if no == 0:  
        return 1  
    else:  
        return no * factorial(no - 1)  
  
print("factorial of a number is:", factorial(8))
```

Output

```
factorial of a number is: 40320
```

The advantages of the recursive function are:

1. By using recursive, we can reduce the length of the code.
2. The readability of code improves due to code reduction.
3. Useful for solving a complex problem

The disadvantage of the recursive function:

1. The recursive function takes more memory and time for execution.
2. Debugging is not easy for the recursive function.

Python Anonymous/Lambda Function

Sometimes we need to declare a function without any name. The nameless property function is called an **anonymous function** or **lambda function**.

The reason behind the using anonymous function is for instant use, that is, one-time usage. Normal function is declared using the def function. Whereas the anonymous function is declared using the lambda keyword.

A Python lambda function is a single expression. But, in a lambda body, we can expand with expressions over multiple lines using parentheses or a multiline string.

ex : lambda n:n+n

Syntax of lambda function:

lambda: argument_list:expression

When we define a function using the lambda keyword, the code is very concise so that there is more readability in the code. A lambda function can have any number of arguments but return only one value after expression evaluation.

Let's see an example to print even numbers without a lambda function and with a lambda function. See the difference in line of code as well as readability of code.

Example 1: Program for even numbers without lambda function

```
def even_numbers(nums):
    even_list = []
    for n in nums:
        if n % 2 == 0:
            even_list.append(n)
    return even_list

num_list = [10, 5, 12, 78, 6, 1, 7, 9]
ans = even_numbers(num_list)
print("Even numbers are:", ans)
```

Output

Even numbers are: [10, 12, 78, 6]

Example 2: Program for even number with a lambda function

```
l = [10, 5, 12, 78, 6, 1, 7, 9]
even_nos = list(filter(lambda x: x % 2 == 0, l))
print("Even numbers are: ", even_nos)
```

Output

Even numbers are: [10, 12, 78, 6]

We are not required to write **explicitly return statements** in the lambda function because the lambda internally returns expression value.

Lambda functions are more useful when we pass a function as an argument to another function. We can also use the lambda function with built-in functions such as filter, map, reduce because this function requires another function as an argument.

filter() function in Python

In Python, the filter() function is used to return the filtered value. We use this function to filter values based on some conditions.

Syntax of filter() function:

filter(function, sequence)

where,

- function – Function argument is responsible for performing condition checking.
- sequence – Sequence argument can be anything like list, tuple, string

Example: lambda function with filter()

```
l = [-10, 5, 12, -78, 6, -1, -7, 9]
positive_nos = list(filter(lambda x: x > 0, l))
print("Positive numbers are: ", positive_nos)
```

Output

```
Positive numbers are: [5, 12, 6, 9]
```

Python Modules

In Python, modules refer to the Python file, which contains Python code like Python statements, classes, functions, variables, etc. A file with Python code is defined with extension.py

For example: In Test.py, where the test is the module name.

In Python, large code is divided into small modules. The benefit of modules is, it provides a way to share reusable functions.

Types of modules

In Python, there are two types of modules.

1. Built-in Modules
2. User-defined Modules

Built-in modules

Built-in modules come with default Python installation. One of Python's most significant advantages is its rich library support that contains lots of built-in modules. Hence, it provides a lot of reusable code.

Some commonly used Python built-in modules are datetime, os, math, sys, random, etc.

User-defined modules

The modules which the user defines or create are called a **user-defined module**. We can create our own module, which contains classes, functions, variables, etc., as per our requirements.

How to import modules?

In Python, the import statement is used to import the whole module. Also, we can import specific classes and functions from a module.

import module name.

When the interpreter finds an import statement, it imports the module presented in a search path. The module is loaded only once, even we import multiple times.

To import modules in Python, we use the Python import keyword. With the help of the import keyword, both the **built-in** and **user-defined** modules are imported. Let's see an example of importing a math module.

```
import math
# use math module functions
print(math.sqrt(5))
# Output 2.23606797749979
```

Import multiple modules

If we want to use more than one module, then we can import multiple modules. This is the simplest form of import statement that we already used in the above example.

Syntax of import statement:

```
import module1,module2,.. moduleN
```

Example

```
# Import two modules
import math, random

print(math.factorial(5))
print(random.randint(10, 20))
```

```
Output
120
18
```

Import only specific classes or functions from a module

To import particular classes or functions, we can use the from...import statement. It is an alternate way to import. Using this way, we can import individual attributes and methods directly into the program.

In this way, we are not required to use the module name. See the following example.

Syntax of from...import statement:

```
from <module_name> import <name(s)>
```

Example

```
# import only factorial function from math module
from math import factorial
print(factorial(5))
```

Output

120

Import with renaming a module

If we want to use the module with a different name, we can use from..import...as statement.

It is also possible to import a particular method and use that method with a different name. It is called **aliasing**. Afterward, we can use that name in the entire program.

Syntax of from..import ..as keyword:

```
from <module_name> import <name> as <alternative_name>
```

Example 1: Import a module by renaming it

```
import random as rand
print(rand.randrange(10, 20, 2))
```

Output

16

Example 2: Import a method by renaming it

```
# rename randint as random_number
from random import randint as random_number

# Gives any random number from range(10, 50)
print(random_number(10, 50))
```

Output

32

Import all names

If we need to import all functions and attributes of a specific module, then instead of writing all function names and attribute names, we can import all using an **asterisk ***.

Syntax of import * statement:

```
import *
```

Example


```
from math import *
print(pow(4,2))
print(factorial(5))

print(pi*3)
print(sqrt(100))
```

Output

```
16.0
120
9.42477796076938
10.0
```

Create Module

In Python, to create a module, write Python code in the file, and save that file with the.py extension. Here our module is created.

Example

```
def my_func():
    print("Welcome to ICAR-IASRI")
```

Output

```
Welcome to ICAR-IASRI
```

Variables in Module

In Python, the module contains Python code like classes, functions, methods, but it also has variables. A variable can list, tuple, dict, etc.

Let's see this with an example:

First, create a Python module with the name test_module.py and write the below code in that file.

Example

```
cities_list = ['Mumbai', 'Delhi', 'Bangalore', 'Karnataka', 'Hyderabad']
```

Now, create a Python file with the name test_file.py, write the below code and import the above module test_module.py in that file. See the following code.

```
import test_module
# access first city
city = test_module.cities_list[1]
print("Accessing 1st city:", city)

# Get all cities
cities = test_module.cities_list
print("Accessing All cities :", cities)
```

When we execute this test_file.py, the variable of test_module.py is accessible using the dot(.)operator.

Output

Accessing 1st city: Delhi

Accessing All cities : ['Mumbai', 'Delhi', 'Bangalore', 'Karnataka', 'Hyderabad']

File Handling in Python

File handling is an important part of any web application. Python too supports file handling and allows users to handle files i.e., to read and write files, along with many other file handling options, to operate on files. Python treats files differently as text or binary and this is important. Each line of code includes a sequence of characters and they form a text file. Each line of a file is terminated with a special character, called the EOL or End of Line characters like comma {,} or newline character. It ends the current line and tells the interpreter a new one has begun.

Types of File

- **Binary File:** The binary files are used to store binary data such as images, video files, audio files, etc.
- **Text File:** Text file usually we use to store character data. For example, test.txt

Binary files in Python

Most of the files that we see in our computer system are called binary files.

Example:

1. **Document files:** .pdf, .doc, .xls etc.
2. **Image files:** .png, .jpg, .gif, .bmp etc.
3. **Video files:** .mp4, .3gp, .mkv, .avi etc.
4. **Audio files:** .mp3, .wav, .mka, .aac etc.
5. **Database files:** .mdb, .accde, .frm, .sqlite etc.
6. **Archive files:** .zip, .rar, .iso, .7z etc.
7. **Executable files:** .exe, .dll, .class etc.

Text files in Python

Text files don't have any specific encoding and it can be opened in normal text editor itself.

Example:

- **Web standards:** html, XML, CSS, JSON etc.
- **Source code:** c, app, js, py, java etc.
- **Documents:** txt, tex, RTF etc.
- **Tabular data:** csv, tsv etc.
- **Configuration:** ini, cfg, reg etc.

Python File Handling Operations

Most importantly there are 4 types of operations that can be handled by Python on files:

- Open
- Read
- Write
- Close

Other operations include:

- Rename
- Delete

Python Create and Open a File

Python has an in-built function called `open()` to open a file. It takes a minimum of one argument as mentioned in the below syntax. The open method returns a file object which is used to access the write, read and other in-built methods.

Here, `file_name` is the name of the file or the location of the file that you want to open, and `file_name` should have the file extension included as well. Which means in `test.txt` – the term `test` is the name of the file and `.txt` is the extension of the file.

The mode in the open function syntax will tell Python as what operation you want to do on a file.

- **‘r’ – Read Mode:** Read mode is used only to read data from the file.
- **‘w’ – Write Mode:** This mode is used when you want to write data into the file or modify it. Remember write mode overwrites the data present in the file.
- **‘a’ – Append Mode:** Append mode is used to append data to the file. Remember data will be appended at the end of the file pointer.
- **‘r+’ – Read or Write Mode:** This mode is used when we want to write or read the data from the same file.

- **‘a+’ – Append or Read Mode:** This mode is used when we want to read data from the file or append the data into the same file.

Note: The above-mentioned modes are for opening, reading or writing text files only. While using binary files, we have to use the same modes with the letter **‘b’** at the end. So that Python can understand that we are interacting with binary files.

- **‘wb’** – Open a file for write only mode in the binary format.
- **‘rb’** – Open a file for the read-only mode in the binary format.
- **‘ab’** – Open a file for appending only mode in the binary format.
- **‘rb+’** – Open a file for read and write only mode in the binary format.
- **‘ab+’** – Open a file for appending and read-only mode in the binary format.

Example 1:

```
fo=open("/Users/akshaydheeraj/Desktop/Python_Practice/test.txt","r+")
```

In the above example, we are opening the file named ‘test.txt’ present at the location ‘C:/Documents/Python/’ and we are opening the same file in a read-write mode which gives us more flexibility.

Let’s create the file named test.txt with the sample text as

```
Hello everyone! Welcome to ICAR-IASRI  
Good morning.  
How are you?
```

Python Read From File

In order to read a file in python, we must open the file in read mode.

There are three ways in which we can read the files in python.

- `read([n])`
- `readline([n])`
- `readlines()`

Here, n is the number of bytes to be read.

Here we are opening the file test.txt in a read-only mode and are reading only the first 5 characters of the file using the **fo.read(5)** method.

```
print(fo.read(5))
```

Output:

```
Hello
```

Here we have not provided any argument inside the **read()** function. Hence it will read all the content present inside the file.

```
print(fo.read())
```

Output:

```
Hello everyone! Welcome to ICAR-IASRI
```

```
Good morning.
```

```
How are you?
```

The **readline()** method reads the lines of the file from the beginning, i.e., if we use the **readline()** method two times, then we can get the first two lines of the file.

```
print(fo.readline())
```

Output:

```
Hello everyone! Welcome to ICAR-IASRI
```

Python provides also the **readlines()** method which is used for the reading lines. It returns the list of the lines till the end of **file(EOF)** is reached.

```
print(fo.readlines())
```

Output:

```
['Hello everyone! Welcome to ICAR-IASRI\n', 'Good morning.\n', 'How are you?\n', '\n']
```

Python Write to File

In order to write data into a file, we must open the file in write mode.

We need to be very careful while writing data into the file as it overwrites the content present inside the file that you are writing, and all the previous data will be erased.

We have two methods for writing data into a file as shown below.

- `write(string)`
- `writelines(list)`

Example 1:

```
my_file = open("C:/Documents/Python/test.txt", "w")
```

```
my_file.write("Hello World")
```

The above code writes the String 'Hello World' into the 'test.txt' file.

Example 2:

```
my_file = open("C:/Documents/Python/test.txt", "w")
```

```
my_file.write("Hello World\n")
my_file.write("Hello Python")
```

The first line will be 'Hello World' and as we have mentioned \n character, the cursor will move to the next line of the file and then write 'Hello Python'.

Remember if we don't mention \n character, then the data will be written continuously in the text file like 'Hello WorldHelloPython'

```
fruits = ["Apple\n", "Orange\n", "Grapes\n", "Watermelon"]
my_file = open("C:/Documents/Python/test.txt", "w")
my_file.writelines(fruits)
```

Python Append to File

To append data into a file we must open the file in 'a+' mode so that we will have access to both the append as well as write modes.

Example 1:

```
my_file = open("C:/Documents/Python/test.txt", "a+")
my_file.write ("Strawberry")
```

The above code appends the string 'Apple' at the **end** of the 'test.txt' file.

Python Close File

In order to close a file, we must first open the file. In python, we have an in-built method called close() to close the file which is opened.

Whenever you open a file, it is important to close it, especially, with write method. Because if we don't call the close function after the write method then whatever data we have written to a file will not be saved into the file.

Example 1:

```
my_file = open("C:/Documents/Python/test.txt", "r")
print(my_file.read())
my_file.close()
```

Python Rename or Delete File

Python provides us with an "os" module which has some in-built methods that would help us in performing the file operations such as renaming and deleting the file.

In order to use this module, first of all, we need to import the "os" module in our program and then call the related methods.

rename() method:

This rename() method accepts two arguments i.e. the current file name and the new file name.

Syntax:

```
os.rename(current_file_name, new_file_name)
```

Example 1:

```
import os  
os.rename("test.txt", "test1.txt")
```

Here 'test.txt' is the current file name and 'test1.txt' is the new file name.

You can specify the location as well as shown in the below example.

Example 2:

```
import os  
os.rename("C:/Documents/Python/test.txt", "C:/Documents/Python/test1.txt")
```

OOPs concepts and Exception handling

Madhu

ICAR-Indian Agricultural Statistics Research Institut, New Delhi - 110 012
madhu.dahiya@icar.gov.in

Introduction:

Procedural programming is about writing procedures or functions that perform operations on the data, while object-oriented programming is about creating objects that contain both data and functions.

Object-Oriented Paradigm

1. Emphasis is on data rather than procedure.
2. Programs are divided into objects.
3. Data Structures are designed such that they Characterize the objects.
4. Methods that operate on the data of an object are tied together in the data structure.
5. Data is hidden and cannot be accessed by external functions.
6. Objects may communicate with each other through methods.

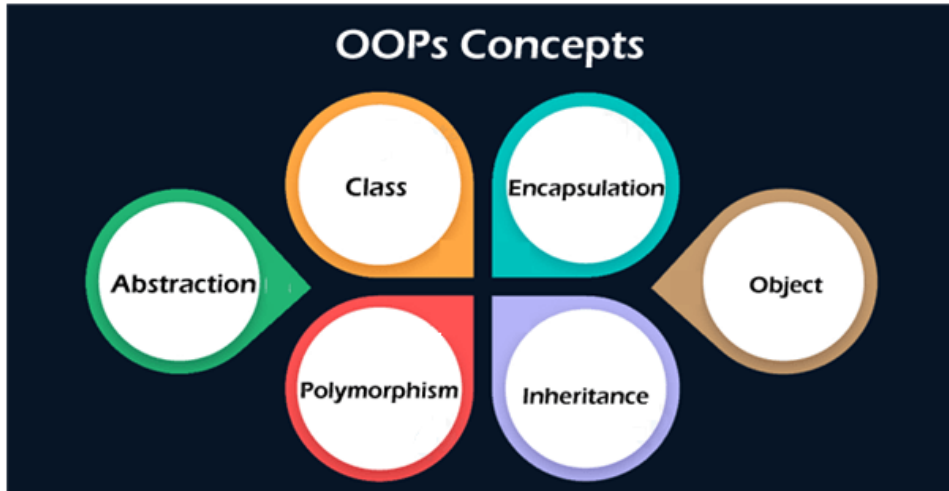
Difference between Procedural and Object oriented programming

Procedural Oriented Programming	Object Oriented Programming
Program is divided into smaller parts called functions.	Program is divided into smaller parts called objects.
It follows top down approach.	It follows bottom up approach.
There is no access specifier in it.	It have access specifiers like public, private, protected.
Adding new data and function is not easy.	Adding new data and function is easy.
Functions are more important than data.	Data is more important than Functions.
It is based on unreal world.	It is based on real world.
It does not have proper way of hiding data hence it is less secure.	It provides data hiding hence it is more secure.
C, FORTRAN, Pascal, Basic etc.	C++, Java, Python, C## etc.

OOPs Concepts:

1. Object

2. Class
3. Inheritance
4. Polymorphism
5. Abstraction
6. Encapsulation



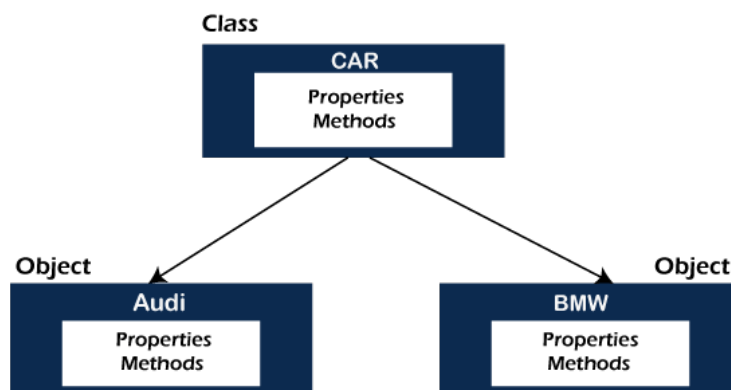
Object

An object is a real-world entity that has attributes, behavior, and properties. It is referred to as an instance of the class. It contains member functions, variables that we have defined in the class. It occupies space in the memory. Different objects have different states or attributes, and behaviors.

Class

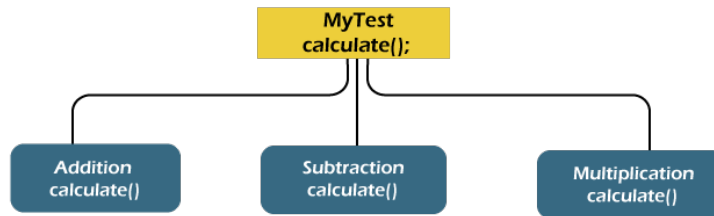
A class is a blueprint or template of an object. It is a user-defined data type. Inside a class, we define variables, constants, member functions, and other functionality. It binds data and functions together in a single unit. It does not consume memory at run time. Note that classes are not considered as a data structure. It is a logical entity. Note that a class can exist without an object but vice-versa is not possible.

The following figure best illustrates the class and object in OOP.



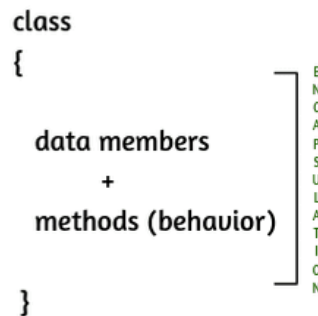
Abstraction

The concept allows us to hide the implementation from the user but shows only essential information to the user. Using the concept developer can easily make changes and added over time.



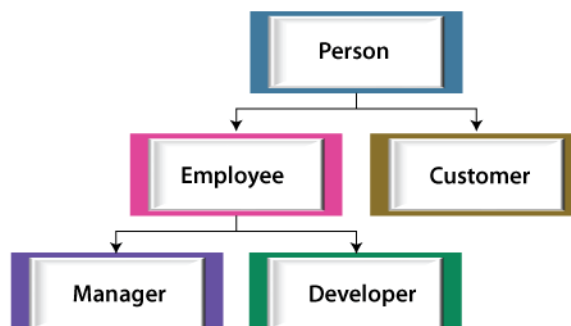
Encapsulation

Encapsulation is a mechanism that allows us to bind data and functions of a class into an entity. It protects data and functions from outside interference and misuse. Therefore, it also provides security. A class is the best example of encapsulation.



Inheritance

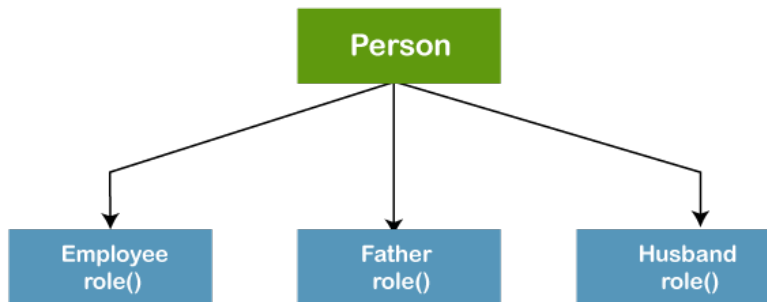
The concept allows us to inherit or acquire the properties of an existing class (parent class) into a newly created class (child class). It is known as **inheritance**. It provides code reusability.



Polymorphism

The word **polymorphism** is derived from the two words i.e. **poly** and **morphism**. Poly means many and morphism means forms. It allows us to create methods with the

same name but different method signatures. It allows the developer to create clean, sensible, readable, and resilient code.



A person plays an employee role in the office, father and husband role in the home.

Benefits of OOP

- Modular, scalable, extensible, reusable, and maintainable.
- It models the complex problem in a simple structure.
- Object can be used across the program.
- Code can be reused.
- We can easily modify, append code without affecting the other code blocs.
- Provides security through encapsulation and data hiding features.
- Beneficial to collaborative development in which a large project is divided into groups.
- Debugging is easy.

Limitations of OOP

- Requires intensive testing processes.
- The size of the programs created using this approach may become larger than the programs written using the procedure-oriented programming approach.
- Software developed using this approach requires a substantial amount of pre-work and planning.
- OOP code is difficult to understand if you do not have the corresponding class documentation.
- In certain scenarios, these programs can consume a large amount of memory.
- Not suitable for small problems.
- Takes more time to solve problems.

Python class:

- Classes are created by keyword class.
- Attributes are the variables that belong to a class.
- Attributes are always public and can be accessed using the dot (.) operator.
Eg.: Myclass.Myattribute

Create a Class

To create a class, use the keyword `class`:

```
class MyClass:  
    x = 5
```

Class Objects

An Object is an instance of a Class. A class is like a blueprint while an instance is a copy of the class with *actual values*.

Suppose a class is a prototype of a building. A building contains all the details about the floor, rooms, doors, windows, etc. we can make as many buildings as we want, based on these details. Hence, the building can be seen as a class, and we can create as many objects of this class.

Consider the following example to create a class **Employee** which contains two fields as Employee id, and name. The class also contains a function **display()**, which is used to display the information of the **Employee**.

```
class Employee:  
    id = 10  
    name = "Devansh"  
    def display (self):  
        print(self.id,self.name)
```

Here, the **self** is used as a reference variable, which refers to the current class object. It is always the first argument in the function definition. However, using **self** is optional in the function call.

Create Object

Now we can use the class named MyClass to create objects:

```
class MyClass:  
    x = 5  
p1 = MyClass()  
print(p1.x)  
output  
5
```

Declaring an object –

```
class Dog:  
    attr1 = "mammal"
```

```
attr2 = "dog"

def fun(self):
    print("I'm a", self.attr1)
    print("I'm a", self.attr2)

Rodger = Dog()

print(Rodger.attr1)

Rodger.fun()
```

Output: -

```
mammal
I'm a mammal
I'm a dog
```

Constructors:

A constructor is a special method in a class used to create and initialize an object of a class. A constructor is a unique function that gets called automatically when an object is created of a class. The main purpose of a constructor is to initialize or assign values to the data members of that class. **It cannot return any value other than none.**

Syntax of Python Constructor

```
def __init__(self):
    # initializations
```

init is one of the reserved functions in Python. In Object Oriented Programming, it is known as a constructor. Self is a reference to the current instance of the class. It is created and passed automatically/implicitly to the `__init__()` when the constructor is called.

Rules of Python Constructor

- It starts with the def keyword, like all other functions in Python.
- It is followed by the word init, which is prefixed and suffixed with double underscores with a pair of brackets, i.e., `__init__()`.
- It takes an argument called self, assigning values to the variables.

Types of Constructors

1. Parameterized Constructor
2. Non-Parameterized Constructor
3. Default Constructor

1. Parameterized Constructor in Python

When the constructor accepts arguments along with **self**, it is known as parameterized constructor. These arguments can be used inside the class to assign the values to the data members.

class Family:

```
# Constructor - parameterized
members=5
def __init__(self, count):
    print("This is parametrized constructor")
    self.members = count
def show(self):
    print("No. of members is", self.members)

object = Family(10)
object.show()
```

Output:

This is parameterized constructor

No. of members is 10

2. Non-Parameterized Constructor in Python

When the constructor doesn't accept any arguments from the object and has only one argument, **self**, in the constructor, it is known as a non-parameterized constructor. This can be used to re-assign a value inside the constructor.

```
class Fruits:
    favourite = "Apple"

    # non-parameterized constructor
    def __init__(self):
        self.favourite = "Orange"

    # a method
    def show(self):
        print(self.favourite)
```

```
# creating an object of the class
obj = Fruits()

# calling the instance method using the object obj
obj.show()

Output:
Orange
```

3. Default Constructor in Python

When you do not write the constructor in the class created, Python itself creates a constructor during the compilation of the program. It generates an empty constructor that has no code in it.

Example

```
class Assignments:
    check= "not done"
    # a method
    def is_done(self):
        print(self.check)

# creating an object of the class
obj = Assignments()

# calling the instance method using the object obj
obj.is_done()
```

Output

not done

More than One Constructor in Single class

```
class example:
    def __init__(self):
        print("One")
    def __init__(self):
        print("Two")
```

```
def __init__(self):  
    print("Three")  
  
e = example()
```

Output:

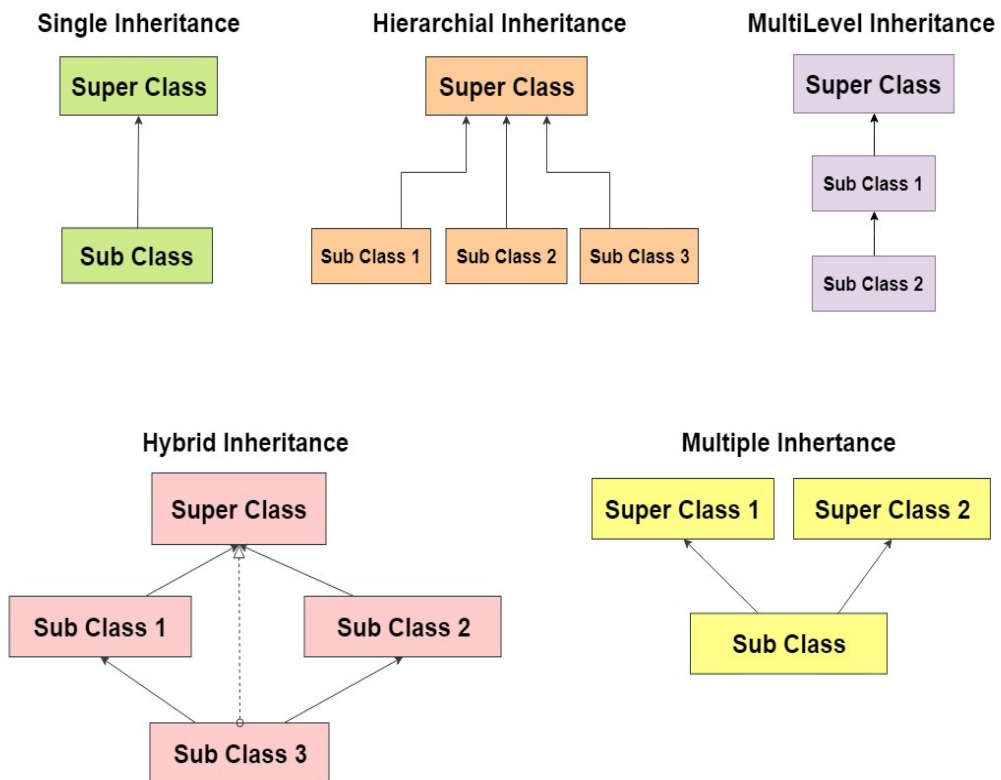
Three

Note: The constructor overloading is not allowed in Python.

Inheritance:

Inheritance provides code reusability to the program because we can use an existing class to create a new class instead of creating it from scratch. In inheritance, the child class acquires the properties and can access all the data members and functions defined in the parent class. A child class can also provide its specific implementation to the functions of the parent class.

Types of Inheritance: -



In python, a derived class can inherit base class by just mentioning the base in the bracket after the derived class name. Consider the following syntax to inherit a base class into the derived class.

Syntax

```
class derived-class(base class):
```



```
<class-suite>
```

A class can inherit multiple classes by mentioning all of them inside the bracket. Consider the following syntax.

Syntax

```
class derive-  
class(<base class 1>, <base class 2>, ..... <base class n>):  
    <class - suite>
```

Single Inheritance:-

```
class Animal:  
    def speak(self):  
        print("Animal Speaking")  
  
#child class Dog inherits the base class Animal  
class Dog(Animal):  
    def bark(self):  
        print("dog barking")  
  
d = Dog()  
d.bark()  
d.speak()
```

Output:

```
dog barking  
Animal Speaking
```

Multi-Level inheritance

```
class Animal:  
    def speak(self):  
        print("Animal Speaking")  
  
#The child class Dog inherits the base class Animal  
class Dog(Animal):  
    def bark(self):  
        print("dog barking")  
  
#The child class Dogchild inherits another child class Dog  
class DogChild(Dog):  
    def eat(self):  
        print("Eating bread...")
```

```
d = DogChild()
d.bark()
d.speak()
d.eat()
```

Output:

```
dog barking
Animal Speaking
Eating bread...
```

Multiple inheritance

```
# creating class for father
class Dad():
    # writing a method for parent class 1
    def singing(self):
        print("Dad sings well")

# creating a class for mother
class Mom():
    # method for parent class 2
    def coding(self):
        print("Mom codes well")

# creating derived class
class Child(Dad, Mom):
    def playing(self):
        print("Kid loves to play")

# creating object of the new derived class
child = Child()

# calling methods of parent classes and derived class
child.singing()
child.coding()
child.playing()
```

Output:

Dad sings well

Mom codes well

Kid loves to play

Hierarchical inheritance

```
# Base class
class Parent:
    def func1(self):
        print("This function is in parent class.")
# Derived class1
class Child1(Parent):
    def func2(self):
        print("This function is in child 1.")
# Derivied class2
class Child2(Parent):
    def func3(self):
        print("This function is in child 2.")
object1 = Child1()
object2 = Child2()
object1.func1()
object1.func2()
object2.func1()
object2.func3()
```

Output:-

This function is in parent class.

This function is in child 1.

This function is in parent class.

This function is in child 2.

Hybrid Inheritance :-

```
class School:
    def func1(self):
```

```
print("This function is in school.")  
  
class Student1(School):  
    def func2(self):  
        print("This function is in student 1. ")  
  
class Student2(School):  
    def func3(self):  
        print("This function is in student 2.")  
  
class Student3(Student1, School):  
    def func4(self):  
        print("This function is in student 3.")  
  
object = Student3()  
object.func1()  
object.func2()
```

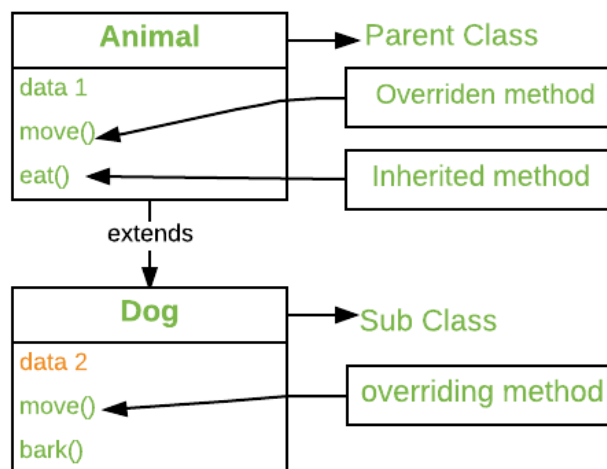
Output:

This function is in school.

This function is in student 1.

Method Overriding in Python

Method overriding is an ability of any object-oriented programming language that allows a subclass or child class to provide a specific implementation of a method that is already provided by one of its super-classes or parent classes. When a method in a subclass has the same name, same parameters or signature and same return type(or sub-type) as a method in its super-class, then the method in the subclass is said to **override** the method in the super-class.



The version of a method that is executed will be determined by the object that is used to invoke it. If an object of a parent class is used to invoke the method, then the version in the parent class will be executed, but if an object of the subclass is used to invoke the method, then the version in the child class will be executed.

Example:

```
class Parent():
    # Constructor
    def __init__(self):
        self.value = "Inside Parent"
    # Parent's show method
    def show(self):
        print(self.value)
# Defining child class
class Child(Parent):
    # Constructor
    def __init__(self):
        self.value = "Inside Child"
    # Child's show method
    def show(self):
        print(self.value)

obj1 = Parent()
obj2 = Child()
obj1.show()
obj2.show()
```

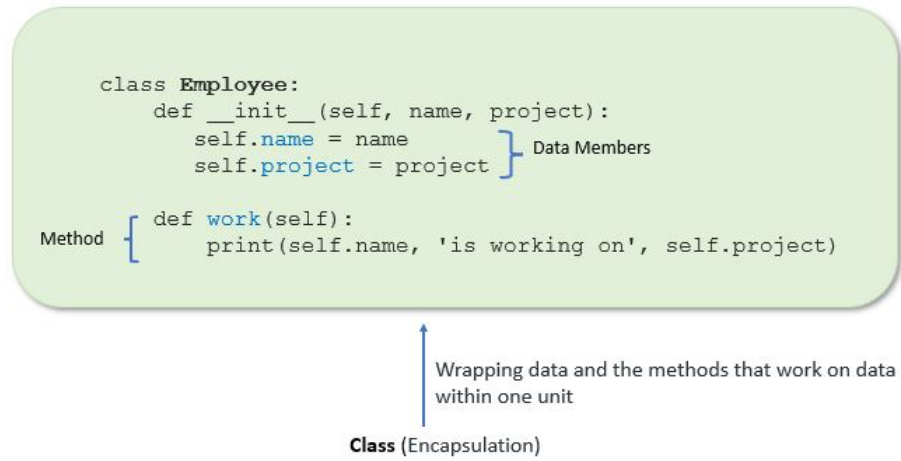
Output:

```
Inside Parent
Inside Child
```

Encapsulation

Encapsulation in Python describes the concept of bundling data and methods within a single unit. So, for example, when you create a class, it means you are implementing encapsulation. A class is an example of encapsulation as it binds all the data members (instance variables) and methods into a single unit.

Implement encapsulation using a class



```
class Employee:
    # constructor
    def __init__(self, name, salary, project):
        # data members
        self.name = name
        self.salary = salary
        self.project = project
    # method to display employee's details
    def show(self):
        # accessing public data member
        print("Name: ", self.name, 'Salary:', self.salary)
    # method
    def work(self):
        print(self.name, 'is working on', self.project)
# creating object of a class
emp = Employee('Jessa', 8000, 'NLP')
# calling public method of the class
emp.show()
emp.work()
```

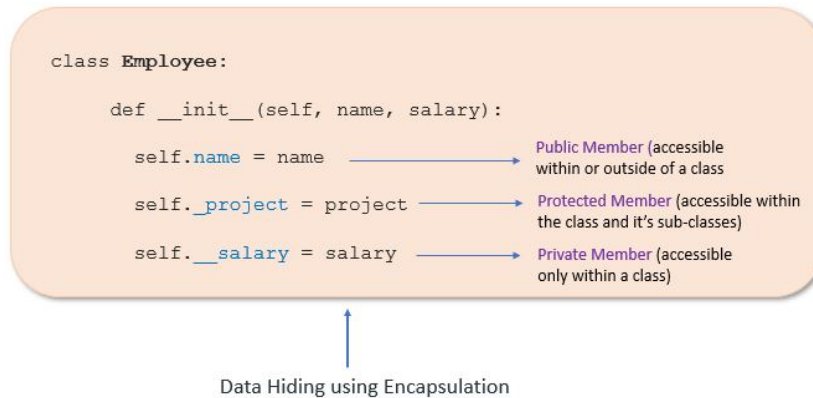
Output:

```
Name:  Jessa Salary: 8000
Jessa is working on NLP
```

Using encapsulation, we can hide an object's internal representation from the outside. This is called information hiding.

Access Modifiers in Python

- **Public Member:** Accessible anywhere from outside class.
- **Private Member:** Accessible within the class
- **Protected Member:** Accessible within the class and its sub-classes



Public Member

Public data members are accessible within and outside of a class. All member variables of the class are by default public.

```
class Employee:
    # constructor
    def __init__(self, name, salary):
        # public data members
        self.name = name
        self.salary = salary
    # public instance methods
    def show(self):
        # accessing public data member
        print("Name: ", self.name, 'Salary:', self.salary)
# creating object of a class
emp = Employee('Jessa', 10000)
# accessing public data members
print("Name: ", emp.name, 'Salary:', emp.salary)
# calling public method of the class
emp.show()
```

Output:-

```
Name:  Jessa Salary: 10000
```

```
Name:  Jessa Salary: 10000
```

Private Member

We can protect variables in the class by marking them private. To define a private variable, **add two underscores** as a prefix at the start of a variable name.

Private members are accessible only within the class, and we can't access them directly from the class objects.

Example:

```
class Employee:
    # constructor
    def __init__(self, name, salary):
        # public data member
        self.name = name
        # private member
        self.__salary = salary
# creating object of a class
emp = Employee('Jessa', 10000)
# accessing private data members
print('Salary:', emp.__salary)
```

Output:

```
AttributeError: 'Employee' object has no attribute '__salary'
```

We can access private members from outside of a class using the following two approaches:

- Create public method to access private members
- Use name mangling

Public method to access private members

Example: Access Private member outside of a class using an instance method

```
class Employee:
    # constructor
    def __init__(self, name, salary):
# public data member
```



```
self.name = name
# private member
    self.__salary = salary
# public instance methods
    def show(self):
        # private members are accessible from a class
    print("Name: ", self.name, 'Salary:', self.__salary)
# creating object of a class
emp = Employee('Jessa', 10000)
# calling public method of the class
emp.show()
```

Output:

Name: Jessa Salary: 10000

Name Mangling to access private members

We can directly access private and protected variables from outside of a class through name mangling. The name mangling is created on an identifier by adding two leading underscores and one trailing underscore, like this `_classname__dataMember`, where `classname` is the current class, and `data member` is the private variable name.

Access private member

```
class Employee:
    # constructor
    def __init__(self, name, salary):
        # public data member
        self.name = name
        # private member
        self.__salary = salary
# creating object of a class
emp = Employee('Jessa', 10000)
print('Name:', emp.name)
# direct access to private member using name mangling
print('Salary:', emp._Employee__salary)
```

Output

Name: Jessa

Salary: 10000

Protected Member

Protected members are accessible within the class and also available to its sub-classes. To define a protected member, prefix the member name with a **single underscore** `_`. Protected data members are used when you implement inheritance and want to allow data members access to only child classes.

```
# base class
class Company:
    def __init__(self):
        # Protected member
        self._project = "NLP"

# child class
class Employee(Company):
    def __init__(self, name):
        self.name = name
        Company.__init__(self)

    def show(self):
        print("Employee name :", self.name)
        # Accessing protected member in child class
        print("Working on project :", self._project)

c = Employee("Jessa")
c.show()

# Direct access protected data member
print('Project:', c._project)
```

Output

Employee name : Jessa

Working on project : NLP

Project: NLP

Polymorphism

Polymorphism is made from 2 words – ‘**poly**’ and ‘**morphs**.’ The word ‘poly’ means ‘many’ and ‘morphs’ means ‘many forms.’ Polymorphism means having multiple forms.

Polymorphism may be used in one of the following ways in an object-oriented language:

- Overloading of operators
- Class Polymorphism in Python
- Method overriding, also referred to as Run time Polymorphism
- Method overloading, also known as Compile time Polymorphism

Polymorphism is supported in Python via method overriding and operator overloading. However, Python does not support method overloading.

Polymorphism in Python through Operator Overloading

Operator overloading is another type of polymorphism in which the same operator performs various operations depending on the operands. Python allows for operator overloading.

1. Polymorphism in + operator:

We already know that the ‘+’ operator is frequently used in Python programs. The + operator behaves differently depending on the type of object on which it is used.

```
a = 10
b = 20

print('Addition of 2 numbers:', a + b)

str1 = 'Hello '
str2 = 'Python'

print('Concatenation of 2 strings:', str1 + str2)

list1 = [1, 2, 3]
list2 = ['A', 'B']

print('Concatenation of 2 lists:', list1 + list2)
```

Output

```
Addition of 2 numbers: 30
Concatenation of 2 strings: Hello Python
Concatenation of 2 lists: [1, 2, 3, 'A', 'B']
```

2. Polymorphism in * operator:

The * operator is used to multiply 2 numbers if the data elements are numeric values. If one of the data types is a string, and the other is numeric, the string is printed that many times as that of the 2nd variable in the multiplication process.

```
a = 10
b = 5
print('Multiplication of 2 numbers:', a * b)
num = 3
mystr = 'Python'
print('Multiplication of string:', num * mystr)
```

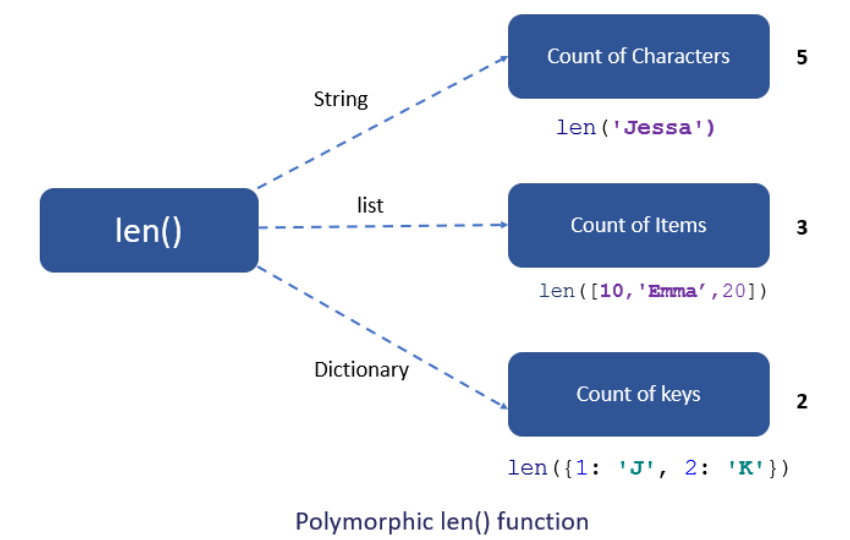
Output

Multiplication of 2 numbers: 50

Multiplication of string: PythonPythonPython

Function Polymorphism in Python

There are certain Python functions that can be used with different data types. The len() function is one example of such a function. Python allows it to work with a wide range of data types. The built-in function len() estimates an object's length based on its type. If an object is a string, it returns the number of characters; or if an object is a list, it returns the number of elements in the list. If the object is a dictionary, it gives the total number of keys found in the dictionary.



```
mystr = 'Programming'
print('Length of string:', len(mystr))
mylist = [1, 2, 3, 4, 5]
```

```
print('Length of list:', len(mylist))
mydict = {1: 'One', 2: 'Two'}
print('Length of dict:', len(mydict))
```

Output

```
Length of string: 11
Length of list: 5
Length of dict: 2
```

Class Polymorphism in Python

Because Python allows various classes to have methods with the same name, we can leverage the concept of polymorphism when constructing class methods. We may then generalize calling these methods by not caring about the object we're working with. Then we can write a for loop that iterates through a tuple of items.

```
class Tiger():
    def nature(self):
        print('I am a Tiger and I am dangerous.')
    def color(self):
        print('Tigers are orange with black strips')
class Elephant():
    def nature(self):
        print('I am an Elephant and I am calm and harmless')
    def color(self):
        print('Elephants are grayish black')
obj1 = Tiger()
obj2 = Elephant()
for animal in (obj1, obj2): # creating a loop to iterate through the obj1
and obj2
    animal.nature()
    animal.color()
```

Output

```
I am a Tiger and I am dangerous.
Tigers are orange with black strips
I am an Elephant and I am calm and harmless
Elephants are grayish black
```

Polymorphism and Inheritance (Method Overriding)

In Python, child classes, like other programming languages, inherit methods and attributes from the parent class. Method Overriding is the process of redefining certain methods and attributes to fit the child class. This is especially handy when the method inherited from the parent class does not exactly fit the child class. In such circumstances, the method is re-implemented in the child class. Method Overriding refers to the technique of re-implementing a method in a child class.

```
class Vehicle:
    def __init__(self, brand, model, price):
        self.brand = brand
        self.model = model
        self.price = price
    def show(self):
        print('Details:', self.brand, self.model, 'Price:', self.price)
    def max_speed(self):
        print('Vehicle max speed is 160')
    def gear_system(self):
        print('Vehicle has 6 shifter gearbox')
# inherit from vehicle class
class Car(Vehicle):
    def max_speed(self):
        print('Car max speed is 260')
    def gear_system(self):
        print('Car has Automatic Transmission')
# Car Object
car = Car('Audi', 'R8', 9000000)
car.show()
# call methods from Car class
car.max_speed()
car.gear_system()
# Vehicle Object
vehicle = Vehicle('Nissan', 'Magnite', 550000)
```

```
vehicle.show()  
  
# call method from a Vehicle class  
  
vehicle.max_speed()  
  
vehicle.gear_system()
```

Output

```
Details: Audi R8 Price: 9000000  
  
Car max speed is 260  
  
Car has Automatic Transmission  
  
Details: Nissan Magnite Price: 550000  
  
Vehicle max speed is 160  
  
Vehicle has 6 shifter gearbox
```

Compile-Time Polymorphism (Method Overloading)

Method overloading occurs when a class contains many methods with the same name. The types and amount of arguments passed by these overloaded methods vary. Python does not support method overloading or compile-time polymorphism. If there are multiple methods with the same name in a class or Python script, the method specified in the latter one will override the earlier one.

Exception Handling

Error in Python can be of two types i.e. Syntax errors and Exceptions. Errors are the problems in a program due to which the program will stop the execution. On the other hand, exceptions are raised when some internal events occur which changes the normal flow of the program.

Difference between Syntax Error and Exceptions

Syntax Error: As the name suggests this error is caused by the wrong syntax in the code. It leads to the termination of the program.

```
amount = 10000  
  
if(amount > 2999)  
  
print("You are eligible to purchase Dsa Self Paced")
```

Exceptions: Exceptions are raised when the program is syntactically correct, but the code resulted in an error. This error does not stop the execution of the program, however, it changes the normal flow of the program.

```
divide_by_zero = 7 / 0
```

Python Logical Errors (Exceptions)

Errors that occur at runtime (after passing the syntax test) are called **exceptions** or **logical errors**.

For instance, they occur when we

- try to open a file(for reading) that does not exist (FileNotFoundError)
- try to divide a number by zero (ZeroDivisionError)
- try to import a module that does not exist (ImportError) and so on.

Whenever these types of runtime errors occur, Python creates an exception object. If not handled properly, it prints a traceback to that error along with some details about why that error occurred.

Python Built-in Exceptions

Illegal operations can raise exceptions. There are plenty of built-in exceptions in Python that are raised when corresponding errors occur.

We can view all the built-in exceptions using the built-in `local()` function as follows:

```
print(dir(locals()['__builtins__']))
```

Python Exception Handling

Python try...except Block: The try...except block is used to handle exceptions in Python. Here's the syntax of try...except block:

```
try:  
    # code that may cause exception  
except:  
    # code to run when exception occurs
```

Here, we have placed the code that might generate an exception inside the try block. Every try block is followed by an except block.

When an exception occurs, it is caught by the except block. The except block cannot be used without the try block.

```
try:  
    numerator = 10  
    denominator = 0  
    result = numerator/denominator  
    print(result)  
except:  
    print("Error: Denominator cannot be 0.")
```


Output:

Error: Denominator cannot be 0.

Catching Specific Exceptions

For each try block, there can be zero or more except blocks. Multiple except blocks allow us to handle each exception differently. The argument type of each except block indicates the type of exception that can be handled by it. For example,

```
try:
    even_numbers = [2,4,6,8]
    print(even_numbers[5])
except ZeroDivisionError:
    print("Denominator cannot be 0.")
except IndexError:
    print("Index Out of Bound.")
```

Output:

Index Out of Bound

Python try with else clause: In some situations, we might want to run a certain block of code if the code block inside try runs without any errors. For these cases, you can use the optional else keyword with the try statement.

```
# program to print the reciprocal of even numbers
```

```
try:
    num = int(input("Enter a number: "))
    assert num % 2 == 0
except:
    print("Not an even number!")
else:
    reciprocal = 1/num
    print(reciprocal)
```

Output:

Enter a number: 2

0.5

Python try...finally

In Python, the finally block is always executed no matter whether there is an exception or not. The finally block is optional. And, for each try block, there can be only one finally block.

```
try:
    numerator = 10
    denominator = 0
    result = numerator/denominator
    print(result)
except:
    print("Error: Denominator cannot be 0.")
    finally:
        print("This is finally block.")
```

Output:

```
Error: Denominator cannot be 0.
This is finally block.
```

Python Custom Exceptions

Defining Custom Exceptions

In Python, we can define custom exceptions by creating a new class that is derived from the built-in Exception class.

The syntax to define custom exceptions,

```
class CustomError(Exception):    ...
    pass
try:
    ...
except CustomError:
    ...
```

Here, CustomError is a user-defined error which inherits from the Exception class.

Note: When we are developing a large Python program, it is a good practice to place all the user-defined exceptions that our program raises in a separate file.

- Many standard modules define their exceptions separately as exceptions.py or errors.py (generally but not always).

Python User-Defined Exception

```
class InvalidAgeException(Exception):
```

```
"Raised when the input value is less than 18"  
pass  
# you need to guess this number  
number = 18  
try:  
    input_num = int(input("Enter a number: "))  
    if input_num < number:  
        raise InvalidAgeException  
    else:  
        print("Eligible to Vote")  
    except InvalidAgeException:  
        print("Exception occurred: Invalid Age")
```

Output

If the user input input_num is greater than 18,

Enter a number: 45

Eligible to Vote

If the user input input_num is smaller than 18,

Enter a number: 14

Exception occurred: Invalid Age

Data Handling and Visualization using NumPy, Pandas, Matplotlib and Seaborn

Sanchita Naha

ICAR-Indian Agricultural Statistics Research Institute, New Delhi - 110 012
sanchita.naha@icar.gov.in

Introduction:

Python offers many software packages for smooth data handling required for deploying Machine Learning or Deep Learning projects. Most important and very frequently used packages of them are NumPy and Pandas for data handling in array or tabular format and Matplotlib for data visualization.

Let us start with the NumPy library first. NumPy stands for Numerical Python. It is a library consisting of functions to handle multidimensional array objects and a collection of routines for processing arrays. NumPy was created in 2005 by Travis Oliphant. It is an open-source project, it can be used freely. NumPy array objects are 50x faster than traditional Python lists. Using NumPy, mathematical and logical operations on arrays can be performed. This tutorial explains the code for declaration and manipulation of multidimensional arrays and its contents using NumPy. To install NumPy in local system use the following code in Python editor.

```
pip install numpy as np
```

After installation, run the following code:

```
import numpy as np
print(np.__version__)
```

Declare a one-dimensional array with the following code:

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
print(arr)
print(type(arr))
```

Datatype of an array:

```
print(arr.dtype)
```

Create arrays of float and string type:

```
arr_float = np.array([10.2, 23.0, 68.5, 98.7, 5.0])
print(arr_float)
print(arr_float.dtype)
```

```
arr_string = np.array(['ramayanas','b','c','d','e'])
print(arr_string)
arr_string.dtype
```

Declare array of zeroes and ones:

```
arr = np.zeros(5)
arr = np.zeros([2,3]) arr
arr = np.ones(5) arr
```

Dimensions in Array:

0-D Arrays: 0-D arrays, or Scalars, are the elements in an array

```
import numpy as np
arr = np.array(42)
print(arr)
```

1-D Array: An array that has 0-D arrays as its elements is called uni-dimensional or 1-D array. These are the most common and basic arrays.

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
print(arr)
```

2-D Arrays: An array that has 1-D arrays as its elements is called a 2-D array. These are often used to represent matrix or 2nd order tensors.

```
arr = np.array([[1, 2, 3], [4, 5, 6]])
print(arr)
```

3-D arrays: An array that has 2-D arrays (matrices) as its elements is called 3-D array. These are often used to represent a 3rd order tensor.

```
arr = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])
print(arr)
```

NumPy Arrays provides the `ndim` attribute that returns an integer that tells us how many dimensions the array have.

```
arr_0D = np.array(42)
arr_1D = np.array([1, 2, 3, 4, 5])
arr_2D = np.array([[1, 2, 3], [4, 5, 6]])
arr_3D = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])
print('Dimension of arr_0D is: ',arr_0D.ndim)
print('Dimension of arr_0D is: ',arr_1D.ndim)
print('Dimension of arr_0D is: ',arr_2D.ndim)
print('Dimension of arr_0D is: ',arr_3D.ndim)
```

Higher Dimensional Arrays An array can have any number of dimensions. When the array is created, you can define the number of dimensions by using the `ndmin` argument.

```
arr = np.array([1, 2, 3, 4], ndmin=5)
print(arr)
print('number of dimensions :', arr.ndim)
```

Indexing of an array:

Access Array Elements: Array indexing is the same as accessing an array element. You can access an array element by referring to its index number. The indexes in NumPy arrays start with 0, meaning that the first element has index 0, and the second has index 1 etc.

```
arr = np.array([1, 2, 3, 4])
print(arr[0])
```

Get third and fourth elements from the following array and add them.

```
arr = np.array([1, 2, 3, 4])
print(arr[2] + arr[3])
```

Access 2-D Arrays: To access elements from 2-D arrays we can use comma separated integers representing the dimension and the index of the element. Think of 2-D arrays like a table with rows and columns, where the dimension represents the row and the index represents the column.

Access the element on the first row, second column:

```
arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])
print('2nd element on 1st row: ', arr[0, 1])
```

#Access the element on the 2nd row, 5th column:

```
print('5th element on 2nd row: ', arr[1, 4])
```

Access 3-D Arrays: To access elements from 3-D arrays we can use comma separated integers representing the dimensions and the index of the element.

```
arr_3d = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])
print(arr_3d.ndim)
print(arr_3d[0, 1, 2])
# Printing size (total number of elements) of array
print("Size of array: ", arr.size)
```

The first number represents the first dimension, which contains two arrays: `[[1, 2, 3], [4, 5, 6]]` and: `[[7, 8, 9], [10, 11, 12]]` Since we selected 0, we are left with the first array: `[[1, 2, 3], [4, 5, 6]]`

The second number represents the second dimension, which also contains two arrays: `[1, 2, 3]` and: `[4, 5, 6]` Since we selected 1, we are left with the second array: `[4, 5, 6]`

The third number represents the third dimension, which contains three values: `4 5 6` Since we selected 2, we end up with the third value: `6`

Negative Indexing Use negative indexing to access an array from the end.

```
arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])
print('Last element from 2nd dim: ', arr[1, -1])
```

NumPy Array Slicing: Slicing in python means taking elements from one given index to another given index. We pass slice instead of index like this: `[start:end]`. We can also define the step, like this: `[start:end:step]`. If we don't pass start its considered 0 If we don't pass end its considered length of array in that dimension. If we don't pass step its considered 1 Note: The result includes the start index, but excludes the end index.

```
arr = np.array([1, 2, 3, 4, 5, 6, 7])
print(arr[1:5])
```

Negative Slicing: Use the minus operator to refer to an index from the end.

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7])
print(arr[-3:-1])
```

STEP Use the step value to determine the step of the slicing:

```
arr = np.array([1, 2, 3, 4, 5, 6, 7])
print(arr[1:5:2])
#Return every other element from the entire array:
arr = np.array([1, 2, 3, 4, 5, 6, 7])
print(arr[::2])
```

Slicing 2-D Arrays From the second element, slice elements from index 1 to index 4 (not included):

```
arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])
print(arr[1, 1:4])
```

Checking the Data Type of an Array The NumPy array object has a property called `dtype` that returns the data type of the array:

```
arr = np.array([1, 2, 3, 4])
print(arr.dtype)
arr1 = np.array(['apple', 'banana', 'cherry'])
print(arr1.dtype)
```

Creating Arrays With a Defined Data Type We use the `array()` function to create arrays, this function can take an optional argument: `dtype` that allows us to define the expected data type of the array elements:

```
arr = np.array([1, 2, 3, 4], dtype='S')
print(arr)
print(arr.dtype)
arr = np.array([1, 2, 3, 4], dtype='i4')
print(arr)
print(arr.dtype)
```

Converting Data Type on Existing Arrays The best way to change the data type of an existing array, is to make a copy of the array with the `astype()` method. The `astype()` function creates a copy of the array, and allows you to specify the data type as a parameter. The data type can be specified using a string, like 'f' for float, 'i' for integer etc. or you can use the data type directly like float for float and int for integer.

```
arr = np.array([1.1, 2.1, 3.1])
newarr = arr.astype('i')
print(newarr)
print(newarr.dtype)
arr = np.array([1, 0, 3])
newarr = arr.astype(bool)
print(newarr)
print(newarr.dtype)
```

Shape of an Array The shape of an array is the number of elements in each dimension. Get the Shape of an Array NumPy arrays have an attribute called `shape` that returns a tuple with each index having the number of corresponding elements.

```
arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])
print(arr.shape)
```


#Create an array with 5 dimensions using ndmin using a vector with values 1,2,3,4 and verify that last dimension has value 4:

```
arr = np.array([1, 2, 3, 4], ndmin=5)
print(arr)
print('shape of array :', arr.shape)
```

Reshaping arrays Reshaping means changing the shape of an array. The shape of an array is the number of elements in each dimension. By reshaping we can add or remove dimensions or change number of elements in each dimension.

```
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])
print(arr.shape)
newarr = arr.reshape(4, 3)
print(newarr)
```

Reshape From 1-D to 3-D:

```
#Convert the following 1-D array with 12 elements into a 3-D array.
#The outermost dimension will have 2 arrays that contains 3 arrays, each with 2 elements:
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])
print(arr)
newarr = arr.reshape(2, 3, 2)
print(newarr)
newarr1 = arr.reshape(-1, 1, 2)
print(newarr1)
```

Can We Reshape Into any Shape? Yes, if the elements required for reshaping are equal in both shapes. We can reshape an 8 elements 1D array into 4 elements in 2 rows 2D array but we cannot reshape it into a 3-elements 3 rows 2D array as that would require $3 \times 3 = 9$ elements. Note: We cannot pass -1 to more than one dimension.

```
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8])
newarr = arr.reshape(2, 2, -1)
print(newarr)
```

Note: We can not pass -1 to more than one dimension.

```
arr = np.array([[1, 2, 3], [4, 5, 6]])
newarr = arr.reshape(-1)
print(newarr)
```

Array Iteration:

```
arr = np.array([[1, 2, 3], [4, 5, 6]])
for x in arr:
    for y in x:
        print(y)
```

Iterating on Each Scalar Element In basic for loops, iterating through each scalar of an array we need to use n for loops which can be difficult to write for arrays with very high dimensionality.

```
arr = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])
for x in np.nditer(arr):
    print(x)
```

Searching Arrays You can search an array for a certain value, and return the indexes that get a match. To search an array, use the where() method.

```
arr = np.array([1, 2, 3, 4, 5, 4, 4])
x = np.where(arr == 4)
print(x)
```

Addition, Subtraction, Division of elements of Matrix:

```
import numpy

# initializing matrices
x = numpy.array([[1, 2], [4, 5]])
y = numpy.array([[7, 8], [9, 10]])

print(x)
print(y)

# using add() to add matrices
```

```
print ("The element wise addition of matrix is : ")
print (numpy.add(x,y))

# using subtract() to subtract matrices
print ("The element wise subtraction of matrix is : ")
print (numpy.subtract(x,y))

# using divide() to divide matrices
print ("The element wise division of matrix is : ")
print (numpy.divide(x,y))
```

Array Multiplication:

```
import numpy

# initializing matrices
x = numpy.array([[1, 2], [4, 5]])
y = numpy.array([[7, 8], [9, 10]])

# using multiply() to multiply matrices element wise
print ("The element wise multiplication of matrix is : ")
print (numpy.multiply(x,y))

# using dot() to multiply matrices
print ("The product of matrices is : ")
print (numpy.dot(x,y))
```

Matrix transpose:

```
print ("The transpose of given matrix is : ")
print (x.T)
```

Matrix Multiplication:

```
# creating two matrices
p = [[1, 2], [2, 3]]
q = [[4, 5], [6, 7]]
print("Matrix p :")
print(p)
print("Matrix q :")
print(q)

# computing product
result = np.dot(p, q)

# printing the result
print("The matrix multiplication is :")
print(result)
```

The `numpy.linspace()` function returns number spaces evenly at a specified interval. Similar to `numpy.arange()` function but instead of step it uses sample number

```
# np.linspace(start, stop, num=50, endpoint=True, retstep=False,
dtype=None, axis=0)

#start = starting value; stop = end value; endpoint = true means
include the last sample

# retstep = true ; stepping between samples a =
np.linspace(1,10,10,retstep=True)

a = np.linspace(1,10,10,endpoint = False,retstep=True) a

#np.arange([start, ]stop, [step, ], dtype=None)

# starting number, ending position (excluding this value), step =
spacing; by default it is set to 1

np.arange(1,100,9)
```

Generation of random numbers:

```
### Random Number

## random.randint(low, high=None, size=None, dtype=int)

## Return random integers from low (inclusive) to high (exclusive)
np.random.seed(10)
```

```
print(np.random.randint(100)) print(np.random.randint(100,1000,6))
print(np.random.randint(10,30,[5,4]))
```

Pandas is an open-source high-performance, easy-to-use Python library for data analysis. In this tutorial, working with Data Frame object has been illustrated. Install pandas as the following:

```
pip install pandas
```

For demonstration of pandas we have used two freely available dataset e.g., Iris.csv and titanic.csv.

```
#load Iris dataset

import pandas as pd

df = pd.read_csv('Iris.csv')

print(df.head())
```

```
#load titanic dataset

import pandas as pd

data = pd.read_csv('titanic.csv')

print(data.head())
```

Pandas is a Python library used for working with data sets. It has functions for analyzing, cleaning, exploring, and manipulating data. The name "Pandas" has a reference to both "Panel Data", and "Python Data Analysis", was created by Wes McKinney in 2008. Pandas allows us to analyze big data and make conclusions based on statistical theories. Pandas can clean messy data sets and make them readable and relevant which is the most important requirement in data science. Pandas gives you answers about the data. Like: Is there a correlation between two or more columns? What is average value? Max value? Min value? Using Pandas, it is possible to delete rows that are not relevant, or contains wrong values, like empty or NULL values. This is called cleaning the data.

Create a dataframe with dictionary data structure:

```
import pandas as pd

mydataset = {
    'cars': ["BMW", "Volvo", "Ford"],
    'passings': [3, 7, 2]
}
```

```
myvar = pd.DataFrame(mydataset)
print(myvar)
```

Pandas as pd Pandas is usually imported under the pd alias. alias: In Python alias are an alternate name for referring to the same thing.

```
print(pd.__version__)
```

Pandas Series What is a Series? A Pandas Series is like a column in a table. It is a one-dimensional array holding data of any type.

```
#printing dataframe elements
```

```
a = [1, 7, 2]
myvar = pd.Series(a)
print(myvar)
print(myvar[0])
```

```
#printing dataframe elements
```

```
a = [1, 7, 2]
myvar = pd.Series(a, index = ["x", "y", "z"])
print(myvar)
print(myvar["y"])
```

```
#Note: The keys of the dictionary become the labels.
```

```
calories = {"day1": 420, "day2": 380, "day3": 390}
myvar = pd.Series(calories)
print(myvar)
```

```
calories = {"day1": 420, "day2": 380, "day3": 390}
myvar = pd.Series(calories, index = ["day1", "day2"])
print(myvar)
```

DataFrames Data sets in Pandas are usually multi-dimensional tables, called DataFrames. Series is like a column, a DataFrame is the whole table.

```
data = {  
    "calories": [420, 380, 390],  
    "duration": [50, 40, 45]  
}  
myvar = pd.DataFrame(data)  
print(myvar)
```

A Pandas DataFrame is a 2-dimensional data structure, like a 2-dimensional array, or a table with rows and columns. The DataFrame is like a table with rows and columns. Pandas use the *loc* attribute to return one or more specified row(s)

Locate Row:

```
data = {  
    "calories": [420, 380, 390],  
    "duration": [50, 40, 45]  
}  
#load data into a DataFrame object:  
df = pd.DataFrame(data)  
print(df)  
print(df.loc[0])
```

#Return row 0 and 1:

#use a list of indexes:

```
print(df.loc[[0, 1]])
```

Named Indexes: With the index argument, you can name your own indexes.

```
data = {  
    "calories": [420, 380, 390],  
    "duration": [50, 40, 45]  
}  
df = pd.DataFrame(data, index = ["day1", "day2", "day3"])  
print(df)
```

Max_rows: The number of rows returned is defined in Pandas option settings. You can check your system's maximum rows with the `pd.options.display.max_rows` statement.

```
import pandas as pd  
print(pd.options.display.max_rows)
```

Generally in local systems the number is set to be 60, which means that if the DataFrame contains more than 60 rows, the `print(df)` statement will return only 60 rows. We can change the maximum number of rows number with the same statement.

```
pd.options.display.max_rows = 9999  
df = pd.read_csv('Iris.csv')  
print(df)  
print (df.head())
```

Data Viewing: One of the most used method for getting a quick overview of the DataFrame, is the `head()` method. The `head()` method returns the headers and a specified number of rows, starting from the top.

```
#print top 7 rows of the data frame  
df = pd.read_csv('Iris.csv')  
print(df.head(7))
```

There is also a `tail()` method for viewing the last rows of the DataFrame. The `tail()` method returns the headers and a specified number of rows, starting from the bottom.

```
# print last 5 rows  
print(df.tail())
```

Info About the Data: The DataFrame object has a method called `info()`, that gives you more information about the data set.

```
import pandas as pd  
df = pd.read_csv('Iris.csv')  
print(df.info())
```

Null Values: The `info()` method shows how many Non-Null values are present in each column, and in the data set. Empty values, or Null values, can be bad when analyzing data, and such rows should be removed with empty values. This is called data cleaning. Data Cleaning means fixing bad data in the data set. Bad data could be Empty cells, Data in wrong format, Wrong data, Duplicates. Empty cells can potentially give wrong result when you analyze data.

Removing Rows: One way to deal with empty cells is to remove rows that contain empty cells.

```
import pandas as pd
df = pd.read_csv('Iris.csv')
new_df = df.dropna()
print(new_df.head())

# Note: By default, the dropna() method returns a new DataFrame, and will not change the original.

# If you want to change the original DataFrame, use the inplace = True argument:
df = pd.read_csv('data.csv')
df.dropna(inplace = True)
print(df)
```

Replace Empty Values: Another way of dealing with empty cells is to insert a new value instead. This way no need to delete entire rows just because of some empty cells. The fillna() method allows to replace empty cells with a value:

```
df = pd.read_csv('data.csv')
df.fillna(130, inplace = True)
```

Replace Only for Specified Columns: The example above replaces all empty cells in the whole Data Frame. To only replace empty values for one column, specify the column name for the DataFrame.

```
df = pd.read_csv('data.csv')
df["Calories"].fillna(130, inplace = True)
```

Matplotlib: Matplotlib is one of the most popular Python packages used for data visualization. It is a cross-platform library for making 2D plots from data in arrays. It provides an object-oriented API that helps in embedding plots in applications using Python GUI toolkits such as PyQt, WxPython or Tkinter. It can be used in Python and IPython shells, Jupyter notebook and web application servers also.

Sample Codes:

```
import matplotlib.pyplot as plt
### Line Plot
x = [1,2,3,4,5,6,7,8,9,10]
y = [1,4,9,16,25,36,49,64,81,100]
plt.figure(figsize=(8,4))
```

```
plt.title("Line Plot  
Graph",fontsize=15,color='red',fontweight='bold') plt.xlabel("X  
Axis",fontsize=12,color='blue',fontweight='bold') plt.ylabel("Y  
Axis",fontsize=12,color='blue',fontweight='bold')  
## labels=[0,1,2,3,4,5,6,7,8,9,10]  
##plt.xticks(labels,fontsize=15,color='green')  
plt.xticks(fontsize=15,color='green')  
plt.yticks(fontsize=15,color='green')  
#plt.plot(x,y,'o-; o--;. ; --; --*; v; ^; o; -s; -*',label="Line  
Plot",color="purple",lw=1)  
plt.plot(x,y,'-o',label="Line Plot",color="purple",lw=1)  
plt.legend(loc=2,fontsize=12)  
plt.grid()  
plt.show()
```

Output:

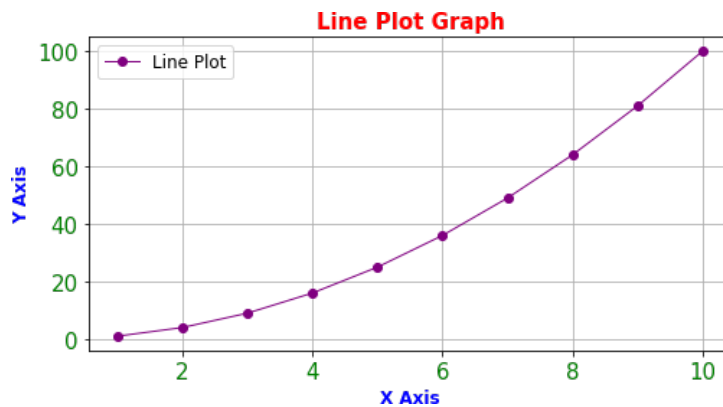


Figure 1: Line plot graph

```
# Bar Plot  
x = ["A", "B", "C", "D", "E"] y = [10,20,40,30,50]  
plt.figure(figsize=(8,4))  
plt.title("Bar Plot Graph",fontsize=15,color='brown',  
fontweight='bold') plt.xlabel("X Axis",fontsize=12,color='blue')  
plt.ylabel("Y Axis",fontsize=12,color='blue')  
plt.xticks(fontsize=15,color=orange)  
plt.yticks(fontsize=15,color='green')  
plt.bar(x,y,label="Bar Plot",color=["orange","green"],width=0.5)  
plt.legend(loc=2,fontsize=12)  
plt.show()
```

Output:

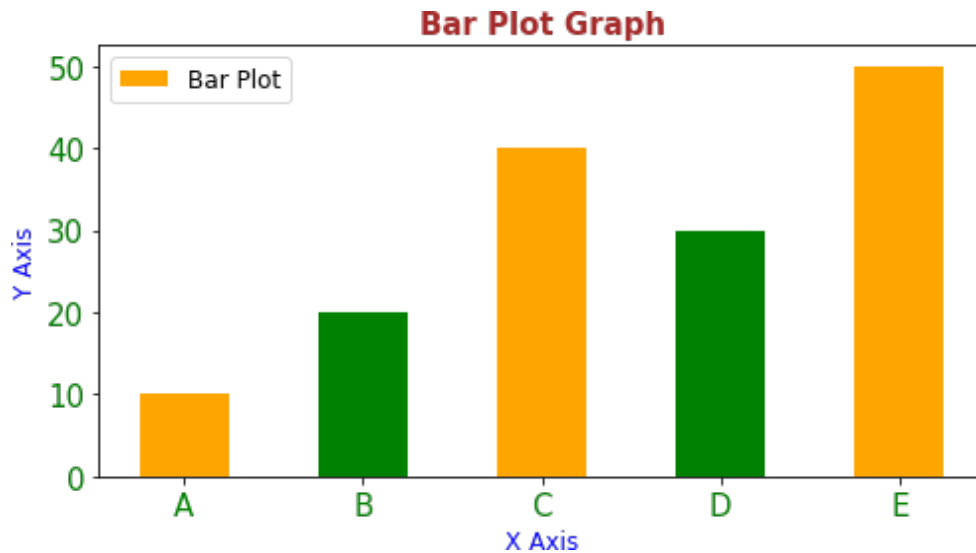


Figure 2: Bar plot graph

Scatter Plot

```
x = [5,7,8,7,2,17,2,9,4,11,12,9,6]
y = [99,86,87,88,111,86,103,87,94,78,77,85,86]
plt.figure(figsize=(6,4))
plt.title("Scatter Plot Graph",fontsize=15,color='red')
plt.xlabel("X Axis",fontsize=12,color='blue') plt.ylabel("Y Axis",fontsize=12,color='blue')
plt.xticks(fontsize=15,color='green')
plt.yticks(fontsize=15,color='green')
plt.scatter(x,y,label="Scatter Plot",color="purple",s=40,marker = "o") plt.legend(loc=2,fontsize=12)
plt.show()
```

Output:

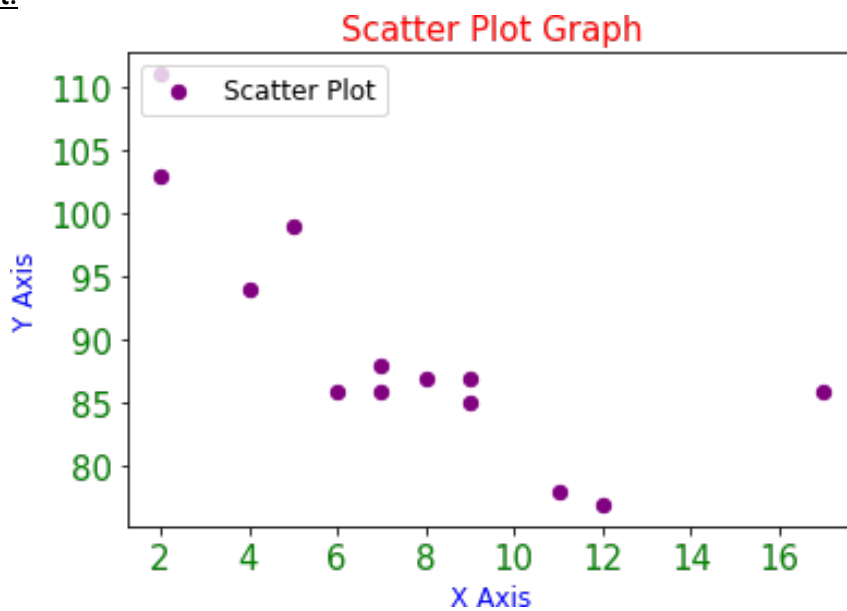


Figure 3: Scatter plot graph

Histogram

```
import numpy as np
np.random.seed(10)
data = np.random.randint(1,100,50) print(data)
plt.hist(data,rwidth=0.5,bins=5,color="pink") plt.show()
```

Output:

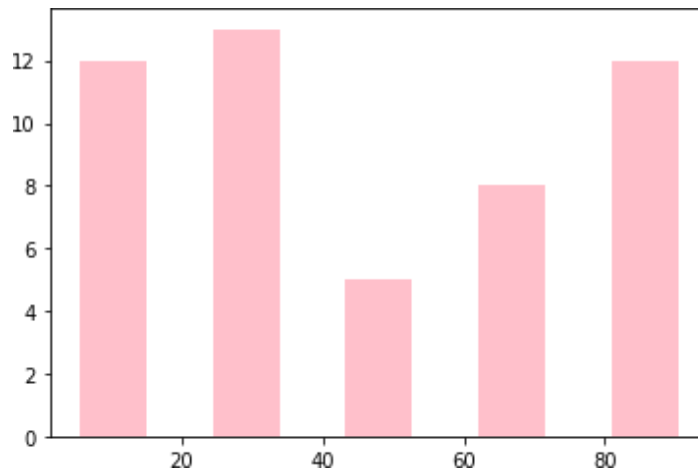


Figure 4: Histogram of a randomly generated set of numbers

simple pie chart

```
import numpy as np
import matplotlib.pyplot as plt
labels=['playing','sleeping','reading','eating'] sizes =
[25,25,25,25]
colors=['red','green','yellow','blue']
plt.pie(sizes, labels=labels, colors=colors,
autopct="%.2f%",) plt.axis('equal')
plt.show()
```

Output:



Figure 5: Pie chart

Pie Chart

```
plt.figure(figsize=(6,6)) slices = [90,80,30,70,10,100]
activities =
["Playing","Eating","Sleeping","Reading","Gyming","Gaming"]
cols = ["red","green","orange","purple","pink","yellow"]
plt.pie(slices,labels=activities,colors=cols,autopct="%1.2f%%",
explode=[0,0,0.3,0,0.3,0])
plt.show()
```

Output:

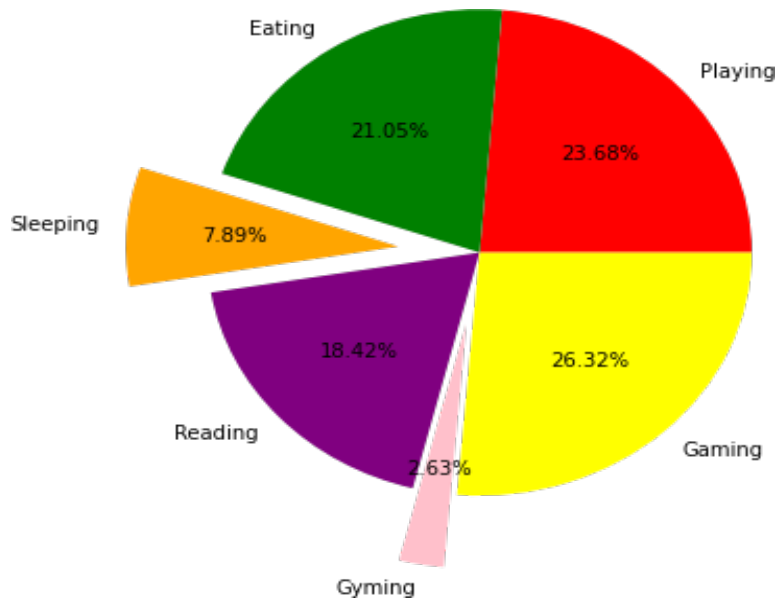


Figure 6: Pie chart with explode feature

Line Plot with two lines

```
x = [1,2,3,4,5]
y1 = [10,20,40,30,50]
y2 = [5,15,25,35,45]
plt.figure(figsize=(8,4))
plt.title("Line Plot Graph",fontsize=15,color='red')
plt.xlabel("X Axis",fontsize=12,color='blue') plt.ylabel("Y
Axis",fontsize=12,color='blue')
plt.xticks(fontsize=15,color='green')
plt.yticks(fontsize=15,color='green')
plt.plot(x,y1,'--',label="Line Plot 1",color="purple",lw=1)
plt.plot(x,y2,'--',label="Line Plot 2",color="red",lw=1)
plt.legend(loc=2,fontsize=12)
plt.grid()
plt.show()
```

Output:



Figure 7: Line Plot with two lines in one diagram

Sub Plot

```
x = [1,2,3,4,5]
y1 = [10,20,40,30,50]
y2 = [5,15,25,35,45]
y3 = [54,154,254,354,445]
y4 = [25,215,225,325,425]
plt.figure(figsize=(10,8))
```

```
plt.subplot(2,2,1)
plt.plot(x,y1,label="Line Plot 1",color="purple",lw=1)
plt.title("Line Plot Graph 1",fontsize=15,color='red')
plt.subplot(2,2,2)
plt.plot(x,y2,label="Line Plot 2",color="purple",lw=1)
plt.title("Line Plot Graph 2",fontsize=15,color='blue')
plt.subplot(2,2,3)
plt.plot(x,y3,label="Line Plot 3",color="purple",lw=1)
plt.title("Line Plot Graph 3",fontsize=15,color='green')
plt.subplot(2,2,4)
plt.plot(x,y4,label="Line Plot 4",color="purple",lw=1)
plt.title("Line Plot Graph 4",fontsize=15,color='purple')
plt.savefig("graph.png")
plt.show()
```

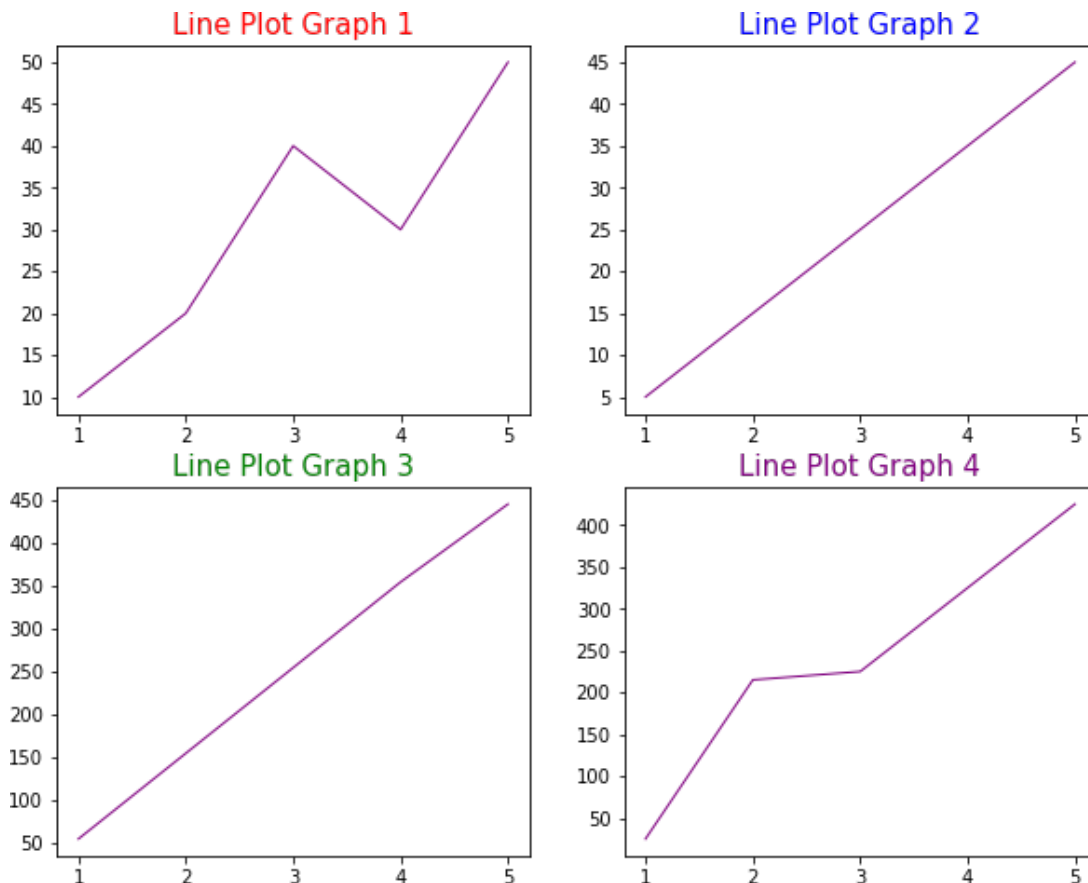


Figure 8: Sub plot feature with four different Line Plot graph

INTRODUCTION TO MACHINE LEARNING AND SCIKIT-LEARN LIBRARY

Ramasubramanian V.

ICAR-Indian Agricultural Statistics Research Institute, New Delhi - 110 012

r.subramanian@icar.gov.in

1. Introduction

Machine learning (ML) can be considered under the gamut of artificial intelligence with the subject of both statistics and computer science focusing on the use of data and algorithms to imitate the way that humans learn, gradually improving its accuracy. ML which usually resort to soft computing and are mostly data driven in the sense that they are very much complementary to the model driven techniques like, say, regression or time series models. They are also called data mining techniques owing to its very nature of mining and revealing any hidden information present in the data what with a large number of variables and data points involved and are very much computer intensive. In contrast to traditional statistical methods, using ML techniques, the very large database is allowed to bring out non-linear complex relationships that may exist between the variables involved which otherwise cannot be captured by the former.

With the objective of adapting to a given environment and learning from experience, ML techniques have immensely transformed the way data are analyzed. ML is programming computers to optimize a performance criterion using example data or past experience (Alpaydin, 2010). We need learning in cases where we cannot directly write a computer program to solve a given problem, but need example data or experience. One case where learning is necessary is when human expertise does not exist, or when humans are unable to explain their expertise. Consider the recognition of spoken speech—that is, converting the acoustic speech signal to an ASCII text; we can do this task seemingly without any difficulty, but we are unable to explain how we do it. Different people utter the same word differently due to differences in age, gender, or accent. In ML, the approach is to collect a large collection of sample utterances from different people and learn to map these into words.

Another case is when the problem to be solved changes in time, or depends on the particular environment. We would like to have general purpose systems that can adapt to their circumstances, rather than explicitly writing a different program for each special circumstance. Consider routing packets over a computer network. The path maximizing the quality of service from a source to destination changes continuously as the network traffic changes. A learning routing program is able to adapt to the best path by monitoring the network traffic.

To fix ideas, let us look definitions of ML given by other researchers. ML is the systematic study of algorithms and systems that improve their knowledge or

performance with experience (Flach, 2012). The term ML refers to the automated detection of meaningful patterns in data (Shwartz and David, 2014). It can be considered as a tool in almost any task that requires information extraction from large data sets. Many application areas are already encountered in our daily routine. Search engines learn by experience and give us the best results, antispam software learn over time to filter our email messages, and credit card transactions have in-built mechanisms that are secured by a software that, in a way, use ML to detect and prevent frauds. Digital cameras learn and are used for facial detection and intelligent personal assistance applications on smart-phones learn to recognize voice commands. One common feature of all of these applications is that, in contrast to more traditional uses of computers, in these cases, due to the complexity of the patterns that need to be detected, a human programmer cannot provide an explicit, fine-detailed specification of how such tasks should be executed. Taking example from intelligent beings, many of our skills are acquired or refined through learning from our experience (rather than following explicit instructions given to us). ML tools are concerned with endowing programs with the ability to “learn” and adapt. Hence automated learning is nothing but ML as computers are programmed in such a way that they “learn” from input available to them. Loosely, learning is the process of converting knowledge or experience into expertise. Here the input to a learning algorithm is ‘training data’, representing experience, and the output is some expertise.

Shwartz and David (2014) also cite an example to demonstrate a typical ML task. Suppose a machine is programmed to learn how to filter spam e-mails. A naive solution would be that the machine simply memorize all previous e-mails that had been labeled as spam e-mails by the human user. When a new e-mail arrives, the machine will search for it in the set of previous spam e-mails. If it matches one of them, it will be trashed (moved to Trash folder). Otherwise, it will be moved to the user’s Inbox folder. While the preceding “learning by memorization” approach is sometimes useful, it lacks an important aspect of learning systems – the ability to label unseen e-mail messages. A successful learner should be able to progress from individual examples to broader generalization. This is also referred to as inductive reasoning or inductive inference. To achieve generalization in the spam filtering task, the learner can scan the previously seen e-mails, and extract a set of words whose appearance in an e-mail message is indicative of spam. Then, when a new e-mail arrives, the machine can check whether one of the suspicious words appears in it, and predict its label accordingly. Such a system would potentially be able correctly predict the label of unseen e-mails.

Using the same spam email problem discussed above, Flach (2012) has given additional conceptual insights. A machine learning problem may have several solutions, but how one chooses among these solutions is the key to performing better. One way to think about this is to understand that we don’t really care that much about performance on training data – we already know which of those e-mails are spam. What we care about is whether future e-mails are going to be classified correctly. While this appears to lead into a vicious circle – in order to know whether an e-mail

is classified correctly one needs to know its true class, but as soon as one knows its true class, he/ she do not need the classifier anymore – it is important to keep in mind that good performance on training data is only a means to an end, not a goal in itself. In fact, trying too hard to achieve good performance on the training data can easily lead to a fascinating but potentially damaging phenomenon called ‘over-fitting’.

The concept of overfitting is further elaborated by Flach (2012) by means of another example. Suppose a student is preparing for an exam. Helpfully, the teacher has made previous exam papers and their worked answers are available. The student begins by trying to answer the questions from previous papers and compares his answers with the model answers provided. Unfortunately, he gets carried away and spends all his time on memorizing the model answers to all past questions. Now, if the upcoming exam completely consists of past questions, he is certain to do very well. But if the new exam asks different questions about the same material, he would be ill-prepared and get a much lower mark than with a more traditional preparation. In this case, one could say that he was overfitting the past exam papers and that the knowledge gained did not generalize to future exam questions. Generalization is thus the most fundamental concept in machine learning. Having said that, overfitting is not the only possible reason for poor performance on new data. It may just be that the training data used by the ML program to set its weights is not representative for the kind of, say, e-mails, one gets. One possible solution is to use (i.e. to supply to the ML program) different training data that exhibits the same characteristics, if possible actual spam and valid e-mails that have been received.

In order that such inductive reasoning might not lead us to false conclusions, learning mechanisms are developed to distinguish between useful learning and rote learning which is crucial to the development of automated learners. While human learners can rely on common sense to filter out random meaningless learning conclusions, once we export the task of learning to a machine, we must provide well defined crisp principles that will protect the program from reaching senseless or useless conclusions. The development of such principles is a central goal of the theory of ML.

Tasks that are too complex to program necessitates the need for ML. There are numerous tasks that human beings perform routinely, yet introspection concerning how they do them is not sufficiently elaborate to extract a well-defined program. Examples of such tasks include driving, speech recognition, and image understanding. In all of these tasks, state of the art ML programs that “learn from their experience,” achieve quite satisfactory results, once exposed to sufficiently many training examples. On the other hand, certain tasks are beyond human capabilities that belong to a wide family of tasks that benefit from ML techniques related to the analysis of very large and complex data sets. These tasks/ data sets could be astronomical data, turning medical archives into medical knowledge, weather prediction, analysis of genomic data, web search engines etc. With more and more available digitally recorded data, it becomes obvious that there are treasures of meaningful information buried in data archives that are way too large and too complex for humans to make

sense of. Learning to detect meaningful patterns in large and complex data sets is a promising domain in which the combination of programs that learn with the almost unlimited memory capacity and ever-increasing processing speed of computers opens up new horizons.

Another important feature of ML is its adaptiveness. By contrast, one limiting feature of programmed tools is their rigidity – once the program has been written down and installed, it stays unchanged. However, many tasks change over time or from one user to another. ML tools – programs whose behavior adapts to their input data – offer a solution to such issues; they are, by nature, adaptive to changes in the environment they interact with. Typical successful applications of machine learning to such problems include programs that decode handwritten text, where a fixed program can adapt to variations between the handwriting of different users; spam detection programs, adapting automatically to changes in the nature of spam e-mails; and speech recognition programs.

In a sense, ML can be viewed as a branch of AI (Artificial Intelligence), since the ability to turn experience into expertise or to detect meaningful patterns in complex sensory data is a cornerstone of human (and animal) intelligence. However, one should note that, in contrast with traditional AI, ML is not trying to build automated imitation of intelligent behavior, but rather to use the strengths and special abilities of computers to complement human intelligence, often performing tasks that fall way beyond human capabilities. For example, the ability to scan and process huge databases allows ML programs to detect patterns that are outside the scope of human perception.

2. Different types of learning in ML

Machines analyze their own behavior, they learn from their own mistakes leading to taking the appropriate decisions based on its analysis. ‘Learn from your mistakes’ is easily said than followed. But statement has made so much of an impact in the minds of the technologists that they started adopting this technique of making machines learn from their mistakes so that they can intelligently work in their future actions. This act of parenting is the new cool in the tech world as the question always lingers in the readers’ minds as to how a machine would learn from the mistakes it commits. The logic behind this concept is very simple and easy to understand. It is very much like how a normal person learns from his mistake and how efficiently he uses his senses to avert committing the same mistake again.

It has been extensively elaborated in the aforesaid discussion that the property that is of utmost importance for a ML model or method is its capacity to learn from its environment and to improve its performance through learning. This learning happens through an interactive process which in turn adjusts the system and the ML method becomes more knowledgeable about its environment after the learning process. That is, we can say that the ML is stimulated by its environment and undergoes changes in its parameters as a result of this stimulation. Eventually,

the ML method responds in a new way to the environment because of the changes that have occurred in its internal structure.

Learning or training is the term used to describe the process of finding the values of the parameters of ML (in neural networks which is one of the several types of ML these parameters are called weights). The two chief types of learning broadly are supervised and unsupervised learning. Supervised learning, also called learning with a teacher, occurs when there is a known target value associated with each input in the training set, when we consider a neural network as an ML method. The output of the network is compared with the target value and this difference is used to train the network (alter the weights). There are many different algorithms for training neural networks using supervised learning with backpropagation (error correction learning) as one of the more common ones. A biological example of supervised learning is when you teach a child the alphabet. You show him or her a letter and based on his or her response you provide feedback to the child. This process is repeated for each letter until the child knows the alphabet.

Unsupervised learning is needed when the training data lacks target output values corresponding to input patterns. The network must learn to group or cluster the input patterns based on some common features. This type of training is also called learning without a teacher because there is no source of feedback in the training process. A biological example would be when a child touches a hot red tip of a mosquito coil lit. He or she soon learns, without any external teaching, not to touch it. In fact, the child may associate a bright red glow with heat and learn to avoid touching objects with this feature.

One particular class of unsupervised system is based on competitive learning. Here the output neurons compete amongst themselves to be activated, with the result that only one is activated at any one time. This activated neuron is called a winner-takes all neuron or simply the winning neuron. Such competition can be induced/implemented by having lateral inhibition connections (negative feedback paths) between the neurons. The result is that the neurons are forced to organise themselves, hence the name, 'Self Organizing Map (SOM)' which learning is used typically in one type of neural network architecture called Kohonen or Self Organizing Feature Map (SOFM).

One more type of learning is what is called the reinforcement learning. Now assume that a baby is trying to walk. For the first few days it would analyze how the people around are walking. It's learning starts right from seeing how others walk, how others move and what others do while walking and continues until it stands up and walks by itself. Now whenever a baby tries to stand up but falls, it learns from itself and again gets up to stand by itself and proceeds until it starts to walk.

According to Narayanan (2018), the concept of reinforcement learning can be likened to a game wherein the player (here the machine) gets a credit whenever he/she takes a right step towards achieving the goal and loses it whenever he takes a bad decision.

In Reinforcement Learning, the player which is an agent, manipulates the environment and then takes the decision. If the decision is correct there is a reward that gets added to the score (0 at the beginning) and if the decision is wrong the reward gets reduced. So, this process is allowed to happen until the agent attains victory in the game. According to the various decisions that are taken by the agent the overall score is calculated and with this information the best way of winning the game is formulated.

The difference between ML and Statistical Modelling (SM) have gone down significantly over past decade. While ML are mostly data driven, SM is the process of applying statistical analysis to a dataset by means of a mathematical representation of observed data. Both the branches have learned from each other a lot and will further come closer in future as they complement each other. A computer scientist might view SM as quaint while a statistician might express bewilderment at the buzz around ML and question why more principled, interpretable SM would not do the trick. This points towards two schools of thinking on problems with different goals.

3. Differences and similarities between ML and SM

Srivastava (2015) has discussed the differences between ML and SM in detail. The common objective behind using either of the tools is learning from data in the sense that both these approaches aim to learn about the underlying phenomena by using data generated in the process. Even though the end goal for both ML and SM is same, the formulation of two are quite different and at the same time the output is many times obtained in a still more different manner. While ML is an algorithm that can learn from data without relying on rules-based programming, SM is formalization of relationships between variables in the form of mathematical equations. More details on ML and on differences between ML and SM can be found in Ramasubramanian (2022a, 2022b).

Carmichael and Marron (2018), when discussing in detail about Data Science (in which one of the subdomains is ML) have given solid opinions and perspectives as to how the business of learning from data has come to be more of associated with data science which was traditionally the business of statistics. They thus argue that ML can be understood as a broader, task-driven and computationally-oriented version of statistics. In this way, evolving approaches to modern data analysis like ML relate to the existing discipline of SM.

While it may outwardly appear that ML is simply a rebranding of SM, there were changes that happened over a period of time to consider for discussing such a viewpoint from both old and new ways of thinking. While SM has been around for a long time, its economics (costs of and value derived) have changed primarily due to technology driving the availability of data, computational capabilities and ease of communication. The most obvious advances are in computer hardware (e.g. faster CPUs, smaller microchips, GPUs, distributed computing). Similarly, algorithmic advances play a big role in making computation faster and cheaper. There are many

new/ improving technologies which allow us to gather data in new, faster and cheaper ways, including drones, medical imaging, sensors, better robotics, etc. Improved software makes it faster, cheaper and easier to execute ML. Such drastic changes over time contributed to the fact that ML in fact churns out more value out of data. Areas of statistics previously considered specialized such as statistical software, exploratory analysis, data visualization, high dimensional analysis, complex data objects and the use of optimization methods have become dramatically more valuable and commonplace. Such an analysis about the genesis and development of ML may give insights into its aspects about why the rise of computation is tied to the rise of exploratory and predictive analysis.

No doubt, ML appears to be involving three major disciplines viz., mathematics/ statistics, computer science and the application domain but SM also do involve all these though not in an explicit fashion. ML focusses more on solving specific problems in contrast with the type of deep understanding that is typical in SM. ML tools are of direct benefit to the users while SM serve as the theoretical basis for developing such tools. It is noted here that computer science experts consider ML with its large-scale computation requiring data storage cum retrieval as belonging to their subject domain than statistics while SM is obviously the forte of statisticians. Even data science (with ML in it) is viewed by some as a subset of statistics while others argue statistics as a subset of data science.

Some are of the opinion that SM has too much theory and not enough computation, while ML is viewed exactly the opposite way even though such views are not fully correct. SM was primarily developed to help people deal with pre-computer era problems and analysis while ML emphasizes on data problems of this digital era, like accessing information from large databases, writing code to manipulate data, and visualizing data including modeling. Rather SM has changed significantly in response to new technology. SM while continuing to emphasize on theory, also have started focusing more on computing in order to take on the data problems of the computer age. Far from viewing SM as something old fashioned, it has transformed into ML. Having said that, the need for sound statistical and theoretical thinking is greater than ever in the data science age.

Breiman *et al.* (2001) discusses many of the differences between predictive modeling (read ML) and inferential modeling (read SM). ML often uses more sophisticated, computationally intensive models that comes with a loss in interpretability and general understanding about how the model works. ML also places less emphasis on theory/ assumptions because there are fairly good, external metrics to tell the analyst how well they are doing (e.g. test set error).

ML is one of the main drivers of artificial intelligence (AI). The fact that data can be used to help computers automate things using ML is perhaps one of the most impactful innovations of recent decades. Modern AI systems are typically based on deep learning (advancement of ML) and are extremely data hungry. One of the biggest ways data is impacting society is by powering automation through ML.

The decades of SM's hard-won knowledge about dealing with data is salient to many applications of data driven automation in ML. In addition, automation presents new technical challenges to SM since ML often involves applying sophisticated modeling techniques to large, complex datasets. One of the primary technical challenges is interpretability about understanding the model itself. Easily comprehensible models are generalized linear models (e.g. of SM) and decision trees (e.g. of ML) while an example of an uninterpretable model is a deep neural network (e.g. of ML). All said, ML still remains a black box model because the analyst cannot fully understand it.

ML, particularly deep learning, may cause complex data processing to be more frequently used in modern data analysis. A lot of work in ML is about measurement: image recognition, image captioning, speech recognition, machine translation, syntactic parsing, sentiment analysis, etc. These examples somewhat blur the line between data processing and data gathering. In due course of time, it will be more common for data analysts to have one or more variables in their datasets which are the output of a deep learning model. ML will create a lot of value in data processing in many domains.

The purpose of (supervised) ML is obtaining a model that can make repeatable predictions. One typically do not care if the model is interpretable, although it is always recommended that testing be done to ensure that model predictions do make sense. Stewart (2019) argues that ML is all about results, it is likely working in a company where your worth is characterized solely by your performance (say, predictions). Whereas, SM is more about finding relationships between variables and the significance of those relationships, whilst also catering for prediction.

Finally, as regards to the question of whether ML and SM are really two different and diverse tools, if statistics is viewed as the narrow discipline, then the answer is in the affirmative. However, if SM embraces the broader idea of greater data science in which ML is a sizable component, then they both become one and the same. ML generally sacrifice interpretability for predictive power. The pursuit of understanding ML models to make them more interpretable is a wide-open area for statisticians. These questions may involve narrowing the gap between ML and SM.

4. scikit-learn library for ML in Python

scikit-learn is an open source library for ML in Python (Pedregosa et al., 2011) integrating a wide range of state-of-the-art ML algorithms for medium-scale supervised and unsupervised problems. This package focuses on bringing ML to non-specialists using a general-purpose high-level language. Emphasis is put on ease of use, performance, documentation, and Application Programming Interface (API) consistency. It has minimal dependencies and is distributed under the simplified BSD license (which stands for Berkeley Source Distribution and is a low restriction and requirement license type used for the distribution of many freeware, shareware and open source software) encouraging its use in both academic and commercial usable

settings. Source code, binaries, and documentation can be downloaded from <http://scikit-learn.sourceforge.net>.

scikit-learn contains simple and efficient tools for predictive data analysis which are accessible to everybody, and reusable in various contexts. It has been built on NumPy, SciPy, and matplotlib of Python. sci-learn can be used for supervised learning problems of ML such as linear models, linear and quadratic discriminant analysis, kernel ridge regression, Support Vector Machines (SVM), stochastic gradient descent, nearest neighbors, Gaussian processes, naive Bayes, decision trees, ensemble methods, multiclass and multioutput algorithms, isotonic regression, neural network models using supervised learning algorithms. It can be used also for unsupervised learning problems such as Gaussian mixture models, Clustering and Neural network models that use unsupervised learning algorithms.

sci-learn can thus be used for classification (Identifying which category an object belongs to) with applications in spam detection, image recognition etc. via algorithms viz., SVM, nearest neighbors, random forest (RF) etc.; regression (i.e. predicting a continuous-valued attribute associated with an object) with applications in drug response, stock prices etc. via algorithms viz., Support Vector Regression (SVR), nearest neighbors, RF etc.; clustering (i.e. automatic grouping of similar objects into sets) with applications in customer segmentation, grouping experiment outcomes etc. via algorithms viz., k-Means, spectral clustering, mean-shift etc.; dimensionality reduction (i.e. reducing the number of random variables to consider) with applications in visualization, increased efficiency etc. via algorithms viz. Principal Component Analysis (PCA), feature selection, non-negative matrix factorization etc.; model selection and evaluation, dimensionality reduction, inspection, visualizations, dataset transformations and preprocessing; model selection (i.e. comparing, validating and choosing parameters and models) with applications in improved accuracy via parameter tuning etc. via algorithms viz., grid search, cross validation, metrics etc.; preprocessing (i.e. feature extraction and normalization) with applications in transforming input data such as text for use with ML algorithms etc. via algorithms viz., preprocessing, feature extraction etc.

References

- Alpaydin, E. (2010). *Introduction to machine learning*, Second edition, MIT, USA.
- Breiman, L. (2001). Statistical modeling: The two cultures (with comments and a rejoinder by the author), *Statistical Science*, **16(3)**, 199-231.
- Carmichael, I. and Marron, J. S. (2018). Data science vs. statistics: two cultures?, *Japanese Journal of Statistics and Data Science*, **1(1)**, 117-138.
- Flach, P. (2012). *Machine Learning - The art and science of algorithms that make sense of data*, Cambridge University Press, UK.

- Narayanan, V. (2018). Reinforcement learning: the art of teaching machines, <https://www.gavstech.com/reinforcement-learning-the-art-of-teaching-machines/>, GAVS engage magazine, March issue, accessed on January, 08, 2022.
- Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine Learning in Python, *Journal of Machine Learning Research*, **12(85)**, 2825–2830.
- Ramasubramanian V. (2022a). Machine learning: Concepts and applications, In: *Training on artificial intelligence for students: e-Compendium*, Eds. Kalambe, D., Panwar, H., Dash, SK, Sharma, A., and Mukhopadhyay, CS., Published as a training compendium under Institutional Development Plan (IDP), Guru Angad Dev Veterinary and Animal Sciences University, Ludhiana, Punjab. Edition-I. ISBN: 978-93-5636-707-4, Module 1: Basics of Artificial Intelligence: From scratch to conceptualization, January 10-13, 2022, pages 25-40.
- Ramasubramanian V. (2022b). Machine learning versus statistical modelling: Conceptual and practical differences, In: *Training on artificial intelligence for students: e-Compendium*, Eds. Kalambe, D., Panwar, H., Dash, SK, Sharma, A., and Mukhopadhyay, CS., Published as a training compendium under Institutional Development Plan (IDP), Guru Angad Dev Veterinary and Animal Sciences University, Ludhiana, Punjab. Edition-I. ISBN: 978-93-5636-707-4, Module 3: Diving deep into the concept of machine learning, February 08-11, 2022, pages 140-146.
- Shwartz, S.S. and David, S.B. (2014). *Understanding machine learning*, Cambridge University Press, USA.
- Srivastava, T. (2015). Difference between machine learning and statistical modelling, <https://www.analyticsvidhya.com/blog/2015/07/difference-machine-learning-statistical-modeling/>, accessed on January 28, 2022
- Stewart, M. (2019). The actual difference between statistics and machine learning, <https://towardsdatascience.com/the-actual-difference-between-statistics-and-machine-learning-64b49f07ea3>, accessed on January 28, 2022.

Regression Analysis using Scikit-Learn

Kanchan Sinha

ICAR-Indian Agricultural Statistics Research Institute, New Delhi - 110 012

kanchan.sinha@icar.gov.in

1. Introduction

In statistical modelling, regression analysis is a set of statistical processes for estimating the relationships between a dependent variable (often called the ‘outcome variable’) and one or more independent variables (often called ‘predictors’, ‘covariates’, or ‘features’). Regression analysis is primarily used for two distinct purposes. First, it is widely used for prediction and forecasting, which overlaps with the field of machine learning. Second, it is also used to infer causal relationships between independent and dependent variables. This methodology is widely used in business, social and behavioral sciences, biological sciences including agriculture. For example, yield of a crop can be predicted by utilizing the relationship between yield and other factors like water temperature, rainfall, quantity of fertilizer, quantity of seeds, irrigation level and relative humidity, etc.

A functional relationship between two variables can be expressed by a mathematical formula. If x denotes the independent variable and y the dependent variable, then y can be related x through a functional relation of the form $y = f(x)$. Given a particular value of x , the function f indicates the corresponding value of y . In regression analysis, the variable x is known as input variable, explanatory variable or predictor variable. This is an exact mathematical relationship. In statistical relation, may not be perfect owing to sampling. The above functional form is made a statistical model by adding an error term as $y = f(x) + \varepsilon$, where ε denotes the error term.

Depending on the nature of the relationships between x and y , regression approach may be classified into two broad categories *viz.*, linear regression models and nonlinear regression models. The response variable is generally related to other causal variables through some parameters. The models that are linear in these parameters are known as linear models, whereas in nonlinear models parameters appear nonlinearly.

2. Simple Linear Regression (SLR) Model

Simple linear regression is useful for finding relationship between two continuous variables. One is predictor or independent variable and other is response or dependent variable. It looks for statistical relationship but not deterministic relationship. Relationship between two variables is said to be deterministic if one variable can be accurately expressed by the other. For example, using temperature in degree Celsius it is possible to accurately predict Fahrenheit. Statistical relationship is not accurate in determining relationship between two variables. For example, relationship between height and weight. The core idea is to obtain a line that best fits the data. The best fit line is the one for which total prediction error (all data points) are as small as possible. Error is the distance between the point to the regression line.

The simple linear regression model is usually written as

$$y_i = \beta_0 + \beta_1 x_i + \varepsilon_i \quad (1)$$

where the ε_i 's are normal random variables with mean 0 and variance σ^2 . The model implies (i) The average y -value at a given x -value is linearly related to x .

(ii) The variation in responses y at a given x value is constant.

(iii) The population of responses y at a given x is normally distributed.

(iv) The observed data are a random sample.

Regression model (1) is said to be simple and linear regression model. It is “simple” in the sense that there is only one predictor variable and “linear” in the sense that all parameters appeared linearly with the predictor variables. The parameters β_0 and β_1 in regression model (1) are called regression coefficients, β_1 is the slope of the regression line. It indicates the change in the mean of the probability distribution of y per unit increase in x . The parameter β_0 is the y intercept of the regression line.

2.1 Estimation of Parameters in a Simple Linear Regression Model

In the above models the variables y and x are known, these are observed. The only unknown quantities are the parameters β 's. In regression analysis, our main concern is how precisely we can estimate these parameters. Once these parameters are estimated, our model becomes known and we can use it for further analysis. The method of least squares is generally used to estimate these parameters. For each observations (x_i, y_i) , the method of least squares considers the error of each observation, i.e, for a simple model $\varepsilon_i = y_i - \beta_0 - \beta_1 x_i$. The method of least squares requires the sum of the n squared errors. This criterion is denoted by S :

$$S = \sum_{i=1}^n (y_i - \beta_0 - \beta_1 x_i)^2 \quad (2)$$

According to the method of least squares, the estimators of β_0 and β_1 are those values of $\hat{\beta}_0$ and $\hat{\beta}_1$, respectively, that minimize the criterion S for the given observations. To minimize S , we differentiate S with respect to each parameter and equate to zero. We get as many equations as the number of parameters. Solving these equations simultaneously, we get the estimates of parameters. For example, for the regression model (1) the values of $\hat{\beta}_0$ and $\hat{\beta}_1$ that minimizes S for any particular set of sample data are given by the following simultaneous equations:

$$\sum_{i=1}^n y_i = n\hat{\beta}_0 + \hat{\beta}_1 \sum_{i=1}^n x_i \quad (3)$$

$$\sum_{i=1}^n x_i y_i = \hat{\beta}_0 \sum_{i=1}^n x_i + \hat{\beta}_1 \sum_{i=1}^n x_i^2 \quad (4)$$

These two equations are called normal equations and can be solved for $\hat{\beta}_0$ and $\hat{\beta}_1$ as follows

$$\hat{\beta}_1 = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^n (x_i - \bar{x})^2} \quad (5)$$

$$\hat{\beta}_0 = \frac{1}{n} (\sum_{i=1}^n y_i - \hat{\beta}_1 \sum_{i=1}^n x_i) = \bar{y} - \hat{\beta}_1 \bar{x} \quad (6)$$

where, \bar{y} and \bar{x} are the means of the y_i and x_i observations, respectively.

3. Multiple Linear Regression Model (MLR) Model

A regression model that involves more than one regressor variable is called a multiple regression model i.e., the multiple linear regression model is used to study the relationship between a dependent variable and one or more independent variables. The generic form of the linear regression model is

$$y = f(x_1, x_2, \dots, x_p) + \varepsilon = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_p x_p + \varepsilon \quad (7)$$

where, y is the dependent or explained variable and x_1, x_2, \dots, x_p are the independent or explanatory variables. The regression model in the equation describes above is linear in the sense, it is a linear function of the unknown parameters $\beta_0, \beta_1, \beta_2, \dots, \beta_p$. In general, any regression model that is linear in the parameters (β 's) is a linear regression model, regardless of the shape of the surface that it generates. We have also assumed that the expected value of the error term ε is zero. The parameter β_0 is the intercept of the regression model. If the range of the data includes $x_1 = x_2 = \dots = x_p = 0$, then β_0 is the mean of y when $x_1 = x_2 = \dots = x_p = 0$. Otherwise β_0 has no physical interpretation. The parameter β_1 indicates the expected change in response

(y) per unit change in x_1 when x_2, \dots, x_p are held constant. Similarly β_2 measures the expected change in response (y) per unit change in x_2 when x_1, \dots, x_p are held constant. For this reason the parameters $\beta_i, \forall i = 1, 2, \dots, p$ are often called as partial regression coefficients.

3.1 Assumptions of the Multiple Linear Regression Model

i. **Linearity**

The model defined by the following equation

$y = f(x_1, x_2, \dots, x_p) + \varepsilon = \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_p x_p + \varepsilon$ specifies a linear relationship between y and \mathbf{x} and our primary interest is in estimation and inference about the parameter vector $\boldsymbol{\beta}$. For the regression to be linear in the sense described here, it must be of the form in the original variables or after some suitable transformation.

ii. **Full rank**

There are no exact linear relationships among the variables in the model. \mathbf{x} is an $n \times p$ matrix with rank p . Hence \mathbf{x} has full column rank; the columns of \mathbf{x} are linearly independent and there are at least p observations ($n \geq p$).

iii. **Exogeneity of the independent variables:**

The disturbance is assumed to have conditional expected value zero at every observation, which we can write as $E[\varepsilon_i | \mathbf{x}] = \mathbf{0}$.

In this equation, the left hand side states, in principle, that the mean of each ε_i conditioned on all observations \mathbf{x} is zero. This strict exogeneity assumption states, in words, that no observations on \mathbf{x} convey information about the expected value of the disturbance.

iv. **Homoscedasticity:**

The fourth assumption concerns the variances and covariance of the disturbances:

$$Var(\varepsilon_i | \mathbf{x}) = \sigma^2, \forall i = 1, \dots, n$$

$$Cov(\varepsilon_i, \varepsilon_j | \mathbf{x}) = 0 \forall i \neq j \quad (8)$$

Constant variance is labelled **homoscedasticity**. Consider a model that describes the profits of firms in an industry as a function of, say, size. Even accounting for size, measured in dollar terms, the profits of large firms will exhibit greater variation than those of smaller firms. The homoscedasticity assumption would be inappropriate here. Survey data on household expenditure patterns often display marked

heteroscedasticity, even after accounting for income and household size. The two assumptions imply that

$$E[\varepsilon\varepsilon'|\mathbf{x}] = \begin{bmatrix} \sigma^2 & 0 & \dots & 0 \\ 0 & \sigma^2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \sigma^2 \end{bmatrix} = \sigma^2 \mathbf{I}$$

(9)

v. **Data generating process for the regressors**

It is common to assume that \mathbf{x}_i is nonstochastic, as it would be in an experimental situation. Here the analyst chooses the values of the regressors and then observes y_i . This process might apply, for example, in an agricultural experiment in which y_i is yield and \mathbf{x}_i is fertilizer concentration and water applied.

vi. **Normality**

It is convenient to assume that the disturbances are normally distributed with zero mean and constant variance. This is a convenience that we will dispense with after some analysis of its implications. The normality assumption is useful for defining the computations behind statistical inference about the regression, such as confidence intervals and hypothesis tests. For practical purposes, it will be useful then to extend those results and in the process develop a more flexible approach that does not rely on this specific assumption.

$$\varepsilon|\mathbf{x} \sim N(\mathbf{0}, \sigma^2 \mathbf{I}) \tag{10}$$

The validity of these assumptions is needed for the results to be meaningful. If these assumptions are violated, the result can be incorrect and may have serious consequences. If these departures are small, the final result may not be changed significantly. But if the deviations are large, the model obtained may become unstable in the sense that a different sample could lead to an entirely different model with opposite conclusions. So such underlying assumptions have to be verified before attempting to regression modeling. One crucial point to keep in mind is that these assumptions are for the population, and we work only with a sample. So the main issue is to make a decision about the population on the basis of a sample of data. Several diagnostic methods to check the violation of regression assumption are based on the study of model residuals and also with the help of various types of graphics.

3.2 Estimation of Parameters in a Multiple Linear Regression (MLR) Model

The method of least squares can be used to estimate the regression coefficients in Eq. (7). Suppose that $n > p$ observations are available, and let y_i denote the i th observed response and x_{ij} denote i th observation or level of regressor x_j . The data will appear in the following table 1. We also assume that the error term ε in the model has $E(\varepsilon) = 0$ and $Var(\varepsilon) = \sigma^2$, and the errors are uncorrelated.

Table 1: Data for Multiple Linear Regression

Observation, i	Response, y	Regressors		
		x_1	x_2	x_p
1	y_1	x_{11}	x_{12}	x_{1p}
2	y_2	x_{21}	x_{22}	x_{2p}
.
.
.
n	y_n	x_{n1}	x_{n2}	x_{np}

We may write the sample regression model corresponding to (7) as

$$\begin{aligned}
 y &= \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \dots + \beta_p x_{ip} + \varepsilon \\
 &= \beta_0 + \sum_{j=1}^p \beta_j x_{ij} + \varepsilon_i, \forall i = 1, 2, \dots, n
 \end{aligned}
 \tag{11}$$

The least - squares function is then used to estimate the model parameters, which are obtained by minimizing the error sum of squares with respect to the parameters $\beta_0, \beta_1, \beta_2, \dots, \beta_p$.

It is more convenient to deal with multiple regression models if they are expressed in matrix notation. This allows a very compact display of the model, data, and results.

In matrix notation, we can express the multiple regression model as

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}
 \tag{12}$$

Where

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ \cdot \\ y_n \end{bmatrix} \quad \mathbf{X} = \begin{bmatrix} 1 & x_{11} & \cdots & x_{1p} \\ 1 & x_{21} & \cdots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & \cdots & x_{np} \end{bmatrix} \quad \boldsymbol{\beta} = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_p \end{bmatrix} \quad \boldsymbol{\varepsilon} = \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \cdot \\ \varepsilon_n \end{bmatrix}$$

\mathbf{y} is a $n \times 1$ vector of responses

\mathbf{X} is a $n \times p$ matrix of the regressor variables

$\boldsymbol{\beta}$ is a $n \times 1$ vector of unknown constants, and

$\boldsymbol{\varepsilon}$ is a $n \times 1$ vector of random errors with $\varepsilon_i \sim \text{NID}(0, \sigma^2)$

We wish to find the vector of least-squares estimators, $\hat{\boldsymbol{\beta}}$ that minimizes

$$S(\boldsymbol{\beta}) = \sum_{i=1}^n \varepsilon_i^2 = \boldsymbol{\varepsilon}' \boldsymbol{\varepsilon} = (\mathbf{y} - \mathbf{X}\boldsymbol{\beta})'(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}) \quad (13)$$

Note that $S(\boldsymbol{\beta})$ may be expressed as

$$\begin{aligned} S(\boldsymbol{\beta}) &= \mathbf{y}'\mathbf{y} - \boldsymbol{\beta}'\mathbf{X}'\mathbf{y} - \mathbf{y}'\mathbf{X}\boldsymbol{\beta} + \boldsymbol{\beta}'\mathbf{X}'\mathbf{X}\boldsymbol{\beta} \\ &= \mathbf{y}'\mathbf{y} - 2\boldsymbol{\beta}'\mathbf{X}'\mathbf{y} + \boldsymbol{\beta}'\mathbf{X}'\mathbf{X}\boldsymbol{\beta} \end{aligned} \quad (14)$$

Since $\boldsymbol{\beta}'\mathbf{X}'\mathbf{y}$ is a 1×1 matrix, or a scalar, and its transpose $(\boldsymbol{\beta}'\mathbf{X}'\mathbf{y})' = \mathbf{y}'\mathbf{X}\boldsymbol{\beta}$ is the same scalar. The least square estimators must satisfy

$$\frac{\partial S}{\partial \boldsymbol{\beta}} = -2\mathbf{X}'\mathbf{y} + 2\mathbf{X}'\mathbf{X}\hat{\boldsymbol{\beta}} = \mathbf{0}$$

Which simplifies

$$\mathbf{X}'\mathbf{X}\hat{\boldsymbol{\beta}} = \mathbf{X}'\mathbf{y} \quad (15)$$

To solve the normal equations, multiply both sides of (iv) by the inverse of $\mathbf{X}'\mathbf{X}$. Thus the least squares estimator of

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{y} \quad (16)$$

So, the vector of fitted values \hat{y}_i corresponding to the observed value y_i is

$$\hat{\mathbf{y}} = \mathbf{X}\hat{\boldsymbol{\beta}} = \mathbf{X}(\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{y} \quad (17)$$

The difference between the observed value y_i and the corresponding fitted values \hat{y}_i is the residual i.e., $e_i = y_i - \hat{y}_i$. The n residuals may be conveniently written in matrix notation as

$$\mathbf{e} = \mathbf{y} - \hat{\mathbf{y}} \quad (18)$$

4. Estimation of Error Term Variance (σ^2)

The variance σ^2 of the error terms ε_i in regression model needs to be estimated to know the variability of the probability distribution of y . In addition, a variety of inferences concerning the regression function and the prediction of y require an estimate of σ^2 . Denote by $SSE = \sum_{i=1}^n (y_i - \hat{y}_i)^2 = \sum_{i=1}^n r_i^2$, is the residual sum of squares. Then an estimate of σ^2 is given by,

$$\hat{\sigma}^2 = \frac{SSE}{n-p}$$

(19)

where p is the total number of parameters involved in the model including the intercept term, if the model contains it. We also denote this quantity by MSE.

5. Inferences in Linear Regression Models

In multiple linear regression model, all variables may not be contributing significantly to the model. In other word, each of the parameters may not be significant. Therefore, these parameters must be tested whether they are significantly different from zero or not. That is, we test the null hypothesis (H_0) against the alternative hypothesis (H_1) for a parameter β_i (say) as follows:

$$H_0: \beta_i = 0 \tag{20}$$

$$H_1: \beta_i \neq 0$$

when $H_0: \beta_i = 0$ is accepted we infer that there is no linear association between y and x_i . For normal error regression model, the condition β_i implies even more than no linear association between y and x_i . $\beta_i = 0$ for the normal error regression model implies not only that there is no linear association between y and x_i but also that there is no relation of any kind between y and x_i , since the probability distribution of y are then identical at all levels of x_i . The test is based on t test

$$t = \frac{\beta_i}{s(\beta_i)} \tag{21}$$

where $s(\beta_i)$ is the standard error of β_i and calculated as $s(\beta_i) = \sqrt{\frac{MSE}{\sum_{i=1}^n (x_i - \bar{x})^2}}$

The decision rule with this test statistic when controlling level of significance at α is

if $|t| \leq t(1 - \alpha/2; n - p)$ conclude H_0 ,

if $|t| > t(1 - \alpha/2; n - p)$ conclude H_1 .

Similarly testing for other parameters can be carried out.

6. Measures of Fitting (R^2)

The overall fitting of a regression line can be judged by the F -statistic by carrying out an analysis of variance. If the F -statistic is significant, we say that our model is fitted well. However, there are times when the degree of linear association is of interest. A frequently used statistic is R^2 . We describe this descriptive measure to describe the degree of linear association between y and x .

Denote by $TSS = \sum_i^n (y_i - \bar{y})^2$, total sum of squares which measures the variation in the observation y_i , or the uncertainty in predicting y , when no account of the predictor variable x is taken. Thus TSS is a measure of uncertainty in predicting y when x is

not considered. Similarly, SSE measures the variation in the y_i when a regression model utilizing the predictor variable x is employed. A natural measure of the effect of x in reducing the variation in y , i.e., in reducing the uncertainty in predicting y , is to express the reduction in variation ($TSS - SSE = SSR$) as a proportion of the total variation and it is denoted by

$$R^2 = \frac{SSR}{TSS} = 1 - \frac{SSE}{TSS} \quad (22)$$

The measure R^2 is called coefficient of determination and $0 \leq R^2 \leq 1$. In practice R^2 is not likely to be 0 or 1 but somewhere between these limits. The closer it is to 1, the greater is said to be the degree of linear association between x and y . Remember that R^2 statistic should be used only when in the model an intercept term is involved. For the model with no intercept, R^2 is not a good statistic. In case of “no intercept” model, sum of all residuals may not be equal to 0, making R^2 inflated.

7. An Illustration of a MLR model

Consider the following data:

Table 2: y as a response variable and x 's as explanatory variables

Case No.	x_1	x_2	x_3	y	Case No.	x_1	x_2	x_3	y
1	12.980	0.317	9.998	57.702	14	14.231	10.401	1.041	41.896
2	14.295	2.028	6.776	59.296	15	15.222	1.220	6.149	63.264
3	15.531	5.305	2.947	56.166	16	15.740	10.612	-1.691	45.798
4	15.133	4.738	4.201	55.767	17	14.958	4.815	4.111	58.699
5	15.342	7.038	2.053	51.722	18	14.125	3.153	8.453	50.086
6	17.149	5.982	-0.055	60.446	19	16.391	9.698	-1.714	48.890
7	15.462	2.737	4.657	60.715	20	16.452	3.912	2.145	62.213
8	12.801	10.663	3.048	37.447	21	13.535	7.625	3.851	45.625
9	17.039	5.132	0.257	60.974	22	14.199	4.474	5.112	53.923
10	13.172	2.039	8.738	55.270	23	15.837	5.753	2.087	55.799
11	16.125	2.271	2.101	59.289	24	16.565	8.546	8.974	56.741
12	14.340	4.077	5.545	54.027	25	13.322	8.589	4.011	43.145
13	12.923	2.643	9.331	53.199	26	15.949	8.290	-0.248	50.706

In the present example, we have 3 three predictor variables x_1 , x_2 and x_3 and there are 26 observations. The response variable denoted by y . Applying least square method we obtain the parameter estimates as follows:

Table 3: ANOVA of a MLR model

Source	Degrees of freedom	Sum of Square	Mean Square	F-value	Prob. > F
Model	3	1062.34	354.11	109.69	<0.0001
Error	22	71.02	3.22		
Corrected Total	25	1133.37			

Table 4: Parameter Estimates of a MLR model

Variable	Degrees of freedom	Parameter Estimates	Standard Error	t-value	Prob. > t
Intercept	1	8.19	6.29	1.30	0.2060
x_1	1	3.56	0.36	9.86	<.0001
x_2	1	-1.64	0.15	-10.28	<.0001
x_3	1	0.33	0.17	1.88	0.0741

The value of R^2 of this model is 0.93. From Table 3, we see that F -statistic is highly significant, indicating that overall model fitting is good. R^2 is also very high. The fitted regression line is $\hat{y} = 8.19 + 3.56x_1 - 1.64x_2 + 0.33x_3$. The corresponding standard errors are given in the 4th column of Table 3. However, while testing the significance of the parameter estimates, we find that the intercept and the parameter for the variable x_3 , i.e., are not significant at 5% level of significance (probability values for these parameters are greater than 0.05).

8. An example of how to implement a multiple linear regression model in Python using the scikit-learn library

```
import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
data=pd.read_csv('yield_data.csv')
# split the data into features (X) and target (y) arrays
X = data.drop("yield", axis=1)
y = data["yield"]
# split the data into training and test sets
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
=0.2)
# create a Linear Regression model
mlr = LinearRegression()
# fit the model to the training data
mlr.fit(X_train, y_train)
#Prediction of test set
y_pred_mlr= mlr.predict(X_test)
#Predicted values
print("Prediction for test set: {}".format(y_pred_mlr))
# evaluate the model performance
print("R^2:", mlr.score(X_test, y_test))
```

9. References

- Chatterjee, S. and Price, B. (1977). *Regression Analysis by Example*, New York: John Wiley & Sons.
- Draper, N. R. and Smith, H. (1998). *Applied Regression Analysis*, New York: Wiley Eastern Ltd.
- Montgomery, D. C., Peck, E. and Vining, G. (2003). *Introduction to Linear Regression Analysis*, 3rd Edition, New York: John Wiley and Sons.

An Overview of Support Vector Machines using Scikit Learn

Mrinmoy Ray

ICAR-Indian Agricultural Statistics Research Institute, New Delhi - 110 012

mrinmoy.ray@icar.gov.in

1. Introduction:

Machine learning is a technique that enables a machine to learn on its own. The Support Vector Machine (SVM) is a well-known supervised machine learning technique developed by Cortes and Vapnik (1995) for binary classification problems. The SVM's goal in binary classification is to find a hyperplane that best divides a dataset into two classes. Vapnik et al. (1997) developed support vector regression (SVR) to deal with regression problems two years after SVM's invention, based on similar principles as SVM classification. SVR, as a non-parametric method, does not rely on assumptions such as linear regression. Another advantage of using SVR is that it allows for the creation of non-linear models. As a result, SVM is popular not only for classification but also for modelling and prediction. The performance of SVM is determined by the kernel that is used. There are various types of kernels that can be used for classification and prediction. SVM has been applied to time series modelling and forecasting in a variety of areas over the last decade, including power load forecasting (Niu et al., 2010), rainfall forecasting (Ortiz-Garcia et al., 2014), wind power forecasting (De Giorgi et al., 2014), and agricultural forecasting (Kumar and Prajneshu, 2015).

2. Support Vector Machine (SVM) in time series:

Application of SVM in time series is generally utilized when the series shows non stationarity and non-linearity process. A tremendous advantage of SVM is that it is not model dependent as well as independent of stationarity and linearity. However, it may be computationally expensive during the training. The training of the data driven prediction process SVM is done by a function which is estimated utilizing the observed data. Let, a time series $y(t)$ which takes the data at time t $\{t = 0,1,2,3, \dots, N\}$.

Now, the prediction function for linear regression is defined as:

$$f(y) = (w \cdot y) + c \quad (1)$$

Whereas, for non linear regression, it will be:

$$f(y) = (w \cdot \phi(y)) + c \quad (2)$$

Where, w denotes the weights, c represents threshold value and $\phi(y)$ is known as kernel function.

If the observed data is linear, then equation (1) will be used. But, for non-linear data, the mapping of $y(t)$ is done to the higher dimension feature space through some

function which is denoted as $\phi(y)$ and eventually it is transformed into the linear process. After that, a linear regression will carry out in that feature space.

The first and foremost objective is to find out the value of w and c which will be optimal. In SVM, there are two things viz., flatness of weights and error after the estimation which are to be minimized. The flatness of the weights is denoted by $\|w\|^2$ which is the euclidian norm. Firstly, one has to concentrate on minimization the $\|w\|^2$. Second important thing is the minimization of the error. This is also called as empirical risk. However, the overall aim is to minimize the regularized risk which is sum of empirical risk and the half of the product of the flatness of weight and a constant term which is known as regularized constant. The regularized risk can be written as-

$$R_{reg}(f) = R_{emp}(f) + \frac{\tau}{2} \|w\|^2 \quad (3)$$

Where, $R_{reg}(f)$ is the regularized risk, $R_{emp}(f)$ denotes the empirical risk, τ is a constant which is called as regularized constant/capacity control term and $\|w\|^2$ is the flatness of weights.

The regularization constant has a significant impact on a better fitting of the data and it can also be useful for the minimization of bad generalization effects. In the other words, this constant deals with the problem of over-fitting. The overfitting of the data can be reduced by the proper selection of this constant value. The empirical risk can be defined as:-

$$R_{emp}(f) = \frac{1}{N} \sum_{i=0}^{N-1} L(y(i), \alpha(i), f(y(i), w)) \quad (4)$$

Where, $\alpha(i)$ denotes the truth data of predicted value, $L(.)$ is known as loss function and i represents the index to the time series.

There are various types of loss function in literature. But, two functions viz., vavnik loss function and quadratic loss function are most popular and they are generally used. The quadratic programming problem has been made to minimize the regularised risk which is-

$$\text{Minimize, } \frac{1}{2} \|w\|^2 + D \sum_{i=1}^n L(\alpha(i), f(y(i), w)) \quad (5)$$

Where,

$$L(\alpha(i), f(y(i), w)) = |\alpha(i) - f(y(i), w)| - \epsilon \text{ if } |\alpha(i) - f(y(i), w)| \geq \epsilon \\ = 0; \text{ otherwise.}$$

Where, D is a constant which equals to the summation normalization factor and ϵ represents the size of the tube.

The computation of ϵ and D is done empirically because they are user defined. One has to choose proper value of D and ϵ . Now, dual optimization problem is formed using the lagrange multiplier which can be written as:

$$\text{Maximize,} \quad -\frac{1}{2} \sum_{i,j=1}^N (\beta_i - \beta_i^*) (\beta_j - \beta_j^*) \langle y(i), y(j) \rangle - \epsilon \sum_{i=1}^N (\beta_i - \beta_i^*) + \sum_{i=1}^N \alpha(i) (\beta_i - \beta_i^*) \quad (6)$$

Subject to, $\sum_{i=1}^N (\beta_i - \beta_i^*) = 0$; $\beta_i, \beta_i^* \in [0, D]$

The function $f(x)$ is defined as;

$$f(x) = \sum_{i=1}^N (\beta_i - \beta_i^*) \langle y, y(i) \rangle + C \quad (7)$$

KKT conditions are used to get the solution of the weights.

The significance of kernel function in non-linear support vector machine (NLSVR) is very much important for mapping the data $y(i)$ into higher dimension feature space $\Phi(y(i))$ in which the data becomes linear. Generally notation for kernel function is given as;

$$k(y, y') = \langle \Phi(y), \Phi(y') \rangle; \quad (8)$$

There are many methods in literature to solve the quadratic programming. However, the most used method is sequential minimization optimization (SMO) algorithm.

3. Kernel function

SVM is a learning algorithm which is based on kernel. There are different types of kernel which can be used for the classification and prediction purpose. However, there is no such rule to make inference on which kernel should one use. All the kernels are used separately for the given datasets and whichever gives the better result, one should choose that one. Various types Kernel are listed below:

1. Non linear
2. Linear
3. Polynomial
4. Radial basis function: a) Gaussian Radial basis function b) Laplace Radial basis function.
5. Sigmoid kernel
6. Hyperbolic tangent kernel
7. Anova radial basis kernel
8. Multi-layer perceptron
9. Linear spline kernel.

Kernel function are used for the transformation of the given data into the required form. Kernel function is actually a mathematical function. RBF is mostly used kernel function. Some kernel functions are described in the following:

Polynomial kernel equation: Polynomial kernel is generally used in the image processing. It is useful for nonlinear modelling. This kernel function is very simple yet efficient method.

$$k(x, y) = (x \cdot y + 1)^p ; p = \text{degree of polynomial} \quad (9)$$

Gaussian kernel function:

$$k(x, y) = \exp\left(-\frac{\|x-y\|^2}{2\sigma^2}\right) \quad (10)$$

Or $k(x, y) = \exp(-\alpha\|x - y\|^2)$, Where, shape of hyperplane is controlled by σ .

Sigmoid kernel function:

Sigmoid function is used as the proxy of artificial neural network.

$$k(x, y) = \tanh(\theta x^T \cdot y + a) \quad (11)$$

Linear kernel function:

Sometimes, linear kernel gives better results as compared to complex and nonlinear kernels. Linear

classifier can be used to test the non-linearity of the datasets.

$$k(x, y) = x \cdot y \quad (12)$$

4. Advantages of SVM:

1. It gives global optimum.
2. Training of SVM is comparatively easier than other machine learning techniques.
3. Well scaling for data with high dimensionality.
4. It can give a good prediction.
5. It is based on statistical learning theory.
6. Work on structural risk minimization.
7. Risk of overfitting problem may overcome by SVM.
8. It has good generalization property.
9. It is useful when there is no prior information about the data.
10. It also work on unstructured data.

5. Illustration:

Data Description:

Time series data on Cotton Production (Million Bales) of India from 1950 to 2016 were taken from the Ministry of Agriculture & Farmers Welfare, Government of India. The data from 1950-2011 have been utilized for model building purpose and the data from 2012 to 2016 were used to predict the cotton production for the validation purpose.

Support Vector Machine:

The most important part in SVM technique is the selection of parameters and kernel which have to be selected with utmost care to improve the performance of the model in order to get better accuracy in forecasting. The best parameters and kernel have been selected using “e1701” package (David, 2017) in R software.

The time series plot of cotton production is illustrated in Fig. 1. It can be seen from Table 1 that the time series show a high value of coefficient variation which represents the presence of highly heterogenous characteristic of the series.

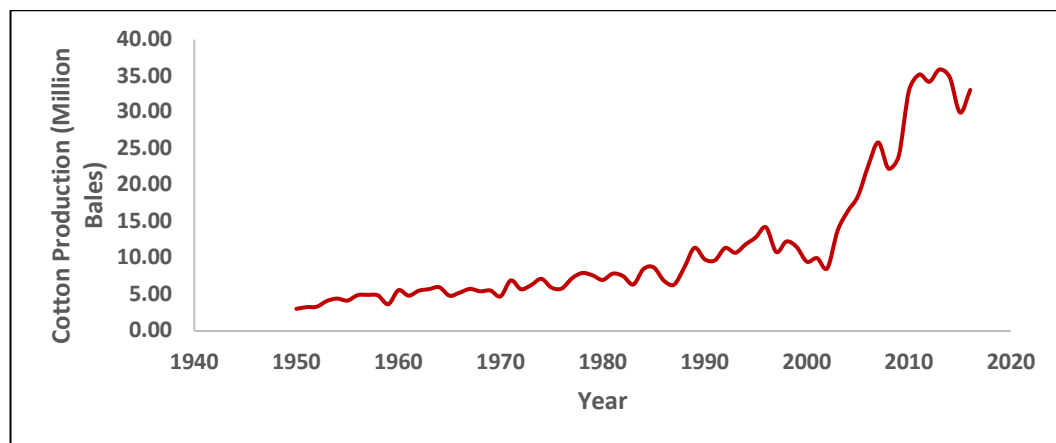


Fig. 1: Time Series Plot of Cotton Production

Table 1: Summary Statistics of Cotton Production

Statistic	Value	Statistic	Value
Minimum	3.04	Maximum	33.20
1 st Quartile	5.54	Standard Deviation	6.81
Median	7.20	Skewness	2.05
Mean	9.60	Kurtosis	4.09
3 rd Quartile	11.26	Coefficient of Variation	70.93

Table 2 displays the estimated best parameters of SVR after sufficient tuning of SVR model and these best parameters have been utilized to build the SVR model. It has been seen that the best SVM-kernel function is Radial basis function for SVR.

Table 2: Parameter estimation of SVR

Sampling method	10-fold cross validation
Epsilon (Best Parameter)	0.1
Cost (Best Parameter)	4
Gamma (Best Parameter)	1
Number of Support Vectors	39
SVM-Type	eps-regression
SVM-Kernel	Radial Basis Function

Fig. 2 shows the graphical representation of the performance of the models for Cotton Production series. Model performance in terms of MSE, MAE and MAPE has been shown in Table 3 and Table 4 for training and testing dataset respectively. Here, ARIMA (2, 2, 1) model has been fitted based on the lowest AIC values among various ARIMA models and the data of cotton production show the non-linearity pattern which is tested by Brock, Dechert and Scheinkman (BDS) test.

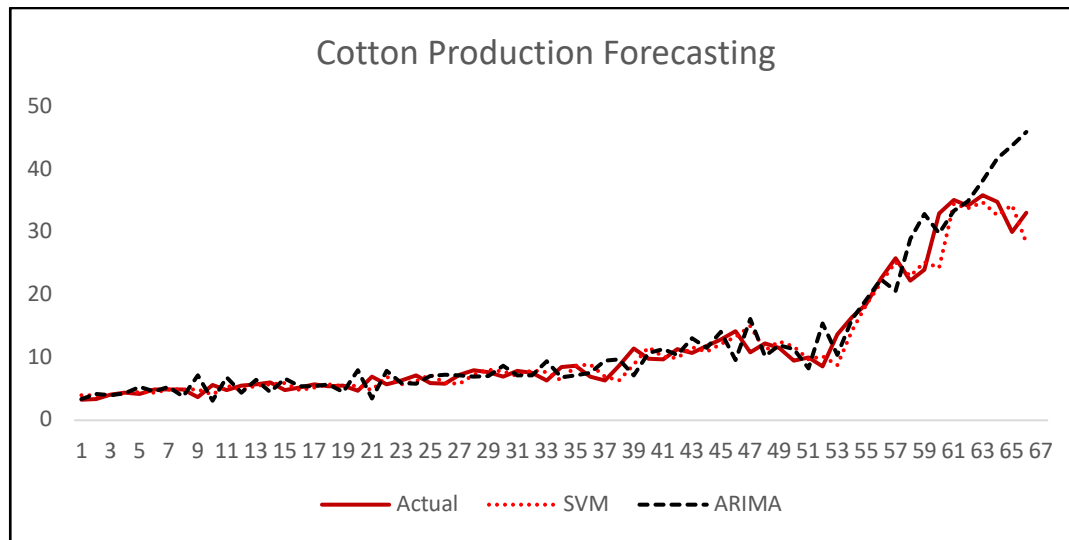


Fig. 2: Graphical representation of the performance of ARIMA and SVM models

Table 3: Model performance in training dataset using ARIMA and SVM

Model	MSE	MAE	MAPE
-------	-----	-----	------

ARIMA	6.70	1.83	21.28
SVM	3.08	1.14	12.73

Table 4: Model performance in testing dataset using ARIMA and SVM

Model	MSE	MAE	MAPE
ARIMA	82.45	7.35	22.76
SVM	9.48	2.54	7.83

Table 5 displays the Out-of-Sample forecast values using ARIMA and SVM.

Table 5: Model performance in testing dataset using ARIMA and SVM

Year	Actual	ARIMA	SVM
2012	34.22	34.98	33.85
2013	35.9	38.21	34.78
2014	34.81	41.79	32.67
2015	30.0	43.82	34.33
2016	33.09	45.99	28.32

It has been seen from the Fig. 2 that the fitted graph of the SVM model is more close to the graph of original data as compare to ARIMA model both in training and forecasting. It is observed from Table 3 and Table 4 that the SVM has a lower MSE, MAE and MAPE compared to the ARIMA model in both training and testing dataset. It has also been seen from Table 5 that the forecasted values of the SVM are closer to the observed values compared to ARIMA. From the above results and discussion, it can be inferred that performance of the SVM model is better than the ARIMA model in terms of forecasting accuracy.

6. Conclusion

In reality, most of the time series data are non-linear in nature. In this study, the data of cotton production show non-stationary as well as non-linearity structure which were difficult to capture for the ARIMA models. However, SVM has shown its' tremendous performance due to the ability of capturing the non-linear pattern. Being a non-linear machine learning technique, SVM has well captured the heterogeneous trend of the given dataset. Based on the results, it can be inferred that SVM outperformed the ARIMA model. Therefore, it can be used in modeling and forecasting of time series to improve the forecasting accuracy in the presence of non-linear pattern.

7. Bibliography

- Cortes, C. and Vapnik, V. (1995). Support-vector network. *Machine Learning*, 20, 1-25.
- David, M. (2017). E1071: Misc Functions of the Department of Statistics. *Probability Theory Group R package version*, 1: 6–8.
- De Giorgi, M.G., Campilongo, S., Ficarella, A. and Congedo, P.M. (2014). Comparison between wind power prediction models based on wavelet decomposition with least-squares support vector machine (LS-SVM) and artificial neural network (ANN). *Energies*, 7:5251-5272.
- Kumar, T.L.M. and Prajneshu (2015). Development of Hybrid Models for Forecasting Time-Series Data Using Nonlinear SVR Enhanced by PSO. *Journal of Statistical Theory and Practice*, 9(4), 699-711.
- Niu, D., Wang, Y. and Wu, D.D. (2010). Power load forecasting using support vector machine and ant colony optimization. *Expert Syst Appl*, 37:2531–2539.
- Ortiz-Garcia, E.G., Salcedo-Sanz, S. and Casanova-Mateom, C. (2014). Accurate precipitation prediction with support vector classifiers: A study including novel predictive variables and observational data. *Atmos Res*, 139:128–136.
- Vapnik, V., Golowich, S., and Smola, A. (1997). Support vector method for function approximation, regression estimation, and signal processing, In Mozer, M., Jordan, M and Petsche, T. (Eds). *Advances in Neural Information Processing Systems*, 9:281-287, Cambridge, MA, MIT Press.

RANDOM FOREST USING SCIKIT-LEARN LIBRARY

Upendra Kumar Pradhan & Sanchita Naha

ICAR-Indian Agricultural Statistics Research Institute, New Delhi - 110 012

upendra.pradhan@icar.gov.in and sanchita.naha@icar.gov.in

1. Introduction

Machine learning can be extensively depicted as computational strategies utilizing previous experience to improve performance or to make precise inferences. Here, past experience alludes to the past data accessible to the learner. This data could be as digitized human labelled training sets or other types of information obtained via interaction with the environment. In all cases, its quality and size are essential to the accomplishment of the expectations made by the learner. The processes involved in machine learning are similar to that of data mining and predictive modelling. Both require searching through data to look for patterns and adjusting program actions accordingly. Machine learning algorithms are often categorized as supervised or unsupervised. Supervised algorithms require both input and desired output, in addition to furnishing feedback about the accuracy of predictions during algorithm training. Once training is complete, the algorithm will apply what was learned to new data. Unsupervised algorithms do not need to be trained with desired outcome data. Since the success of a learning algorithm depends on the data used, machine learning is inherently related to data analysis and statistics. More generally, machine learning techniques are data-driven methods combining fundamental concepts in computer science with ideas from statistics, probability and optimization. The standard machine learning tasks which have been extensively studied are classification, Regression, Ranking, clustering and dimensionality reduction.

Recently there has been a lot of interest in “ensemble learning” — methods that generate many classifiers and aggregate their results. Two well-known methods are boosting (see, e.g., Shapire et al., 1998) and bagging Breiman (1996) of classification trees. In boosting, successive trees give extra weight to points incorrectly predicted by earlier predictors. In the end, a weighted vote is taken for prediction. In bagging, successive trees do not depend on earlier trees — each is independently constructed using a bootstrap sample of the data set. In the end, a simple majority vote is taken for prediction. Breiman (2001) proposed random forests, which add an additional layer of randomness to bagging. In addition to constructing each tree using a different bootstrap sample of the data, random forests change how the classification or regression trees are constructed. In standard trees, each node is split using the best split among all variables. In a random forest, each node is split using the best among a subset of predictors randomly chosen at that node. This somewhat counterintuitive strategy turns out to perform very well compared to many other classifiers, including discriminant analysis, support vector machines and neural networks, and is robust against overfitting (Breiman, 2001). In addition, it is very user-friendly in the sense that it has only two parameters (the number of variables in the random subset at each

node and the number of trees in the forest), and is usually not very sensitive to their values.

2. The Random Forest algorithm

The random forests algorithm (for both classification and regression) is as follows:

1. Draw n_{tree} bootstrap samples from the original data.
2. For each of the bootstrap samples, grow an *unpruned* classification or regression tree, with the following modification: at each node, rather than choosing the best split among all predictors, randomly sample m_{try} of the predictors and choose the best split from among those variables. (Bagging can be thought of as the special case of random forests obtained when $m_{\text{try}} = p$, the number of predictors.)
3. Predict new data by aggregating the predictions of the n_{tree} trees (i.e., majority votes for classification, average for regression).

An estimate of the error rate can be obtained, based on the training data, by the following:

1. At each bootstrap iteration, predict the data not in the bootstrap sample (what Breiman calls “out-of-bag”, or OOB, data) using the tree grown with the bootstrap sample.
2. Aggregate the OOB predictions (On the average, each data point would be out-of-bag around 36% of the times, so aggregate these predictions). Calculate the error rate, and call it the OOB estimate of error rate.

Our experience has been that the OOB estimate of error rate is quite accurate, given that enough trees have been grown (otherwise the OOB estimate can bias upward; see Bylander (2002)).

The Random Forest optionally produces two additional pieces of information: a measure of the importance of the predictor variables, and a measure of the internal structure of the data (the proximity of different data points to one another).

Variable importance: This is a difficult concept to define in general, because the importance of a variable may be due to its (possibly complex) interaction with other variables. The random forest algorithm estimates the importance of a variable by looking at how much prediction error increases when (OOB) data for that variable is permuted while all others are left unchanged. The necessary calculations are carried out tree by tree as the random forest is constructed.

proximity measure: The (i, j) element of the proximity matrix produced by Random Forest algorithm is the fraction of trees in which elements i and j fall in the same terminal node. The intuition is that “similar” observations should be in the same terminal nodes more often than dissimilar ones. The proximity matrix can be used to identify structure in the data (see Breiman, 2002).

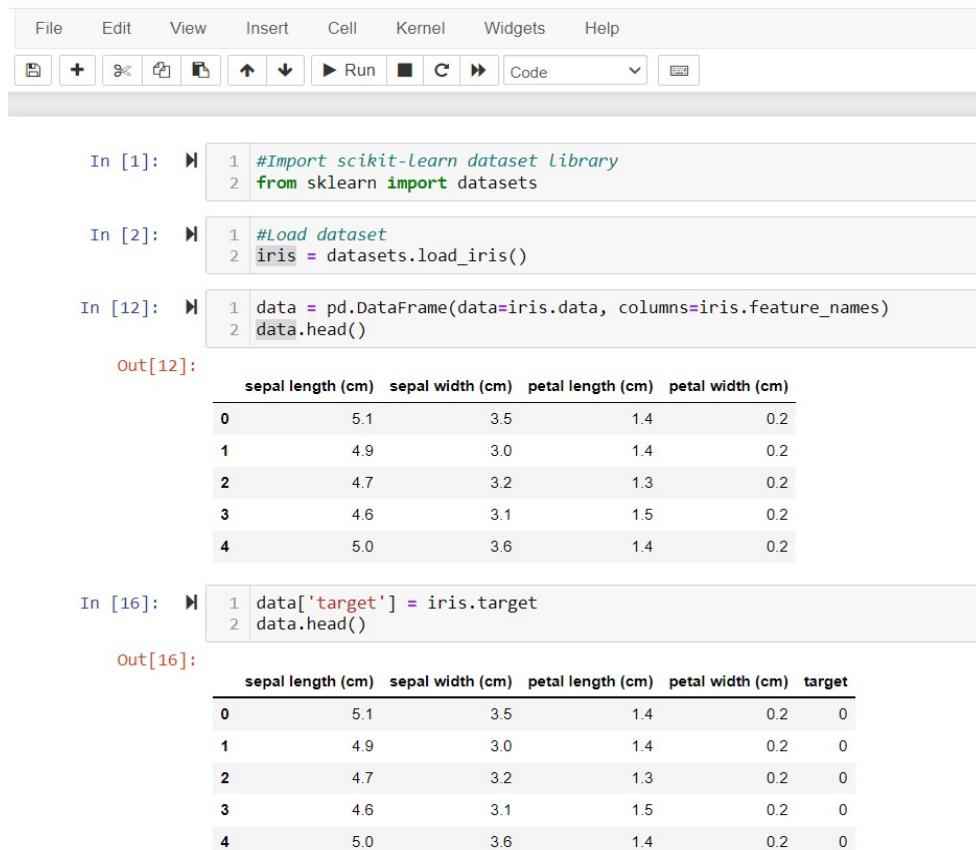
3. Random Forest using Scikit-Learn library

Random Forest ensembles can be implemented from scratch, although this can be challenging for beginners. The scikit-learn Python machine learning library provides an implementation of Random Forest for machine learning. Random Forest is provided via the *RandomForestRegressor* and *RandomForestClassifier* classes. Both models operate the same way and take the same arguments that influence how the decision trees are created. Randomness is used in the construction of the model. This means that each time the algorithm is run on the same data, it will produce a slightly different model. When using machine learning algorithms that have a stochastic learning algorithm, it is good practice to evaluate them by averaging their performance across multiple runs or repeats of cross-validation. When fitting a final model, it may be desirable to either increase the number of trees until the variance of the model is reduced across repeated evaluations, or to fit multiple final models and average their predictions. Let's take a look at how to develop a Random Forest ensemble for both classification tasks.

Building a Random Forest Classifier using Scikit-learn

We will be using the IRIS flower dataset for this tutorial. It comprises the independent attributes sepal length, sepal width,

 jupyter Random_Forest_iris Last Checkpoint: 29 minutes ago (unsaved changes)



The screenshot shows a Jupyter Notebook with the following code and outputs:

```
In [1]: 1 #Import scikit-learn dataset Library
        2 from sklearn import datasets

In [2]: 1 #Load dataset
        2 iris = datasets.load_iris()

In [12]: 1 data = pd.DataFrame(data=iris.data, columns=iris.feature_names)
         2 data.head()

Out[12]:
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
0                5.1                3.5                1.4                0.2
1                4.9                3.0                1.4                0.2
2                4.7                3.2                1.3                0.2
3                4.6                3.1                1.5                0.2
4                5.0                3.6                1.4                0.2

In [16]: 1 data['target'] = iris.target
         2 data.head()

Out[16]:
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  target
0                5.1                3.5                1.4                0.2        0
1                4.9                3.0                1.4                0.2        0
2                4.7                3.2                1.3                0.2        0
3                4.6                3.1                1.5                0.2        0
4                5.0                3.6                1.4                0.2        0
```

petal length, petal width, and type of flowers. There are three species or classes: setosa, versicolor and virginia. We will be developing a RF classifier to classify the

flowers among any of the three groups based on the independent data values. The dataset is available in the scikit-learn library.

Start by importing the datasets library from scikit-learn package, load the iris dataset and convert it into a *pandas DataFrame* object called 'data'.

In the mentioned dataset target values are encoded as 0: setosa, 1: versicolor, 2: virginica. Next, separate the columns into dependent and independent variables (or Y and X respectively).

Then

split the entire dataset into training and test set using the *train_test_split()* method from *sklearn.model_selection*.

```

1 # Import train_test_split function
2 from sklearn.model_selection import train_test_split
3
4 X=data[['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']] # Features
5 y=data['target'] # Labels
6
7 # Split dataset into training set and test set
8 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3) # 70% training and 30% test
9 print(X_train.columns)
10
Index(['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)',
      'petal width (cm)'],
      dtype='object')

```

After splitting, train the RF model on the training set and perform predictions using the test set.

```

1 #Import Random Forest Model
2 from sklearn.ensemble import RandomForestClassifier
3
4 #Create a Gaussian Classifier
5 rf_model = RandomForestClassifier(n_estimators=100)
6
7 #Train the model using the training sets y_pred=clf.predict(X_test)
8 rf_model.fit(X_train,y_train)
9
10 y_pred=rf_model.predict(X_test)

```

After training, we can print the accuracy of the model based on actual and predicted values as follows.

```

1 #Import scikit-Learn metrics module for accuracy calculation
2 from sklearn import metrics
3 # Model Accuracy
4 print("Accuracy:",metrics.accuracy_score(y_test, y_pred))

```

Accuracy: 0.9555555555555556

We can also print the classification report for the trained model using the *classification_report*

```

1 from sklearn.metrics import classification_report
2 print(classification_report(y_test, y_pred))

```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	19
1	0.93	0.93	0.93	14
2	0.92	0.92	0.92	12
accuracy			0.96	45
macro avg	0.95	0.95	0.95	45
weighted avg	0.96	0.96	0.96	45

Module of scikit-learn. Classification report is a performance evaluation metric in machine learning. It shows the precision, recall, F1 Score, and support of the trained classifier.

References

- L. Breiman . Bagging predictors. *Machine Learning*, 24 (2):123–140, 1996.
- L. Breiman. Random forests. *Machine Learning*, 45(1): 5–32, 2001.
- L. Breiman. Manual on setting up, using, and understanding random forests v3.1, 2002. http://oz.berkeley.edu/users/breiman/Using_random_forests_V3.1.pdf.
- T. Bylander. Estimating generalization error on two class datasets using out-of-bag estimates. *Machine Learning*, 48:287–297, 2002
- R. Shapire, Y. Freund, P. Bartlett, and W. Lee. Boosting the margin: A new explanation for the effectiveness of voting methods. *Annals of Statistics*, 26 (5):1651–1686, 1998
- W. N. Venables and B. D. Ripley. Modern Applied Statistics in S. *Springer*, 4th edition, 2002.

Hands on Artificial Neural Network using Scikit-learn and Keras

Chandan Kumar Deb

ICAR-Indian Agricultural Statistics Research Institute, New Delhi - 110 012

chandan.deb@icar.gov.in

Artificial Neural Network using Scikit-learn

Step 1 - Loading the Required Libraries and Modules

```
# Import required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import sklearn
from sklearn.neural_network import MLPClassifier
from sklearn.neural_network import MLPRegressor

# Import necessary modules
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from math import sqrt
from sklearn.metrics import r2_score
```

Step 2 - Reading the Data and Performing Basic Data Checks

```
df = pd.read_csv('/content/sample_data/diabetes.csv')
print(df.shape)
df.describe().transpose()
```

	count	mean	std	min	25%	50%	75%	max
Pregnancies	768.0	3.845052	3.369578	0.000	1.00000	3.0000	6.00000	17.00
Glucose	768.0	120.894531	31.972618	0.000	99.00000	117.0000	140.25000	199.00
BloodPressure	768.0	69.105469	19.355807	0.000	62.00000	72.0000	80.00000	122.00
SkinThickness	768.0	20.536458	15.952218	0.000	0.00000	23.0000	32.00000	99.00
Insulin	768.0	79.799479	115.244002	0.000	0.00000	30.5000	127.25000	846.00
BMI	768.0	31.992578	7.884160	0.000	27.30000	32.0000	36.60000	67.10
DiabetesPedigreeFunction	768.0	0.471876	0.331329	0.078	0.24375	0.3725	0.62625	2.42
Age	768.0	33.240885	11.760232	21.000	24.00000	29.0000	41.00000	81.00
Outcome	768.0	0.348958	0.476951	0.000	0.00000	0.0000	1.00000	1.00

Figure 1: output of df.describe().transpose()

```
df.head(10)
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1
5	5	116	74	0	0	25.6	0.201	30	0
6	3	78	50	32	88	31.0	0.248	26	1
7	10	115	0	0	0	35.3	0.134	29	0
8	2	197	70	45	543	30.5	0.158	53	1
9	8	125	96	0	0	0.0	0.232	54	1

Figure 2: output of df.head(10)

Step 3 - Creating Arrays for the Features and the Response Variable

```
target_column = ['Outcome']
predictors = list(set(list(df.columns))-set(target_column))
df[predictors] = df[predictors]/df[predictors].max()
df.describe().transpose()
```

Step 4 - Creating the Training and Test Datasets

```
X = df[predictors].values
y = df[target_column].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30,
random_state=40)
```

Step 5 - Building, Predicting, and Evaluating the Neural Network Model

```
from sklearn.neural_network import MLPClassifier
mlp = MLPClassifier(hidden_layer_sizes=(8,8,8), activation='relu', solver=
'adam', max_iter=500)
mlp.fit(X_train,y_train)
predict_train = mlp.predict(X_train)
predict_test = mlp.predict(X_test)
```

```
from sklearn.metrics import classification_report,confusion_matrix
print(classification_report(y_train,predict_train))
```

	precision	recall	f1-score	support
0	0.81	0.90	0.85	358
1	0.74	0.56	0.64	179
accuracy			0.79	537
macro avg	0.77	0.73	0.75	537
weighted avg	0.78	0.79	0.78	537

Figure 3: Output of classification report

Artificial Neural Network using Keras

```
from numpy import loadtxt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
```

Define the keras model

```
model = Sequential()
model.add(Dense(12, input_shape=(8,), activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

Compile the keras model

```
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Fit the keras model on the dataset

```
model.fit(X, y, epochs=50, batch_size=10)
```

Evaluate the keras model

```
_, accuracy = model.evaluate(X, y)
print('Accuracy: %.2f' % (accuracy*100))
```

Handwritten Digit Recognition on MNIST dataset bold text bold text

```
import tensorflow as tf
from tensorflow import keras
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np
(X_train, y_train), (X_test, y_test) = keras.datasets.mnist.load_data()
```

Compile the keras model

```
model = keras.Sequential([
    keras.layers.Dense(10, input_shape=(784,)), activation='sigmoid'
])
```

```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

```
model.fit(X_train_flattened, y_train, epochs=5)
```

Recurrent Neural Networks using Keras:

Ease of use: the built-in `keras.layers.RNN`, `keras.layers.LSTM`, `keras.layers.GRU` layers enable you to quickly build recurrent models without having to make difficult configuration choices.

Ease of customization: You can also define your own RNN cell layer (the inner part of the for loop) with custom behavior, and use it with the generic `keras.layers.RNN` layer (the for loop itself). This allows you to quickly prototype different research ideas in a flexible way with minimal code.

Steps in implementing RNN using keras

- Read the dataset from a given URL
- Split the data into training and test sets
- Prepare the input to the required Keras format
- Create an RNN model and train it
- Make the predictions on training and test sets and print the root mean square error on both sets
- View the result

Step 1, 2: Reading Data and Splitting Into Train and Test

The following function reads the train and test data from a given URL and splits it into a given percentage of train and test data. It returns single-dimensional arrays for train and test data after scaling the data between 0 and 1 using `MinMaxScaler` from `scikit-learn`.

```
from pandas import read_csv
import numpy as np
from keras.models import Sequential
from keras.layers import Dense, SimpleRNN
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
import math
import matplotlib.pyplot as plt
def get_train_test(url, split_percent=0.8):
    df = read_csv(url, usecols=[1], engine='python')
    data = np.array(df.values.astype('float32'))
    scaler = MinMaxScaler(feature_range=(0, 1))
    data = scaler.fit_transform(data).flatten()
    n = len(data)
    # Point for splitting data into train and test
    split = int(n*split_percent)
    train_data = data[range(split)]
    test_data = data[split:]
    return train_data, test_data, data
```


Step 3: Reshaping Data for Keras

```
# Prepare the input X and target Y
def get_XY(dat, time_steps):
    Y_ind = np.arange(time_steps, len(dat), time_steps)
    Y = dat[Y_ind]
    rows_x = len(Y)
    X = dat[range(time_steps*rows_x)]
    X = np.reshape(X, (rows_x, time_steps, 1))
    return X, Y
```

Step 4: Create RNN Model and Train

```
def create_RNN(hidden_units, dense_units, input_shape, activation):
    model = Sequential()
    model.add(SimpleRNN(hidden_units, input_shape=input_shape, activation=activation[0]))
    model.add(Dense(units=dense_units, activation=activation[1]))
    model.compile(loss='mean_squared_error', optimizer='adam')
    return model
```

Step 5: Compute and Print the Root Mean Square Error

```
def print_error(trainY, testY, train_predict, test_predict):
    # Error of predictions
    train_rmse = math.sqrt(mean_squared_error(trainY, train_predict))
    test_rmse = math.sqrt(mean_squared_error(testY, test_predict))
    # Print RMSE
    print('Train RMSE: %.3f RMSE' % (train_rmse))
    print('Test RMSE: %.3f RMSE' % (test_rmse))
```

Step 6: View the Result

```
# Plot the result
def plot_result(trainY, testY, train_predict, test_predict):
    actual = np.append(trainY, testY)
    predictions = np.append(train_predict, test_predict)
    rows = len(actual)
    plt.figure(figsize=(15, 6), dpi=80)
    plt.plot(range(rows), actual)
    plt.plot(range(rows), predictions)
    plt.axvline(x=len(trainY), color='r')
    plt.legend(['Actual', 'Predictions'])
    plt.xlabel('Observation number after given time steps')
    plt.ylabel('Sunspots scaled')
    plt.title('Actual and Predicted Values. The Red Line Separates The Training And Test Examples')
```

Call required functions for implementing RNN

```
sunspots_url = 'https://raw.githubusercontent.com/jbrownlee/Datasets/master/monthly-sunspots.csv'
time_steps = 12
train_data, test_data, data = get_train_test(sunspots_url)
```

```
trainX, trainY = get_XY(train_data, time_steps)
testX, testY = get_XY(test_data, time_steps)

# Create model and train
model = create_RNN(hidden_units=3, dense_units=1, input_shape=(time_steps,
1),
                    activation=['tanh', 'tanh'])
model.fit(trainX, trainY, epochs=20, batch_size=1, verbose=2)

# make predictions
train_predict = model.predict(trainX)
test_predict = model.predict(testX)

# Print error
print_error(trainY, testY, train_predict, test_predict)

#Plot result
plot_result(trainY, testY, train_predict, test_predict)
```

Concepts of Artificial Neural Network: Perceptron, RNN, LSTM

Anshu Bharadwaj

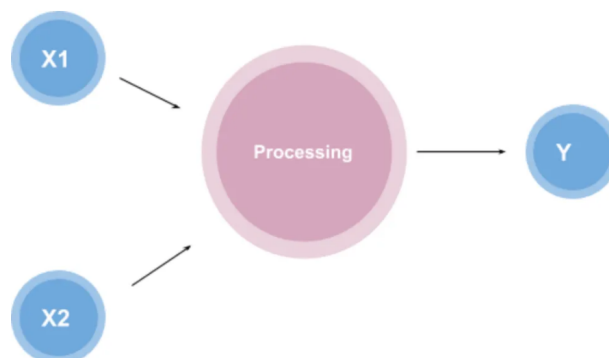
ICAR-Indian Agricultural Statistics Research Institute, New Delhi - 110 012

anshu.bharadwaj@icar.gov.in

What is a Perceptron?

A Perceptron is an algorithm used for supervised learning of binary classifiers. Binary classifiers decide whether an input, usually represented by a series of vectors, belongs to a specific class. In short, a perceptron is a single-layer neural network. They consist of four main parts including input values, weights and bias, net sum, and an activation function. A perceptron model, in Machine Learning, is a supervised learning algorithm of binary classifiers. A single neuron, the perceptron model detects whether any function is an input or not and classifies them in either of the classes. Representing a biological neuron in the human brain, the perceptron model or simply a perceptron acts as an artificial neuron that performs human-like brain functions. A linear ML algorithm, the perceptron conducts binary classification or two-class categorization and enables neurons to learn and register information procured from the inputs. This model uses a hyperplane line that classifies two inputs and classifies them on the basis of the 2 classes that a machine learns, thus implying that the perceptron model is a linear classification model. Invented by Frank Rosenblatt in 1957, the perceptron model is a vital element of Machine Learning as ML is recognized for its classification purposes and mechanism. The perceptron was first introduced by American psychologist, Frank Rosenblatt in 1957 at Cornell Aeronautical Laboratory and is a vital element of Machine Learning as ML is recognized for its classification purposes and mechanism.

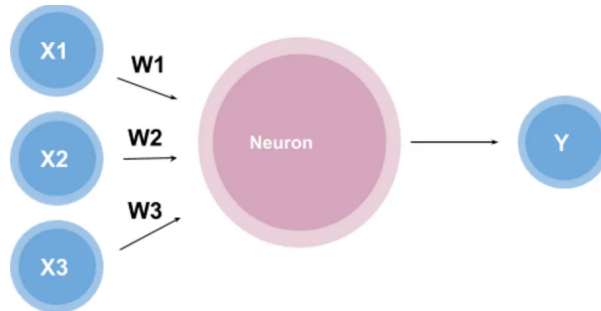
Rosenblatt was heavily inspired by the biological neuron and its ability to learn. Rosenblatt's perceptron consists of one or more inputs, a processor, and only one output.



A perceptron works by taking in some numerical inputs along with what is known as weights and a bias. It then multiplies these inputs with the respective weights (this is known as the weighted sum). These products are then added together along with the bias. The activation function takes the weighted sum and the bias as inputs and returns

a final output. A perceptron consists of four parts: input values, weights and a bias, a weighted sum, and activation function.

Assume we have a single **neuron** and three inputs **x1**, **x2**, **x3** multiplied by the weights **w1**, **w2**, **w3** respectively as shown below,



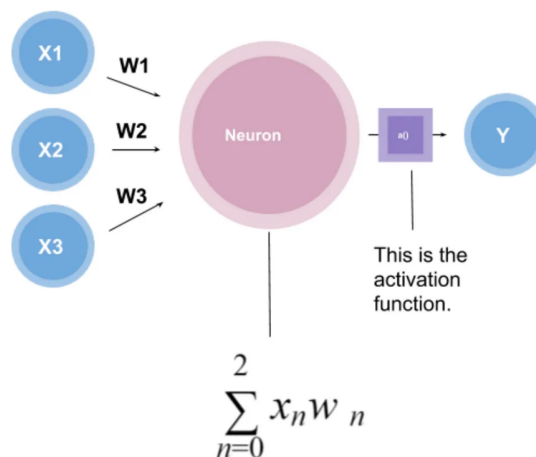
The idea is simple, given the numerical value of the inputs and the weights, there is a function, inside the neuron, that will produce an output. The question now is, what is this function?

One function may look like

$$y = x_1 w_1 + x_2 w_2 + x_3 w_3$$

This function is called the weighted sum because it is the sum of the weights and inputs. This looks like a good function, but what if we wanted the outputs to fall into a certain range say 0 to 1.

We can do this by using something known as an activation function. An activation function is a function that converts the input given (the input, in this case, would be the weighted sum) into a certain output based on a set of rules.



There are different kinds of activation functions that exist, for example:

1. **Hyperbolic Tangent:** used to output a number from -1 to 1.
2. **Logistic Function:** used to output a number from 0 to 1.

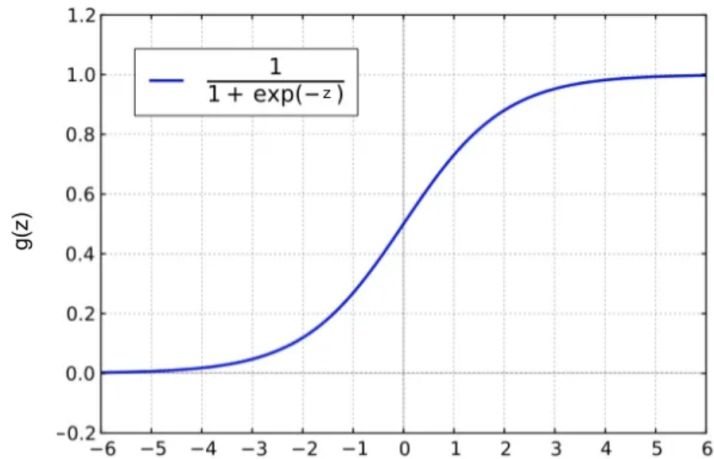
3. Since the range we are looking for is between 0 and 1, we will be using a Logistic Function to achieve this.

4. **Logistic Functions**

5. Logistical functions have the formula,

$$g(z) = \frac{1}{1 + e^{-z}}$$

Where the graph looks like,



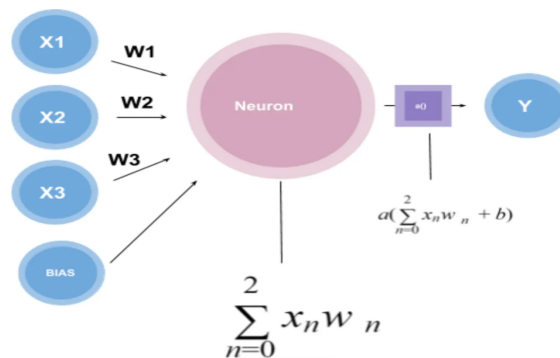
Notice that $g(z)$ lies between the points 0 and 1 and that this graph is not linear. This will allow us to output numbers that are between 0 and 1 which is exactly what we need to build our perceptron.

Now we have *almost* everything we need to make our perceptron. The last thing we are missing is the bias. The bias is a threshold the perceptron must reach before the output is produced. So the final neuron equation looks like:

Represented visually we see (where typically the bias is represented near the inputs),

$$Y = \sum(\text{weight} * \text{input}) + \text{bias}$$

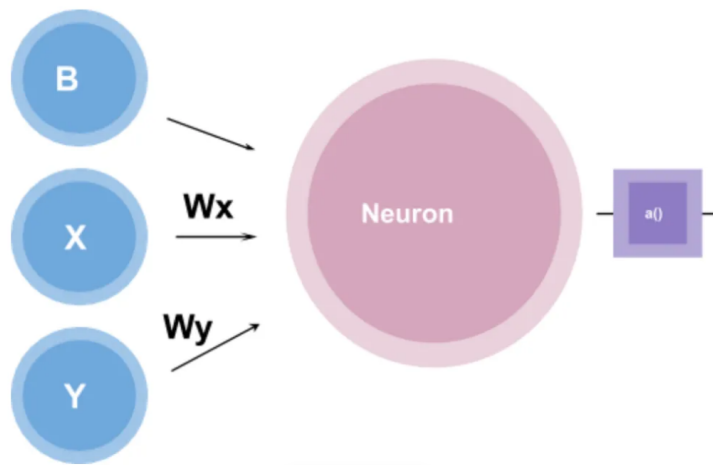
Represented visually we see (where typically the bias is represented near the inputs),



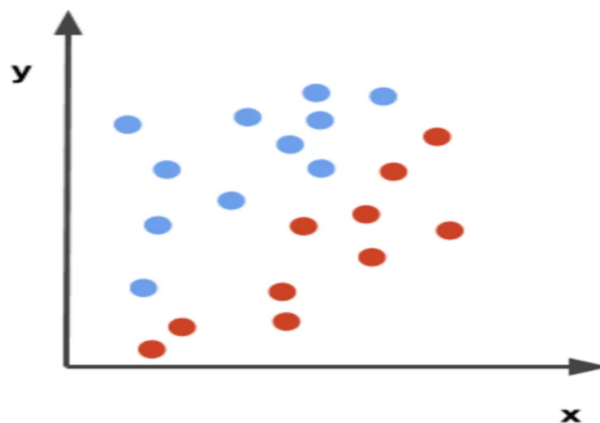
Notice that the activation function takes in the weighted sum plus the bias as inputs to create a single output. Using the Logistical Function this output will be between 0 and 1.

Why are perceptron's used?

Perceptrons are the building blocks of neural networks. It is typically used for supervised learning of binary classifiers. This is best explained through an example. Let's take a simple perceptron. In this perceptron we have an input x and y , which is multiplied with the weights w_x and w_y respectively, it also contains a bias.



Let's also create a graph with two different categories of data represented with red and blue dots.



Notice that the x-axis is labeled after the input x and the y-axis is labeled after the input y .

Suppose the goal was to separates this data so that there is a distinction between the blue dots and the red dots. How can we use the perceptron to do this?

A perceptron can create a decision boundary for a binary classification, where a decision boundary is regions of space on a graph that separates different data points.

Let's play with the function to better understand this. We can say,

$$w_x = -0.5$$

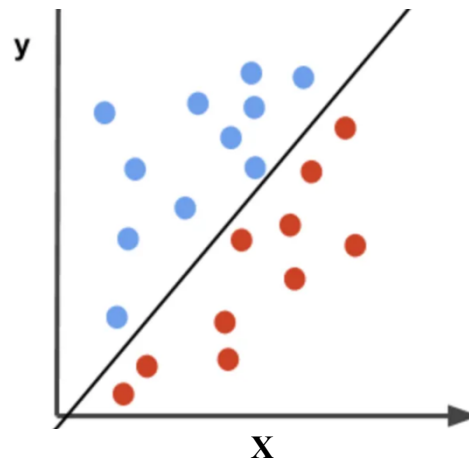
$$wy = 0.5$$

$$\text{and } b = 0$$

Then the function for the perceptron will look like,

$$0.5x + 0.5y = 0$$

and the graph will look like,

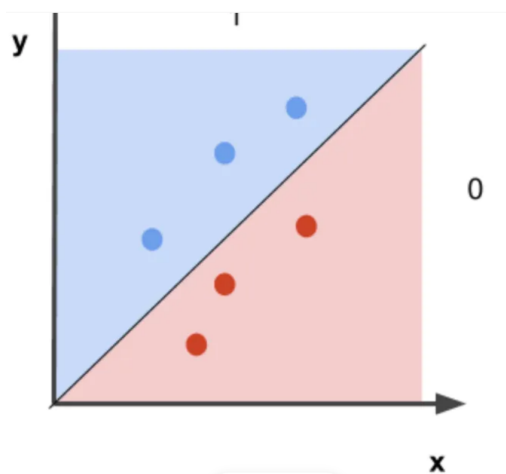


Let's suppose that the activation function, in this case, is a simple step function that outputs *either* 0 or 1. The perceptron function will then label the blue dots as 1 and the red dots as 0. In other words,

$$\text{if } 0.5x + 0.5y \geq 0, \text{ then } 1$$

$$\text{if } 0.5x + 0.5y < 0, \text{ then } 0.$$

Therefore, the function $0.5x + 0.5y = 0$ creates a decision boundary that separates the red and blue points.

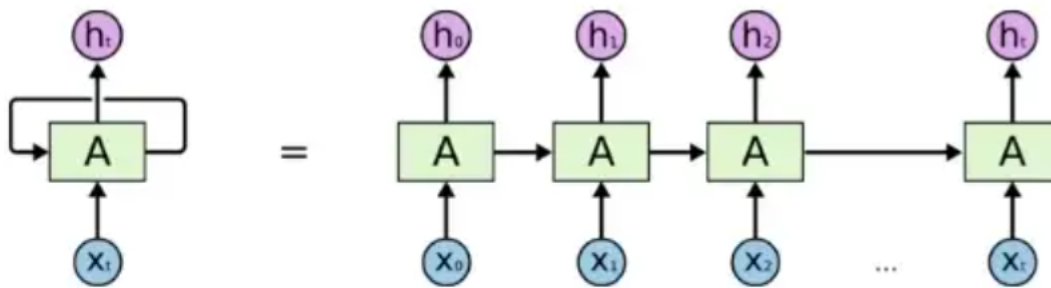


Overall, we see that a perceptron can do basic classification using a decision boundary.

What is Recurrent Neural Network (RNN)?

Recurrent Neural Network is a generalization of feedforward neural network that has an internal memory. RNN is recurrent in nature as it performs the same function for every input of data while the output of the current input depends on the past one computation. After producing the output, it is copied and sent back into the recurrent network. For making a decision, it considers the current input and the output that it has learned from the previous input.

Unlike feedforward neural networks, RNNs can use their internal state (memory) to process sequences of inputs. This makes them applicable to tasks such as unsegmented, connected handwriting recognition or speech recognition. In other neural networks, all the inputs are independent of each other. But in RNN, all the inputs are related to each other.



An unrolled recurrent neural network.

First, it takes the $X(0)$ from the sequence of input and then it outputs $h(0)$ which together with $X(1)$ is the input for the next step. So, the $h(0)$ and $X(1)$ is the input for the next step. Similarly, $h(1)$ from the next is the input with $X(2)$ for the next step and so on. This way, it keeps remembering the context while training.

The formula for the current state is

$$h_t = f(h_{t-1}, x_t)$$

Applying Activation Function:

$$h_t = \tanh (W_{hh}h_{t-1} + W_{xh}x_t)$$

W is weight, h is the single hidden vector, W_{hh} is the weight at previous hidden state, W_{hx} is the weight at current input state, \tanh is the Activation function, that implements a Non-linearity that squashes the activations to the range[-1.1]

Output:

$$y_t = W_{hy}h_t$$

Y_t is the output state. W_y is the weight at the output state.

More about RNNs: The input and output of standard ANNs are interdependent. However, the output of an RNN is reliant on the previous nodes in the sequence. Each neuron in a feed-forward network or multi-layer perceptron executes its function with inputs and feeds the result to the next node.

RNN works on the principle of saving the output of a particular layer and feeding this back to the input in order to predict the output of the layer.

Below is how you can convert a Feed-Forward Neural Network into a Recurrent Neural Network:

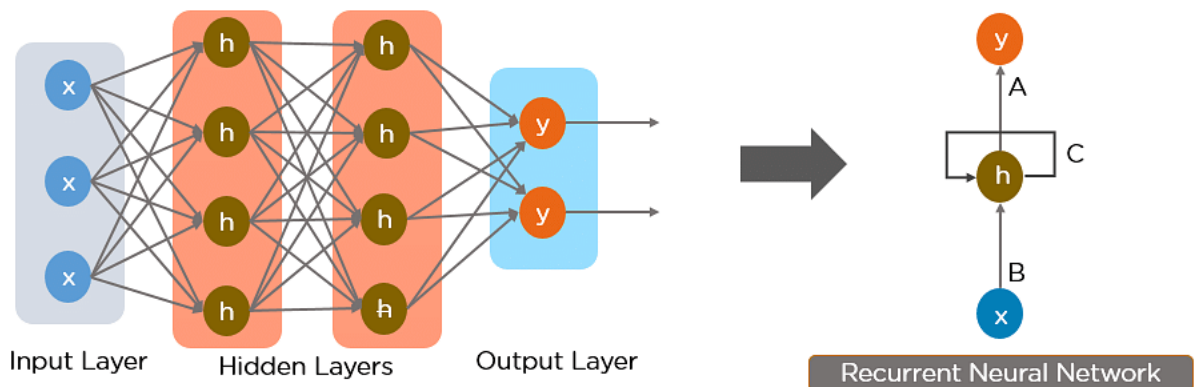


Fig: Simple Recurrent Neural Network

As the name implies, recurrent neural networks have a recurrent connection in which the output is transmitted back to the RNN neuron rather than only passing it to the next node. Each node in the RNN model functions as a memory cell, continuing calculation and operation implementation.

If the network's forecast is inaccurate, the system self-learns and performs backpropagation toward the correct prediction.

An RNN remembers every piece of information throughout time. It is only effective in time series prediction because of the ability to recall past inputs. This is referred to as long short-term memory. Recurrent neural networks combine with convolutional layers to widen the effective pixel neighborhood.

Types of RNN

RNNs are categorized based on the four network sequences, namely,

- One to One Network
- One to Many Network
- Many to One Network
- Many to Many Network

RNN: One to One Model

The one-to-one RNN is a typical sequence in neural networks, with only one input and one output. Application – Image classification

RNN: One to Many Model

One to Many network has a single input feed into the node, producing multiple outputs. Application – Music generation, image captioning, etc.

RNN: Many to One model

Many to One architecture of RNN is utilized when there are several inputs for generating a single output. Application – Sentiment analysis, rating model, etc.

RNN: Many to Many Model

Many to Many RNN models, as the name implies, have multiple inputs and produce multiple outputs. This model is also incorporated where input and output layer sizes are different. Application – Machine translation.

Advantages of Recurrent Neural Network

- RNN can model sequence of data so that each sample can be assumed to be dependent on previous ones.
- Recurrent neural networks are even used with convolutional layers to extend the effective pixel neighbourhood.

Disadvantages of Recurrent Neural Network

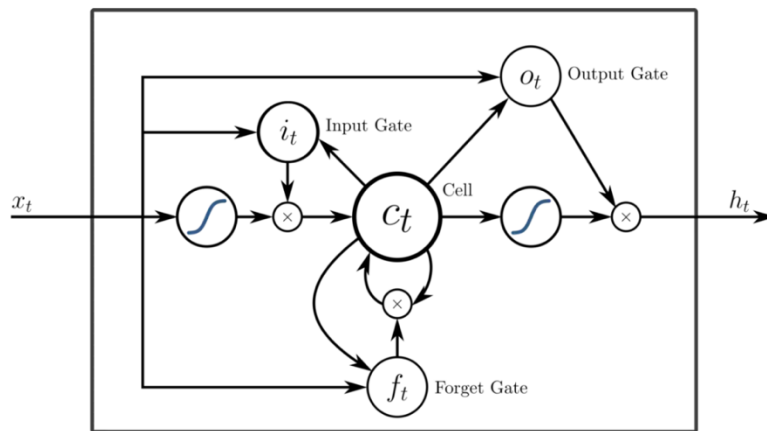
- Gradient vanishing and exploding problems.
- Training an RNN is a very difficult task.
- It cannot process very long sequences if using *tanh* or *relu* as an activation function

In a nutshell, RNN is defined as a neural network with some internal state updated at each step. Hidden states are employed to use prior information during output sequence prediction. Its applications include speech recognition, language modeling, machine translation, and the development of chatbots.

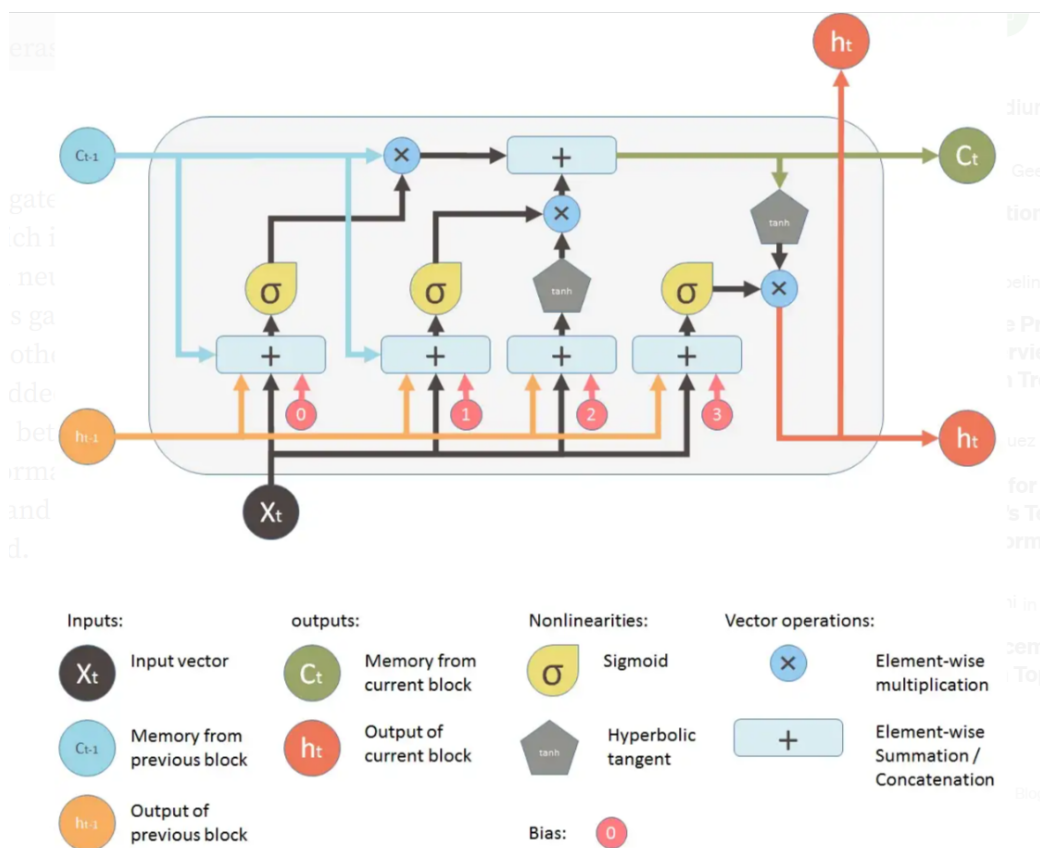
Long Short Term Memory (LSTM)

As we discussed RNN are not able to memorize data for long time and begins to forget its previous inputs. To overcome this problem of vanishing and exploding gradient LSTM is used. They are used as solution for short term memory learning. Also in RNN when a new information is added RNN completely modifies the existing information. RNN is not able to distinguish between important or not so important information.

Whereas in LSTM there is small modification in existing information when a new information is added because LSTM contains gate which determine the flow of information.

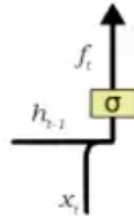


The gates decide which data is important and can be useful in future and which data has to be erased. The three gates are input gate, output gate and forget gate.

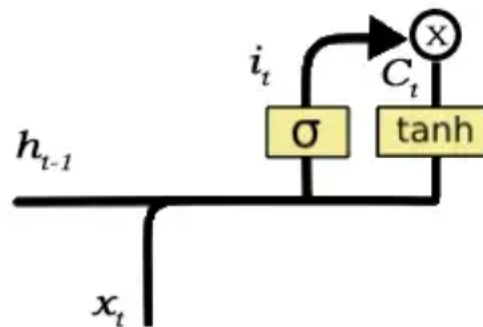


- Forget Gate:** This gate decide which information is important and should be stored and which information to forget. It removes the non important information from neuron cell. This results in optimization of performance. This gate takes 2 input- one is the output generated by previous cell and other is input of current cell. Following required bias and weights are

added and multiplied and sigmoid function is applied to the value. A value between 0 and 1 is generated and based on this we decide which information to keep. If value is 0 the forget gate will remove that information and if value is 1 then information is important and has to be remembered.



- **Input Gate:** This gate is used to add information to neuron cell. It is responsible of what values should be added to cell by using activation function like sigmoid. It creates an array of information that has to be added. This is done by using another activation function called tanh. It generates a value between -1 and 1. The sigmoid function act as a filter and regulate what information has to be added in cell.



- **Output Gate:** This gate is responsible for selecting important information from current cell and show it as output. It creates a vector of values using tanh function which ranges from -1 to 1. It uses previous output and current input as a regulator which also includes sigmoid function and decides which values should be shown as output.

Squashing / Activation Functions in LSTM

1. Logistic (sigmoid): Outputs range from 0 to 1
2. Hyperbolic Tangent (tanh): Outputs range from -1 to 1.

More about Long short-term memory (LSTM) in machine learning

Long Short-Term Memory (LSTM) networks are a modified version of recurrent neural networks, which makes it easier to remember past data in memory. The vanishing gradient problem of RNN is resolved here. LSTM is well-suited to classify,

process and predict time series given time lags of unknown duration. It trains the model by using back-propagation.

- LSTM is a type of RNN with higher memory power to remember the outputs of each node for a more extended period to produce the outcome for the next node efficiently.
- LSTM networks combat the RNN's vanishing gradients or long-term dependence issue.
- Gradient vanishing refers to the loss of information in a neural network as connections recur over a longer period.
- In simple words, LSTM tackles gradient vanishing by ignoring useless data/information in the network.
- For example, if an RNN is asked to predict the following word in this phrase, "have a pleasant _____," it will readily anticipate "day."
- The input data is very limited in this case, and there are only a few possible output results

Deep Learning and Convolutional Neural Networks

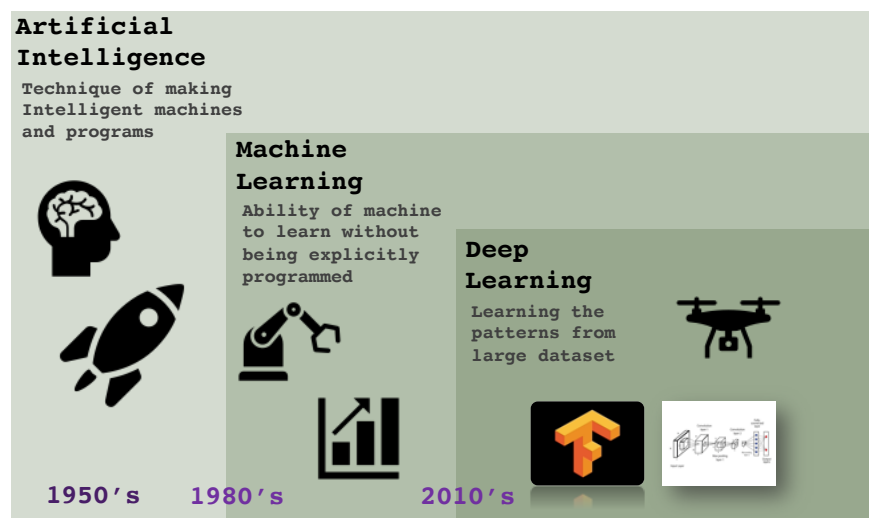
Md. Ashraful Haque and Akshay Dheeraj

ICAR-Indian Agricultural Statistics Research Institute, New Delhi - 110 012

ashraful.haque@icar.gov.in and akshay.dheeraj@icar.gov.in

Introduction:

Deep Learning (DL) is a particular kind of machine learning (subset of machine learning) that achieves great power and flexibility by learning to represent the world as a nested hierarchy of concepts. Deep learning is an important element of data science, which includes statistics and predictive modelling. Deep learning allows machines to learn patterns in an automated nature from a very large number of datasets. Recently, the deep learning technique have gained much popularity in the area of computer vision and neural machine translation. Fundamentally, artificial neural networks (ANNs) clubbed with multi-level representation learning forms the backbone of the deep learning concepts. Deep learning methods aim at learning feature hierarchies with features from higher levels of the hierarchy formed by the composition of lower level features (LeCun *et al.*, 2015). The idea of learning the right representation for the data provides one perspective on deep learning. Another perspective on deep learning is that depth allows the computer to learn a multi-layer network. Deep learning techniques consists of computational models that applies multiple processing layers to learn patterns of data with multiple levels of abstraction (LeCun *et al.*, 2015). Each layer of processing units can be used to extract progressively higher-level, abstract features from raw dataset under study in parallel with others. Automatically learning features at multiple levels of abstraction allow a machine to learn complex functions mapping the input to the output directly from data, without depending completely on human-crafted features. To achieve an acceptable level of performance, the deep learning techniques require access to immense amounts of training data and processing power.



Chronology of AI, ML and DL

Deep Learning techniques:

There are several types of deep learning techniques available now-a-days such as Multi-Layer Perceptron (MLPs), Convolutional Neural Networks (CNNs), Deep Recurrent Neural networks (RNNs) & Long Short-term Memory (LSTM) networks etc.

Multi-Layer Perceptron (MLP): MLPs are the most traditional types of deep learning architectures. Every element of a previous layer, is connected to every element of the next layer. Such layer is called dense layer or fully connected layer. Fell out of favor, in part because they are hard to train.

Convolution Neural Network (CNN): CNNs are the advanced type of feed forward neural network. The CNNs takes a fixed size inputs, process it using multiple processing units and generates fixed-size outputs. Mostly used in computer vision applications for object detection, classification and semantic segmentation. Generally, appropriate for image and video processing tasks.

Recurrent Neural Network (RNN): The type of feed forward neural networks extended to include feedback connections within. The network use its internal memory to process arbitrary sequence of inputs, hence can handle arbitrary input/output length. RNNs are mostly useful for time series data where features representing the past are assumed to have bearing on the future. Generally, ideal for text and speech analysis.

Deep learning frameworks:

Building the deep learning solution in largescale is quite difficult due to huge computational complexity of the deep learning models. There exists several deep learning frameworks available for building deep learning solutions. These frameworks offers a higher level of abstraction and simplify potentially complex network programming. The frameworks are:

- a. *TensorFlow:*** TensorFlow (also called as TensorFlow engine) is the most widely used platform for machine learning and deep learning applications. It is a free and open-source software library developed by Google team in 2015 for research and analysis in the field of AI and ML. It provides a collection of workflows with intuitive, high-level APIs for both beginners and experts to create machine learning models in numerous languages (Python/R/Java). It has a comprehensive, flexible ecosystem of tools, libraries, and community resources that lets researchers push the state-of-the-art in ML and developers easily build and deploy ML-powered real world applications.
- b. *Keras:*** Keras is the official high-level deep learning API for TensorFlow. It is developed by Google to make the implementation of TensorFlow simple and productive. Keras is open source, free to use and implement and written in python programming language. Keras is embedded in TensorFlow and can be used to perform deep learning fast as it provides inbuilt modules for all neural

network computations. Keras is relatively easy to learn and work with because it provides a python frontend with a high level of abstraction while having the option of multiple back-ends for computation purposes.

- c. **PyTorch:** PyTorch is an open source machine learning and deep learning framework developed by the Facebook AI Research (FAIR) team in 2017. PyTorch is based on the Python programming language and the Torch library. It has been designed to provide greater flexibility and increased speed for deep neural network implementation. Presently, PyTorch is the most favoured library for the ML and DL researchers and practitioners worldwide.

Deep learning network development flow:

A general framework of any deep learning network for any problem domain follows as series of steps as provided below:

- a. Problem statement formulation
- b. Collection and preparation of dataset
- c. Selection of a deep learning framework for network development
- d. Designing the architecture of the deep learning network
- e. Training the network
- f. Performance Evaluation of the model
- g. Saving the parameters and architecture of network in a binary files
- h. Network implementation

Benefits of Deep learning:

Deep learning techniques have several advantages over the traditional machine learning methods include:

- a. *Automatic feature learning:* Deep learning algorithms can automatically learn the features from the raw data, that means there is no requirement of hand-engineered feature extraction. This is particularly useful for tasks where the features are difficult to define, such as image recognition.
- b. *Handling large and complex data:* Deep learning techniques can efficiently handle large and complex datasets which would be very difficult for traditional machine learning models to process.
- c. *Handling non-linear relationships:* Deep learning can uncover non-linear relationships in data that would be difficult to detect through traditional methods.
- d. *Handling structured and unstructured data:* Deep learning algorithms can handle both structured and unstructured data such as images, text, and audio.

- e. *Handling missing data:* Deep learning algorithms can handle missing data and still make predictions, which is useful in real-world applications where data is often incomplete.
- f. *Handling sequential data:* Deep learning algorithms such as Recurrent Neural Networks (RNNs) and Long Short-term Memory (LSTM) networks are particularly suited to handle sequential data such as time series, speech, and text. These algorithms have the ability to maintain context and memory over time, which allows them to make predictions or decisions based on past inputs.
- g. *Scalability:* Deep learning models can be easily scaled to handle an increasing amount of data and can be deployed on cloud platforms and edge devices.
- h. *Improved generalization:* Deep learning models can generalize well to new situations or contexts, as they are able to learn abstract and hierarchical representations of the data.

Limitations of Deep learning:

In addition to the advantages, deep learning has some disadvantages too which are discussed as follows:

- a. *High computational cost:* Training deep learning models requires significant computational resources, including powerful GPUs and large amounts of memory. This can be costly and time-consuming.
- b. *Dependence on data quality:* Deep learning algorithms rely on the quality of the data they are trained on. If the data is noisy, incomplete, or biased, the model's performance will be negatively affected.
- c. *Dependency on data quantity:* Deep learning algorithms require a large quantity of data to be trained on. If the data is not sufficient, the model's performance will be affected.
- d. *Black box models:* some deep learning models are considered as "black-box" models, as it is difficult to understand how the model is making predictions and identifying the factors that influence the predictions.
- e. *Dependency on the domain expertise:* Deep learning requires a good understanding of the domain and the problem you are trying to solve. If the domain expertise is lacking, it can be difficult to formulate the problem and select the appropriate algorithm.

Application of Deep Learning:

The Deep learning concept has a wide range of applications across multiple sector and areas such as:

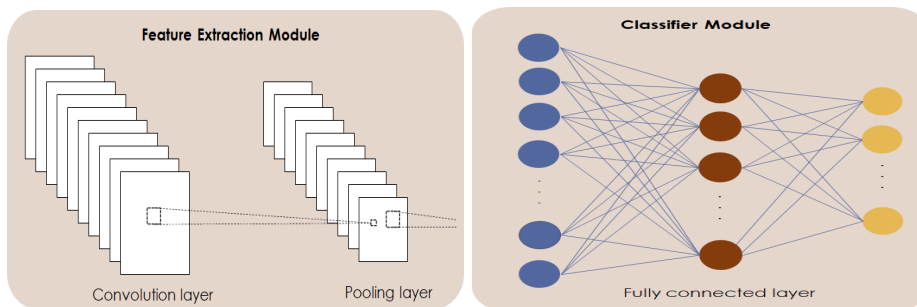
- a. *Computer vision:* Deep learning is used several computer vision related tasks such as image and video recognition, object detection, semantic segmentation,

etc. Applications include self-driving cars, security cameras, face recognition, Disease-pest detection etc.

- b. *Natural language processing*: Deep learning is used in natural language understanding, machine translation, sentiment analysis, and other natural language processing tasks. Applications include chatbots, virtual assistants, and language-based search engines.
- c. *Speech recognition*: Deep learning is used in speech recognition, voice identification, and voice synthesis. Applications include voice-controlled assistants, voice-enabled devices and voice-controlled robots.
- d. *Predictive analytics*: Deep learning is used to analyze historical data and make predictions about future events. Applications include fraud detection, customer churn prediction, and demand forecasting.
- e. *Recommender systems*: Deep learning is used to analyze patterns in data to recommend items to users. Applications include movie and music recommendations, news recommendations, and product recommendations.
- f. *Marketing*: Deep learning is used to analyze customer data, to predict customer behavior and to personalize marketing campaigns. Applications include customer segmentation, customer lifetime value prediction, and personalization
- g. *Robotics*: Deep learning is used to enable robots to learn from experience and adapt to their environment. Applications include autonomous vehicles, drones, and industrial robots.
- h. *Cybersecurity*: Deep learning is used to detect patterns in network traffic, and to identify and respond to cyber threats. Applications include intrusion detection and prevention, and malware detection.

Convolutional Neural Networks (CNNs or ConvNets):

Convolutional Neural Networks (CNNs or ConvNets) are the Deep Learning-based models universally used for Computer Vision task. The CNNs are the advanced version of feed forward Artificial Neural Networks comprised of artificial neurons with learnable weights vectors and biases (Haque *et al.*, 2022a). The concept of the CNNs/ConvNets were evolved from the Biological Visual Cortex. Experimental findings of Hubel and Wiesel (1958-59) reported that biological visual cortex consists of some neurons having small local receptive fields and some neurons have larger receptive fields. Study of the Visual Cortex inspired the development of Neocognitron concept in 1980 (Fukushima, 1980). CNNs have managed to achieve superhuman performance on very complex visual tasks. Applied on major computer vision problems such as image classification, object detection, segmentation etc. Moreover, CNNs are also successful at other tasks, such as voice recognition, natural language processing (NLP), Recommender systems, Time series analysis etc. A CNN algorithm can receive raw pixels of an image as inputs, performs dot products on the pixels using a set of filters then follows with non-linear function and finally generates the classification scores that are used to differentiate one image from another. The CNNs are generally comprised of two modules: feature extraction module and classifier module which are discussed below (Haque *et al.*, 2022b):



Conventional outlook of CNNs/ConvNets

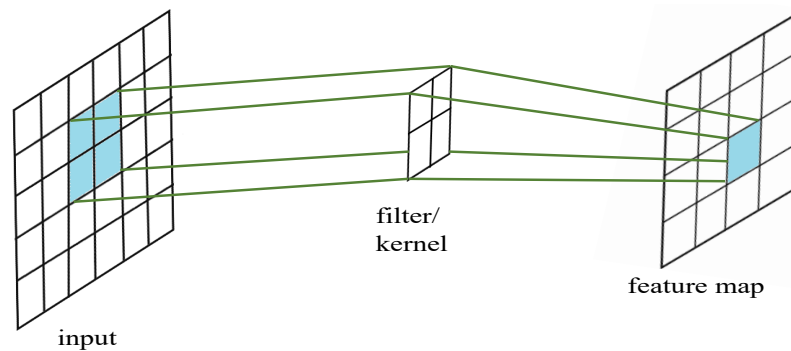
Feature extraction module:

The feature extraction module of CNN extracts promising and inherent features/patterns directly from the raw images. Then, generates a set of feature maps or activation maps in a hierarchical manner. It enables the network to learn semantic information of the images from a series of feature/activation maps. Generally there are two operations involved: Convolution and pooling (subsampling) operations which are discussed below:

Convolution operation: The convolution layer is the first and most significant layer in the convolutional neural networks. It allows the network to learn distinguishable visual features (local patterns or low-level features) such as oriented edges, curves, corners, endpoints, textures etc. from the input images (LeCun et al., 1998). These learned features are then combined in the following layers for detecting the higher-order features in the images. The convolution operation extracts the features by convolving a set of filters (or kernel or weight matrices) through the image pixels and

generates a set of feature maps. Each unit of a feature map in the current layer is associated to the one or more local receptive fields of the previous layer's feature maps through a group of filters or kernels or weight matrices (LeCun et al., 2015). The main advantages of convolution operation are:

- The convolution operation provides sparse connectivity in network as the input pixels of the receptive field are connected only to the respective kernels
- In convolution, shared parameters are used across the layers i.e. one parameter set (kernel matrix) is learned for every location of the image
- Convolution makes the network translation invariance for the input images



Convolution operation of CNNs

The units in a particular feature map of an image share the same set of filters (or weight matrices), whose same receptive fields are situated at different locations in that image (LeCun et al., 1998). The local weighted sum of the input image pixels and the filters (weight matrices) is passed through mapping functions that are non-linear in nature. In CNNs, generally rectified linear unit (ReLU) functions are used as the activation functions. ReLU is actually a non-linear function that acts in a linear way to learn the complex patterns in the data. The ReLU function activates or fires an input node if the value of the node is above or below a particular threshold value. ReLU outputs a zero, if the value of input node is below the threshold value and whenever the input values rise above the threshold value, it will act linearly with the values of the input node. The feature maps are calculated using the formula (1)-

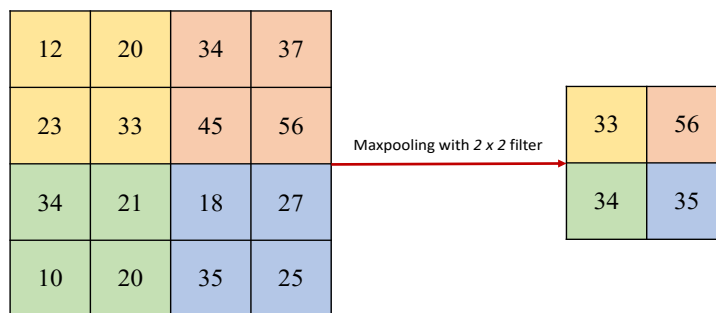
$$x_j^L = f\left(\sum_{j=C_j} x_j^{L-1} \cdot W_{ij}^L + b_j^L\right) \quad (1)$$

where, L denotes the L^{th} layer of the network, x_j^L is the output of the j^{th} feature map at L^{th} layer, W_{ij}^L represents the convolution kernel of the layer, C_j denotes the no. of input feature maps and $f(.)$ denotes the ReLU activation function that is defined by the formula (2)-

$$f(x_j) = \max(0, x_j) \quad (2)$$

where, x_j is the j^{th} feature map of the network. The use of ReLU in deep CNNs has made the training speed of the network faster several times than the other activation functions (Krizhevsky et al., 2012).

Pooling Layer: Pooling operation is a subsampling process where the input feature maps are down-sampled. This operation reduce the computational load, the memory usage, and the number of parameters (thereby limiting the risk of overfitting) of the network. It makes the representations in the feature maps smaller and more manageable. Pooling layer is connected to the outputs of a limited number of neurons in the previous layer, located within a small rectangular receptive field (Haque *et al.*, 2022c). It operates upon each feature map separately to create a new set of pooled feature maps. In convolutional neural network, the most commonly used pooling technique is max-pool in which a max-filter is applied over a non-overlapping region in the feature maps. The max-pool operation reduces the spatial resolution of the feature maps which in turn imposes more insensitivity towards the shift or distortion variations. Therefore, the pooling operation shrinks the no. of learnable parameters of the network and reduces the computational load in the resources. The convolution and Maxpooling operation together act as the automated feature extractor for the convolutional neural networks. The pooling layer also introduces some level of invariance to small translations in the feature maps such as rotations, scaling, transformations.



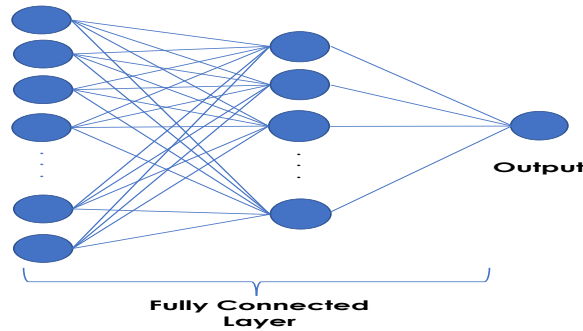
Maxpooling operation in feature maps

Classifier module:

The classifier module is the part of the network that actually classifies the learned feature maps into respective classes. In CNN, the classifier module is implemented by one or more feed forward neural network (fully connected) layer also known as fully connected layer (fc layer)

Fully connected Layer (fc layer): Fully Connected layer (or *fc* layer) of any CNN model typically resembles the architecture of the artificial neural networks. The *fc* layers are generally composed of one input layer, one or more hidden layers and one output layer. In this *fc* layer, each unit of the current layer is associated to each and every unit of the immediate next layer in the network. The feature maps that are generated from the convolution and max-pool layers, are primarily 2-dimentional matrices and get flattened to a 1-dimentional vector of features before input to the fc

layer of the CNNs. The last layer in the *fc* layer contains the equal number of nodes to the total number of classes considered in the classification problem.



Fully connected layer (*fc layer*) of CNNs

Loss Function in CNNs:

The loss function is a tool for the machines to optimize the pattern-learning process from the datasets. This function determines how well the algorithm has modelled with the dataset. If the curve of the loss function has an increasing trend then the predictions are off and if it is having decreasing trend then the predictions are pretty good (Hennig and Kutlukaya, 2007). The loss function is responsible for reducing the errors in the network with the help of some optimization function. The loss functions are of generally two types: Regression loss and Classification loss. Regression loss deals with the real-valued quantities and having several metrics such as mean absolute error (MAE), mean square error (MSE), etc. The classification loss predicts an output from a set of finite categorical values. The popular loss functions for classification problems are binary cross-entropy, categorical cross-entropy, etc. In this experiment, categorical cross-entropy function was used to perform the multi-class classification. It is the most popular and most commonly used loss function. The cross-entropy loss curve shows an increasing trend if the predicted probability value diverges from the true label. An important aspect of the categorical cross-entropy loss is that it penalizes the predictions heavily when these are confident but wrong. The categorical cross-entropy is defined by the following formula (Eq. 3)-

$$\mathbf{CrossEntropyLoss} = -(y_i(\log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)) \quad (3)$$

Where, y_i is the true class label of the i^{th} variable in the test data and \hat{y}_i is the predicted class label of the i^{th} variable in the test data and *CrossEntropyLoss* is the calculated loss of the model.

Optimization function in CNNs

An optimization function is a function that finds a set of input parameters to a function within an allowed set which causes the function to give maximum or minimum output. The word “Optimization” comes from the root word “optimal”, which means the best. When we are optimizing something, we are trying or make it ‘best’ or near to best (Kochenderfer and Wheeler, 2019). An optimization algorithm compares the solutions obtained from repeated executions till an optimum or a satisfactory solution

is achieved. The optimization algorithms are categorized into two types viz. deterministic optimization algorithms and stochastic optimization algorithms. The deterministic algorithm uses pre-specified rules for generating the optimal solution for the given problem and the stochastic optimization algorithms are probabilistic in nature with probabilistic translation rules for generating the optimal solution of the problem (Engelbrecht, 2007). In this study, two most widely used optimization algorithm such as Stochastic Gradient Descent and Adam were used.

a. Stochastic gradient descent (SGD): Stochastic gradient descent (SGD) algorithm is the probabilistic approximation of the well-known algorithm gradient descent. It is an iterative procedure that applies appropriate smoothness properties such as differentiability or sub-differentiability for optimizing any function. SGD performs the parameter-updates for each training data having a data x^i with label y^i by using the formula in eq- 4:

$$\theta = \eta \cdot \nabla_{\theta} J(\theta; x^i; y^i) \quad (4)$$

SGD deals with the problem of redundancy by performing the parameter-updates one at a time and that makes it faster than the other approaches (Ketkar, 2017). In SGD, high variance frequent parameter updates cause deliberate fluctuations in the objective function and enables to reach to potentially better local minima solutions (Bottou, 2012).

b. Adam: In recent years, the Adam optimization algorithm gained much wider adoption for deep learning applications in the area of computer vision. It is the extended version of the stochastic gradient descent algorithm. The name ‘Adam’ has been derived from the term ‘adaptive moment estimation’ because it computes the learning rates for network weights by estimating the first and second order moments of the gradient (Kingma and Ba, 2014). The learning rate in Adam is adaptive in nature where individual learning rates are computed for each network parameters. The Adam algorithm combines the best characteristics from two extended SGD algorithms such as the AdaGrad and RMSprop algorithm. Adam computes the exponential moving average of the gradient and squared gradients like the AdaGrad algorithm and applies the squared gradients to scale the learning rate like the RMSprop algorithm (Goodfellow et al., 2016).

Softmax Function:

Softmax function is the most widely used activation function in deep learning models for classification problems. It provides a vector of values that represents the probability distribution over the total output class. The softmax function converts the numeric output of the network’s last layer into probability values by taking exponents of each network output and normalizing each number by the sum of the exponents. In any CNN model, the softmax function is added in the final layer of the network. The softmax function are used in a classification model having mutually-exclusive classes. The softmax function can be defined the formula provided in the equation Eq. 5-

$$S(y)_i = \frac{e^{y_i}}{\sum_{j=1}^f e^{y_j}} \quad (i = 1, 2, \dots, N) \quad (5)$$

Where, y_i is the i^{th} numeric output of the last convolution layer, f is the number of kernels/filters in the last convolution layer.

References:

- Bottou, L. (2012). Stochastic gradient descent tricks. *In Neural networks: Tricks of the trade*, pp. 421-436. Springer, Berlin, Heidelberg.
- Engelbrecht, A. P. (2007). *Computational Intelligence: An Introduction*. John Wiley & Sons.
- Fukushima, K. (1980). Neocognitron: a self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, **36(4)**, 193.
- Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning (Vol.1, No.2)*. Cambridge: MIT Press.
- Haque, M. A., Marwaha, S., Arora, A., Deb, C. K., Misra, T., Nigam, S., & Hooda, K. S. (2022c). A lightweight convolutional neural network for recognition of severity stages of maydis leaf blight disease of maize. *Frontiers in Plant Science*, 13.
- Haque, M. A., Marwaha, S., Deb, C. K., Nigam, S., & Arora, A. (2022b). Recognition of diseases of maize crop using deep learning models. *Neural Computing and Applications*, 1-15.
- Haque, M. A., Marwaha, S., Deb, C. K., Nigam, S., Arora, A., Hooda, K. S., Soujanya, P. L., Aggarwal, S. K., Lall, B., Kumar, M., Islam, S., Panwar, M., Kumar, P., & Agrawal, R. C. (2022a). Deep learning-based approach for identification of diseases of maize crop. *Scientific Reports*, 12(1), 6334. <https://doi.org/10.1038/S41598-022-10140-Z>
- Hennig, C., & Kutlukaya, M. (2007). Some thoughts about the design of loss functions. *REVSTAT–Statistical Journal*, **5(1)**, 19-39.
- Ketkar, N. (2017). Stochastic gradient descent. *In Deep learning with Python*, pp. 113-132. Apress, Berkeley, CA.
- Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *In Proceedings of International Conference of Learning Representations*. arXiv preprint arXiv:1412.6980.
- Kochenderfer, M. J., & Wheeler, T. A. (2019). *Algorithms for optimization*. Mit Press.
- Krizhevsky, A., Sutskever, I. & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *In Proceedings of Advances in Neural Information Processing Systems*, 1097-1105.

- LeCun, Y., Bengio, Y. & Hinton, G. (2015). Deep learning. *Nature*, **521(7553)**, 436-444.
- LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., & Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural Computation*, **1(4)**, 541-551.
- LeCun, Y., Bottou, L., Bengio, Y. & Haffner, P. (1998). Gradient-based learning applied to document recognition. *In Proceedings of the IEEE*, **86(11)**, 2278-2324

Image Classification using CNN: Tensorflow and Keras

Akshay Dheeraj and Md. Ashraful Haque

ICAR-Indian Agricultural Statistics Research Institute, New Delhi - 110 012

akshay.dhreeraj@icar.gov.in and ashraful.haque@icar.gov.in

Image Classification/Recognition:

Image classification (or Image recognition) is the way of recognizing the instances in the image as a whole and assigning a class or label to the image. It's a one of most important subdomain of computer vision. The main aim is to categorize all the pixels of a digital image into one of the several classes. The image classification involves pre-processing of images, feature extraction and training, and classifying into predefined classes. Now-a-days, the CNNs have become the de-facto techniques for image classification/recognition tasks.

Fashion MNIST Dataset

Recently, Zalando research published a new dataset, which is very similar to the well-known MNIST database of handwritten digits. The dataset is designed for machine learning classification tasks and contains in total 60,000 training and 10,000 test images (grey scale) with each 28x28 pixel. Each training and test case is associated with one of ten labels (0–9). Up till here Zalando's dataset is basically the same as the original handwritten digits data. However, instead of having images of the digits 0–9, Zalando's data contains (not unsurprisingly) images with 10 different fashion products. Consequently, the dataset is called Fashion-MNIST dataset.

The 10 different class labels are:

- 0 T-shirt/top
- 1 Trouser
- 2 Pullover
- 3 Dress
- 4 Coat
- 5 Sandal
- 6 Shirt
- 7 Sneaker
- 8 Bag
- 9 Ankle boot

Implementation of CNN for image classification

In this section, a simple image classification model will developed using tensorflow, Keras framework using the Fashion MNIST dataset. Therefore, there are several steps that are followed during image classification which are provided below:

Step 1: Import Libraries

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
import keras
```

Step 2: Load data

```
(X_train, y_train), (X_test, y_test)=tf.keras.datasets.fashion_mnist.load_
data()
#print the shape of data
X_train.shape,y_train.shape , X_test.shape,y_test.shape
#X_train - NumPy array of grayscale image data with shapes (60000, 28, 28)
, containing the training data.
#y_train - NumPy array of labels (integers in range 0-
9) with shape (60000,) for the training data.
#X_test - NumPy array of grayscale image data with shapes (10000, 28, 28),
containing the test data.
#y_test - NumPy array of labels (integers in range 0-
9) with shape (10000,) for the test data.
#check the data
X_train[0]
y_train[0]

#classes of the dataset
class_labels = ["T-shirt/top", "Trouser", "Pullover", "Dress", "Coat",
"Sandal", "Shirt", "Sneaker", "Bag", "Ankle boot"]

#show training images
plt.figure(figsize=(10,10))
j=1
for i in np.random.randint(0,1000,9):
    plt.subplot(3,3,j);j+=1
    plt.imshow(X_train[i],cmap='Greys')
    plt.axis('off')
    plt.title('{} / {}'.format(class_labels[y_train[i]],y_train[i]))

#add one more dimension to the dataset as input to CNN have three
dimensions
X_train=np.expand_dims(X_test,axis=1)
X_test = np.expand_dims(X_test,-1)

#Feature Scaling (Normalise the data)
X_train = X_train/255
X_test= X_test/255
```

```
#Split the training data randomly in validation and training data (20% as validation and 80% as training data)
```

```
from sklearn.model_selection import train_test_split
X_train,X_Validation,y_train,y_Validation=train_test_split(X_train,y_train
,test_size=0.2,random_state=2023)
```

Step 3: Building the CNN model

```
model=keras.models.Sequential([

keras.layers.Conv2D(filters=32,kernel_size=3, strides=(1,1),padding='valid'
,activation='relu',input_shape=[28,28,1]),
                    keras.layers.MaxPooling2D(pool_size=(2,2)),
                    keras.layers.Conv2D(filters=32,kernel_size=3,
strides=(1,1),activation='relu'),
                    keras.layers.MaxPooling2D(pool_size=(2,2)),
                    keras.layers.Flatten(),
                    keras.layers.Dense(units=128,activation='relu'),
                    keras.layers.Dropout(0.3),
                    keras.layers.Dense(units=10,activation='softmax')

])
```

```
# we have created the model with 2 convolution, 2 max pooling and 1 dense layer with 128 neuron and 1 softmax layer with 10 nodes.
```

```
#Check the summary of the developed model
model.summary()
```

```
#Compile the model with Adam optimizer with learning rate = 0.001 and sparse_categorical_crossentropy as loss function
```

```
model.compile(optimizer=keras.optimizers.Adam(lr=0.001),loss='sparse_categorical_crossentropy',metrics=['accuracy'])
```

```
#Fit the model on the training and validation data with parameter epoch and batch size
```

```
model.fit(X_train,y_train,epochs=3,batch_size=512,verbose=1,validation_data=(X_Validation,y_Validation))
```

```
#Plot the accuracy and loss curve
```

```
def accuracy_loss_plots(model):
    fig, (ax1,ax2) = plt.subplots(nrows=1,ncols=2,figsize=(12,5))
    ax1.plot(model.history.history['accuracy'], label='Training accuracy')
    ax1.plot(model.history.history['val_accuracy'], label='Validation accuracy')
    ax1.set_title('Accuracy Curve')
```

```
ax1.set_xlabel('Epoch')
ax1.set_ylabel('Accuracy')
#ax1.set_ylim(0,1)
ax1.legend()

ax2.plot(model.history.history['loss'], label='Training loss')
ax2.plot(model.history.history['val_loss'], label='Validation loss')
ax2.set_title('Loss Curve')
ax2.set_xlabel('Epoch')
ax2.set_ylabel('Loss')
#ax2.set_ylim(0,1)
ax2.legend();

accuracy_loss_plots(model)

#storing predictions in variable y_pred which gives the probability value
the predicted class.

y_pred = model.predict(X_test)

#evaluating model on test set to check the loss and accuracy on the test
data
model.evaluate(X_test, y_test)

#Display the random images with actual and predicted label.
plt.figure(figsize=(12,12))

j=1
for i in np.random.randint(0, 1000,9):
    plt.subplot(3,3, j); j+=1
    plt.imshow(X_test[i].reshape(28,28), cmap = 'Greys')
    plt.title('Actual = {} / {} \nPredicted = {} / {}'.format(class_labels[y
_test[i]], y_test[i], class_labels[np.argmax(y_pred[i])],np.argmax(y_pred[
i])))
    plt.axis('off')

#classification report and confusion matrix
from sklearn.metrics import confusion_matrix
y_pred_labels = [ np.argmax(label) for label in y_pred ]
cm = confusion_matrix(y_test, y_pred_labels) #normalize='true'

from sklearn.metrics import classification_report
cr= classification_report(y_test, y_pred_labels, target_names=class_labels
)
print(cr)
```

```
sns.heatmap(cm, annot=True, fmt='d',xticklabels=class_labels, yticklabels=
class_labels)
plt.title('Fashion MNIST Confusion Matrix')
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
```

Basics of Digital Image Processing

Alka Arora and Chandan Kumar Deb

ICAR-Indian Agricultural Statistics Research Institute, New Delhi - 110 012

alka.arora@icar.gov.in and Chandan.deb@icar.gov.in

Digital, Image, Processing are three important words; which means processing of the Image which is Digital in nature by a Digital Computer using algorithms.

Motivation of Digital Image Processing:

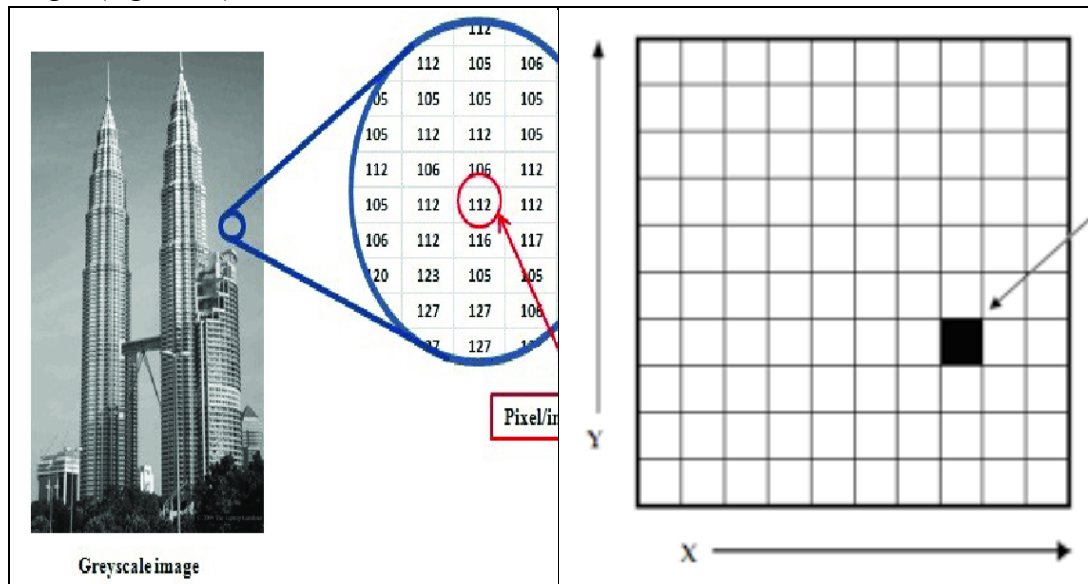
1. Improvement of pictorial information
2. Information can be extracted
3. Efficient Storage, Security and Transmission

Basics of Digital Image

An image may be defined as a two-dimensional function, where x and y are *spatial* (plane) coordinates, and the amplitude of f at any pair of coordinates (x, y) is called the *intensity* or *gray level* of the image at that point. Each element $f(x,y)$ at location (x,y) is called a Pixel. Ex: $f(1,1) = 130$; $f(1,1)$ is pixel location and 130 is pixel intensity value.

Digital Image Representation in Computer:

A digital image is the composition of individual pixels or picture elements. The pixels are arranged in the form of row and column to form a picture area. The number of pixels in an image is a function of the size of the image and number of pixels per unit length (e.g., inch) in horizontal as well as vertical direction.



The size of the image is defined as the total number of pixels in the horizontal direction times the total number of pixels in the vertical direction (512 x 512,640 x 480, or 1024 x 768).

Resolution

The resolution can be defined in many ways. Such as pixel resolution, spatial resolution, temporal resolution, spectral resolution. Out of which we are going to discuss pixel resolution. You have probably seen that in your own computer settings, you have **monitor resolution** of 800 x 600, 640 x 480 etc. In pixel resolution, the term resolution refers to the total number of count of pixels in a digital image. For example. **If an image has M rows and N columns, then its resolution can be defined as M X N.**

If we define resolution as the total number of pixels, then pixel resolution can be defined with set of two numbers. The first number the width of the picture, or the pixels across columns, and the second number is height of the picture, or the pixels across its width. We can say that the higher is the pixel resolution, the higher is the quality of the image.

Bit depth refers to the color information stored in an image. The higher the bit depth of an image, the more colors it can store. The simplest image, a 1 bit image, can only show two colors, black and white. That is because the 1 bit can only store one of two values, 0 (white) and 1 (black). An 8 bit image can store 256 possible colors, while a 24 bit image can display over 16 million colors. As the bit depth increases, the file size of the image also increases because more color information has to be stored for each pixel in the image.

Image Color

Binary Image– The binary image as its name suggests, contain only two pixel elements i.e 0 & 1, where 0 refers to black and 1 refers to white. This image is also known as Monochrome.

Grayscale Image- In this image format, each pixel can take value between 0 and number of grayscales. These images appear like normal black and white photographs. These images can have 256 shades of grey. Humans can distinguish into 40 shades of grey .

Color Image (RGB)- Color images are similar to grey scale images except that there are three bands/channels corresponding to Red, Green and Blue. Each pixel has three values associated with it. Image color can also be represented as HSV- Hue, Saturation and Value(Intensity) of the image.

8 bit COLOR FORMAT– It is the most famous image format. It has 256 different shades of colors in it and commonly known as Grayscale Image. In this format, 0 stands for Black, and 255 stands for white, and 127 stands for gray.

16 bit COLOR FORMAT– It is a color image format. It has 65,536 different colors in it. It is also known as High Color Format. In this format the distribution of color is not as same as Grayscale image.

Open CV Introduction

OpenCV was started at Intel in 1999 by Gary Bradsky and the first release came out in 2000. Vadim Pisarevsky joined Gary Bradsky to manage Intel's Russian software OpenCV team. In 2005, OpenCV was used on Stanley, the vehicle who won 2005 DARPA Grand Challenge. Later its active development continued under the support of Willow Garage, with Gary Bradsky and Vadim Pisarevsky leading the project.

OpenCV supports a wide variety of programming languages like C++, Python, Java etc and is available on different platforms including Windows, Linux, OS X, Android, iOS etc. Also, interfaces based on CUDA and OpenCL are also under active development for high-speed GPU operations.

OpenCV-Python is the Python API of OpenCV. It combines the best qualities of OpenCV C++ API and Python language.

In OpenCV, the CV is an abbreviation form of a computer vision, which is defined as a field of study that helps computers to understand the content of the digital images such as photographs and videos. It works with NumPy, Scikit-image, and Matplotlib. It uses the BGR (Blue, Green and Red) color model for the images.

The purpose of computer vision is to understand the content of the images. It extracts the description from the pictures, which may be an object, a text description, and three-dimension model, and so on.

Working with OpenCV-Python

Read an image from file (using `cv::imread`)

Display an image in an OpenCV window (using `cv::imshow`)

Write an image to a file (using `cv::imwrite`)

OpenCv-Python is to be installed first. The following commands work well with Google Collab, which has already installed CV library.

Code:

```
#!pip install opencv-python
import cv2 as cv
import matplotlib.pyplot as plt
```

Syntax of imread `cv.imread(filename[,flag])` **filename:** Name of the file along with path to be loaded

flag: The flag specifies the color type of a loaded image **The imread()** function returns a matrix. Flag 0 --grey; 1- coloured; -1: Unchanged

```
img=cv.imread('/content/drive/MyDrive/AI Training/lena.tif')
## reads image in BGR format
img # To display image matrix
type(img) # To display type numpy
plt.imshow(img) # To display image
plt.show() # To display image
## To display image on the console
##cv.imshow('display window',img)
#cv.waitKey(3) # 0- Any key, 3- Enter Key
#cv.destroyAllWindows()
grey =cv.imread('/content/drive/MyDrive/AI Training/lena.tif'
,0) # To read image in grey format
plt.imshow(grey)
plt.show()
```

To Save Image to Desired Location

```
cv.imwrite('/content/drive/MyDrive/AI Training/Lena_Grey.png',grey)
```

To convert image from one format to another

```
img=cv.imread('/content/drive/MyDrive/AI Training/lena.tif')
rgb = cv.cvtColor(img, cv.COLOR_BGR2RGB )
plt.imshow(rgb)
plt.axis('off') ## Display Off for the axis
plt.show()
```

Properties of Image

```
#shape of image
dimensions= img.shape
print('Image Dimension      : ',dimensions)
#height
height = img.shape[0]
print('Image Height        : ',height)
# width
width = img.shape[1]
print('Image Width          : ',width)
# no. of channel
channels = img.shape[2]
print('Number of Channels : ',channels)
# Size of the image
size1 = img.size
print('Image Size      :', size1)
```

Splitting the image channels

```
## splitting the image into its color channels
img=cv.imread("lena.tif")
b,g,r = cv.split(img)
## show the respective color channels
plt.subplot(1,3,1)
plt.imshow(b) ## for red channel write r and for green channel write g
plt.subplot(1,3,2) ## To display images in one row; 1-
no of rows, 3-
no of column, 2 is the col position where image will start
plt.imshow(g)
plt.subplot(1,3,3)
plt.imshow(r)
plt.show() ## Code to display different shades together
```

Drawing Different Shapes on Images User can draw the various shapes on an image such as circle, line, rectangle, ellipse, etc. It is used when we want to highlight any object in the input image.

Drawing Circle User can draw the circle on the image by using the cv2.circle() function. The syntax is the following:

```
cv2.circle(img, center, radius, color[,thickness [, lineType]])
img=cv.imread('/content/drive/MyDrive/AI Training/lena.tif')
```

```
#img=cv.imread("lena.tif")
#cv.circle(img, (300,300), 80, (0,255,255), -1)
#cv.circle?

img_circle=cv.circle(img, (300,300), 80, (0,255,255), 5)
img_circle=cv.circle(img, (300,300), 20, (0,255,255), -1)
plt.imshow(img_circle)
plt.show()
```

Drawing Rectanle cv2.rectangle(img, pt1, pt2, color[, thickness[,lineType]])

```
img=cv.imread('/content/drive/MyDrive/AI Training/lena.tif')
img_rect=cv.rectangle(img, (200,200), (400,400), (0,255,255), 15)

plt.imshow(img_rect)
plt.show()
```

Drawing lines

cv2.line(img, pt1, pt2, color[, thickness])

```
img=cv.imread('/content/drive/MyDrive/AI Training/lena.tif')
cv.line(img, (100,100), (250,250), (0,0,0), 30)
plt.imshow(img)
plt.show()
```

Write Text on Image User can write text on the image by using the putText() function. The syntax is given below.

cv2.putText(img, text, org, font, color)

```
img=cv.imread('/content/drive/MyDrive/AI Training/lena.tif')
font = cv.FONT_HERSHEY_SIMPLEX
cv.putText(img, 'Hello', (10,500), font, 2, (255,255,255), 5)
plt.imshow(img)
plt.show()
```

Resize the image

In the image processing, we need to resize the image to perform the particular image processing operation.

cv2.resize(src, dsize[, dst[, fx[,fy[,interpolation]]])

- src - source/input image (required).
- dsize - desired size for the output image(required)
- fx - Scale factor along the horizontal axis.(optional)
- fy - Scale factor along the vertical axis.
- Interpolation(optional) - This flag uses following methods:
 - INTER_NEAREST - A nearest-interpolation

- INTER_AREA - resampling using pixel area relation. When we attempt to do image zoom, it is similar to the INTER_NEAREST method.
- INTER_CUBIC - A bicubic interpolation over 4×4 pixel neighborhood.
- INTER_LANCOZS4 - Lanczos interpolation over 8×8 pixel neighborhood.

```
## Resize the image dimension
img1 = cv.imread('/content/drive/MyDrive/AI Training/healthy.
jpeg', 1)
resized = cv.resize(img1, (256, 256), interpolation=cv.INTER_
AREA)

#Size of original image
print('Original Image Size      : ',img1.shape)

#Size of resized image
print('Resize Image Size      : ',resized.shape)

## show the resized image
plt.imshow(resized)
plt.show()
img = cv.imread('/content/drive/MyDrive/AI Training/healthy.j
peg', 1)
print('Original Dimensions : ', img.shape)
#plt.imshow(img)
#plt.show()
width = img.shape[1] # keep original width
height = 1000
dim = (width, height)

# resize image
resized = cv.resize(img, dim, interpolation=cv.INTER_AREA)

print('Resized Dimensions : ', resized.shape)
plt.imshow(resized)
plt.show()

img = cv.imread('/content/drive/MyDrive/AI Training/healthy.j
peg', 1)
width = 350
height = 450
dim = (width, height)
# resize image
resized = cv.resize(img, dim, interpolation=cv.INTER_AREA)
print('Resized Dimensions : ', resized.shape)
plt.imshow(resized)
plt.show()
```

Image Transformation

A transformation that can be expressed in the form of a matrix multiplication (linear transformation) followed by a vector addition (translation). OpenCV calculates the affine matrix that performs affine transformation, which means it does not preserve the angle between the lines or distances between the points, although it preserves the ratio of distances between points lying on the lines. Following transformations are possible

- **Rotation:** The image can be rotated in various angles (90,180,270 and 360)
- **Scaling:** scaling the image to certain level

Opencv uses two function for performing the Transformations. The syntax is:

```
M = cv2.getRotationMatrix2D(center, angle, scale)
```

- **center:** It represents the center of the image.
- **angle:** It represents the angle by which a particular image to be rotated in the anti-clockwise direction.
- **scale:** The value 1.0 is denoted that the shape is preserved. Scale the image according to the provided value.

```
transform = cv2.warpAffine(img,M,(w,h))
```

```
img: original Image
M: Rotation matrix
h, w = Height and width of the original Image
# get image height, width
img = cv.imread('/content/drive/MyDrive/AI Training/healthy.j
peg', 1)
(h, w) = img.shape[:2]
# calculate the center of the image
center = (w / 2, h / 2)

## rotate image 90 degree
M= cv.getRotationMatrix2D(center,90, 1)
rotated90 = cv.warpAffine(img, M, (h, w))
# 180 degrees
M = cv.getRotationMatrix2D(center,180,1)
rotated180 = cv.warpAffine(img, M, (w, h))
# 270 degrees
M = cv.getRotationMatrix2D(center,270,1)
rotated270 = cv.warpAffine(img, M, (w, h))
## show the original image
plt.imshow(img)
plt.show()

## show the rotated image
plt.imshow(rotated90)
plt.show()
plt.imshow(rotated180)
plt.show()
plt.imshow(rotated270)
```

```
plt.show()
## scaling the image
img = cv.imread('/content/drive/MyDrive/AI Training/healthy.j
peg', 1)
# get image height, width
(h, w) = img.shape[:2]
# calculate the center of the image
center = (w / 2, h / 2)

## Scale the image to certain level
M = cv.getRotationMatrix2D(center,0, 1.5)
scaled = cv.warpAffine(img, M, (h, w))

## show the original image
plt.imshow(img)
plt.show()

## show the scaled image
plt.imshow(scaled)
plt.show()
```

References

<https://opencv.org/>

https://docs.opencv.org/4.x/d9/df8/tutorial_root.html

Tutorial: <https://www.javatpoint.com/opencv>

<https://www.tutorialspoint.com/opencv/index.htm>

ENSEMBLE TECHNIQUES

Shashi Dahiya

ICAR-Indian Agricultural Statistics Research Institute, New Delhi - 110 012

shashi.dahiya@icar.gov.in

Introduction:

Ensemble methods are techniques that aim at improving the accuracy of results in models by combining multiple models instead of using a single model. They combine multiple algorithms to produce better classification performance. It is a machine learning approach to combine multiple other models in the prediction process. The combined models increase the accuracy of the results significantly. Those models are referred to as base estimators. It is a solution to overcome the following technical challenges of building a single estimator: High variance: The model is very sensitive to the provided inputs to the learned features.

- Low accuracy: One model or one algorithm to fit the entire training data might not be good enough to meet expectations.
- Features noise and bias: The model relies heavily on one or a few features while making a prediction.

Bagging is used to reduce the variance of weak learners. Boosting is used to reduce the bias of weak learners. Stacking is used to improve the overall accuracy of strong learners.

Ensemble Algorithm:

A single algorithm may not make the perfect prediction for a given dataset. Machine learning algorithms have their limitations and producing a model with high accuracy is challenging. If we build and **combine** multiple models, the overall accuracy could get boosted. The combination can be implemented by aggregating the output from each model with two objectives: reducing the model error and maintaining its generalization. The way to implement such aggregation can be achieved using some techniques. Some textbooks refer to such architecture as *meta-algorithms*.

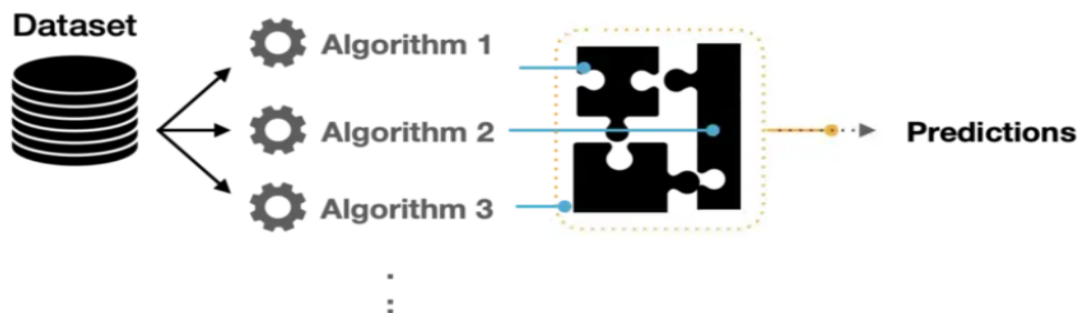


Figure 1: Diversifying the model predictions using multiple algorithms.

Ensemble Learning:

Building ensemble models is not only focused on the variance of the algorithm used. For instance, we could build multiple C45 models where each model is learning a specific pattern specialized in predicting one aspect. Those models are called **weak learners** that can be used to obtain a meta-model. In this architecture of ensemble learners, the inputs are passed to each weak learner while collecting their predictions. The combined prediction can be used to build a final ensemble model.

One important aspect to mention is those weak learners can have different ways of mapping the features with variant decision boundaries.

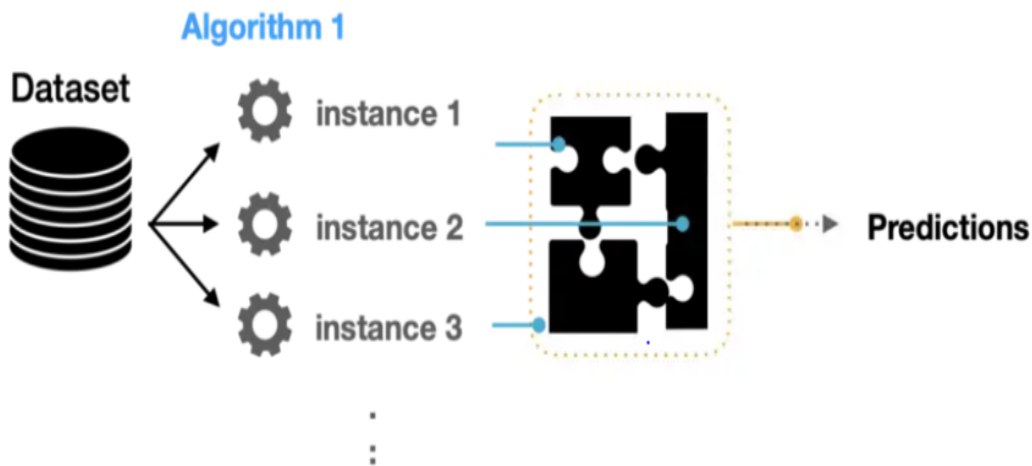


Figure 2: Aggregated predictions using multiple weak learners of the same algorithm.

Ensemble Techniques:

a. Bagging:

We use bagging for combining weak learners of high variance. Bagging aims to produce a model with lower variance than the individual weak models. These weak learners are homogenous, meaning they are of the same type. Bagging is also known as Bootstrap aggregating. It consists of two steps: bootstrapping and aggregation.

Bootstrapping: Involves resampling subsets of data with replacement from an initial dataset. In other words, subsets of data are taken from the initial dataset. These subsets of data are called bootstrapped datasets or, simply, bootstraps. Resampled 'with replacement' means an individual data point can be sampled multiple times. Each bootstrap dataset is used to train a weak learner.

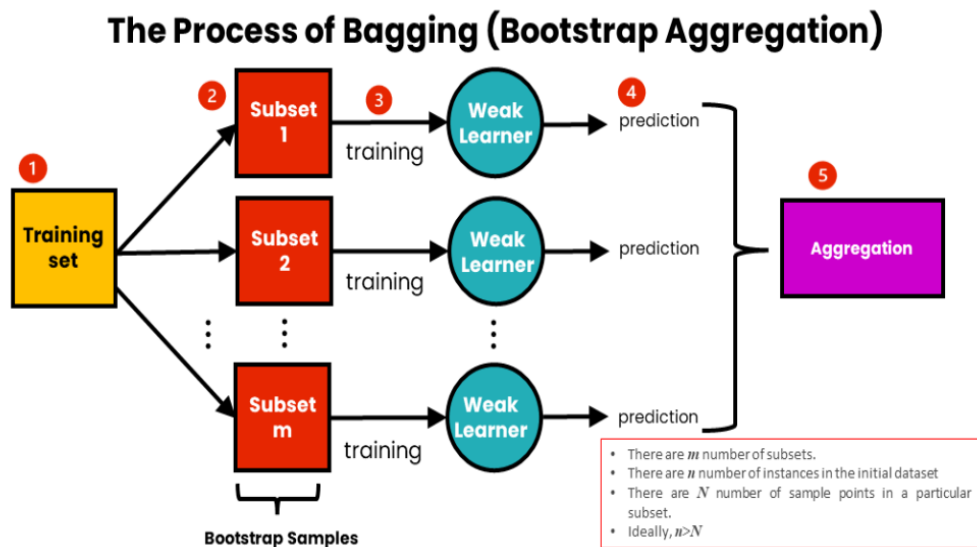
Aggregating: The individual weak learners are trained independently from each other. Each learner makes independent predictions. The results of those predictions are aggregated at the end to get the overall prediction. The predictions are aggregated using either max voting or averaging.

- **Max Voting** is commonly used for classification problems. It consists of taking the mode of the predictions (the most occurring prediction). It is called

voting because like in election voting, the premise is that ‘the majority rules’. Each model makes a prediction. A prediction from each model counts as a single ‘vote’. The most occurring ‘vote’ is chosen as the representative for the combined model.

- **Averaging** is generally used for regression problems. It involves taking the average of the predictions. The resulting average is used as the overall prediction for the combined model.

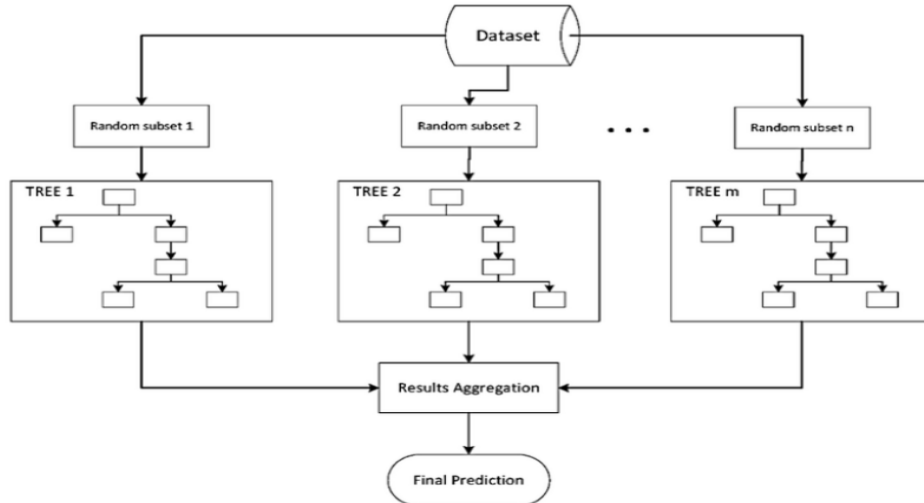
It is one of the most straightforward and most intuitive ensemble-based algorithms that create separate samples of the training dataset. Each training dataset is used to train a different classification.



The idea of bagging is based on making the training data available to an iterative process of learning. Each model learns the error produced by the previous model using a slightly different subset of the training dataset. Bagging reduces variance and minimizes overfitting. One example of such a technique is the Random Forest algorithm.

The steps of Bagging are as follows:

1. We have an initial training dataset containing n -number of instances.
2. We create a m -number of subsets of data from the training set. We take a subset of N sample points from the initial dataset for each subset. Each subset is taken with replacement. This means that a specific data point can be sampled more than once.
3. For each subset of data, we train the corresponding weak learners independently. These models are homogeneous, meaning that they are of the same type.
4. Each model makes a prediction.
5. The predictions are aggregated into a single prediction. For this, either max voting or averaging is used.



Given a Dataset, bootstrapped subsamples are pulled. A Decision Tree is formed on each bootstrapped sample. The results of each tree are aggregated to yield the strongest, most accurate predictor.

Bagging Algorithm:

Input:

Data Set $D = \{(X_1, Y_1), (X_2, Y_2), \dots, (X_n, Y_n)\}$

Number of iteration T

Process:

Step 1: for $i = 1$ to T

(a) Through sampling data points with replacement, create a dataset sample S_m .

(b) From each dataset sample, S_m learns a classifier C_m .

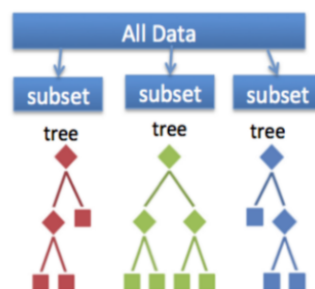
Step 2: for every test example.

(a) Try all classifiers C_m .

(b) Estimate the class that earns the largest number of votes

b. Random Forest:

Random Forest is another ensemble machine learning algorithm that follows the bagging technique. It is an extension of the bagging estimator algorithm. The base estimators in random forest are decision trees. Unlike bagging meta estimator, random forest randomly selects a set of features which are used to decide the best split at each node of the decision tree. It uses subset of training samples as well as subset of features to build multiple split trees. Multiple decision trees are built to fit each training set. The distribution of samples/features is typically implemented in a random mode.



A random forest takes a random subset of features from the data, and creates n random trees from each subset. Trees are aggregated together at end.

Looking at it step-by-step, this is what a random forest model does:

1. Random subsets are created from the original dataset (bootstrapping).
2. At each node in the decision tree, only a random set of features are considered to decide the best split.
3. A decision tree model is fitted on each of the subsets.
4. The final prediction is calculated by averaging the predictions from all decision trees.

Note: The decision trees in random forest can be built on a subset of data and features. Particularly, the sklearn model of random forest uses all features for decision tree and a subset of features are randomly selected for splitting at each node.

To sum up, Random forest randomly selects data points and features, and builds multiple trees (Forest).

c. Extra-Trees Ensemble:

Extra-Trees Ensemble is another ensemble technique where the predictions are combined from many decision trees. Similar to Random Forest, it combines a large number of decision trees. However, the Extra-trees use the whole sample while choosing the splits randomly.

d. Boosting:

We use boosting for combining weak learners with high bias. Boosting aims to produce a model with a lower bias than that of the individual models. Like in bagging, the weak learners are homogeneous.

Boosting involves sequentially training weak learners. Here, each subsequent learner improves the errors of previous learners in the sequence. A sample of data is first taken from the initial dataset. This sample is used to train the first model, and the model makes its prediction. The samples can either be correctly or incorrectly predicted. The samples that are wrongly predicted are reused for training the next model. In this way, subsequent models can improve on the errors of previous models. Unlike bagging, which aggregates prediction results at the end, boosting aggregates the results at each step. They are aggregated using weighted averaging.

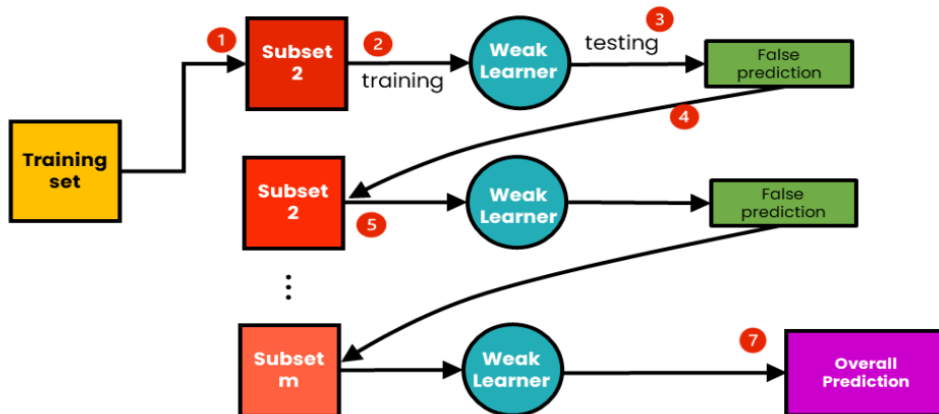
Weighted averaging involves giving all models different weights depending on their predictive power. In other words, it gives more weight to the model with the highest predictive power. This is because the learner with the highest predictive power is considered the most important.

Boosting works with the following steps:

1. We sample m-number of subsets from an initial training dataset.
2. Using the first subset, we train the first weak learner.

3. We test the trained weak learner using the training data. As a result of the testing, some data points will be incorrectly predicted.
4. Each data point with the wrong prediction is sent into the second subset of data, and this subset is updated.
5. Using this updated subset, we train and test the second weak learner.
6. We continue with the following subset until the total number of subsets is reached.
7. We now have the total prediction. The overall prediction has already been aggregated at each step, so there is no need to calculate it.

The Process of Boosting



Algorithm:

Input:

Data set $D = \{(X_1, Y_1), (X_2, Y_2), \dots, (X_n, Y_n)\}$

Number of iteration T

Process:

Step 1: Initialize Weight: Each case receives the same weight.

$W_i = 1/N$, where $i = 1, 2, 3 \dots N$.

Step 2: Construct a classifier using current weight, Compute its error:

$$E_m = \frac{\sum w_i \times I\{Y_i \neq g_m(x_i)\}}{\sum w_i}$$

Step 3: Get a classifier influence and update example weight.

$$a_m = \log\left(\frac{1 - E_m}{E_m}\right)$$

Step 4: Go to step 2.

e. Adaptive Boosting (AdaBoost):

Adaptive Boosting (AdaBoost) is an ensemble of algorithms, where we build models on the top of several weak learners. As we mentioned earlier, those learners are called weak because they are typically simple with limited

prediction capabilities. It is one of the simplest boosting algorithms. Usually, decision trees are used for modelling. Multiple sequential models are created, each correcting the errors from the last model. AdaBoost assigns weights to the observations which are incorrectly predicted and the subsequent model works to predict these values correctly.

The adaptation capability of AdaBoost made this technique one of the earliest successful binary classifiers. **Sequential** decision trees were the core of such adaptability where each tree is adjusting its weights based on prior knowledge of accuracies. Hence, we perform the training in such a technique in sequential rather than parallel process. In this technique, the process of training and measuring the error in estimates can be repeated for a given number of iteration or when the error rate is not changing significantly.

AdaBoost was the first boosting technique and is still now widely used in several domains. AdaBoost, in theory, is not prone to overfitting. Stage-wise estimation may slow down the learning process since parameters aren't jointly optimized. AdaBoost may be used to increase the accuracy of the weak classifiers, allowing it to be more flexible. It requires no normalization and has a low generalization error rate. However, training the algorithm takes enormous time. The method is also susceptible to noisy data and outliers. Therefore, removing them before employing them is strongly advised.

Looking at it step-by-step, this is what a AdaBoost model does:

1. Initially, all observations in the dataset are given equal weights.
2. A model is built on a subset of data.
3. Using this model, predictions are made on the whole dataset.
4. Errors are calculated by comparing the predictions and actual values.
5. While creating the next model, higher weights are given to the data points which were predicted incorrectly.
6. Weights can be determined using the error value. For instance, higher the error more is the weight assigned to the observation.
7. This process is repeated until the error function does not change, or the maximum limit of the number of estimators is reached.

f. Gradient Boosting:

Gradient Boosting or GBM is another ensemble machine learning algorithm that works for both regression and classification problems. GBM uses the boosting technique, combining a number of weak learners to form a strong learner. Regression trees used as a base learner, each subsequent tree in series is built on the errors calculated by the previous tree. Gradient boosting algorithms are great techniques that have high predictive performance. Xgboost, LightGBM, and CatBoost are popular boosting algorithms that can

be used for regression and classification problems. Their popularity has significantly increased after their proven ability to win some Kaggle competitions.

g. Stacking

Stacking, also known as Stacked Generalization, is used to improve the prediction accuracy of strong learners. Stacking aims to create a single robust model from multiple heterogeneous strong learners.

Stacking differs from bagging and boosting in that:

- It combines strong learners
- It combines heterogeneous models
- It consists of creating a Metamodel. A metamodel is a model created using a new dataset.

Individual heterogeneous models are trained using an initial dataset. These models make predictions and form a single new dataset using those predictions. This new data set is used to train the metamodel, which makes the final prediction. The prediction is combined using weighted averaging. Because stacking combines strong learners, it can combine bagged or boosted models.

Stacking is a method similar to boosting. It is an interesting way of combining different models where multiple different algorithms are applied to the training dataset to create a model. The Meta classifier is used to predict unseen data accurately. They produce more robust predictors. It is a process of learning how to create such a stronger model from all weak learners' predictions.

It is an ensemble technique that combines multiple classifications or regression models via a meta-classifier or a meta-regressor. The base-level models are trained on a complete training set, then the meta-model is trained on the features that are outputs of the base-level model. The base-level often consists of different learning algorithms and therefore stacking ensembles are often heterogeneous.

The models(Base-Model) in stacking are typically different (e.g. not all decision trees) and fit on the same dataset. Also, a single model(Meta-model) is used to learn how to best combine the predictions from the contributing models.

The architecture of a stacking model involves two or more base models, often referred to as level-0 models and a meta-model. Meta-model, also referred to as a level-1 model combines the predictions of the base models.

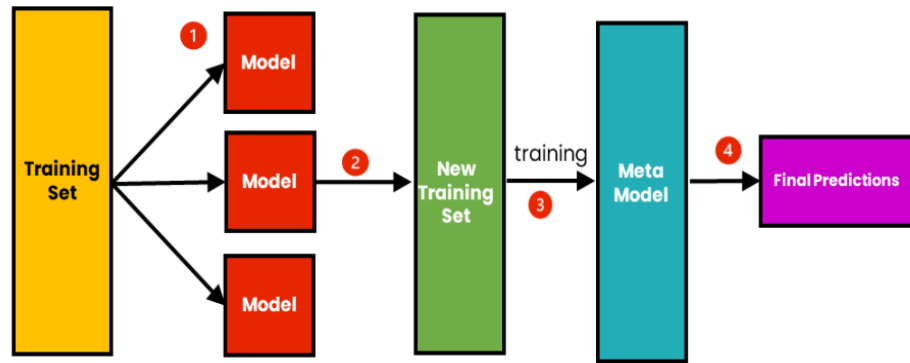
The steps of Stacking are as follows:

1. We use initial training data to train m-number of algorithms.
2. Using the output of each algorithm, we create a new training set.
3. Using the new training set, we create a meta-model algorithm.

- Using the results of the meta-model, we make the final prediction. The results are combined using weighted averaging.

The outputs from the base models used as input to the meta-model may be real values in the case of regression, and probability values, probability like values, or class labels in the case of classification.

The Process of Stacking



Please note that what is being learned here (as features) is the prediction from each model.

When to use Bagging, Boosting and Stacking?

	Bagging	Boosting	Stacking
Purpose	Reduce Variance	Reduce Bias	Improve Accuracy
Base Learner Types	Homogeneous	Homogeneous	Heterogeneous
Base Learner Training	Parallel	Sequential	Meta Model
Aggregation	Max Voting, Averaging	Weighted Averaging	Weighted Averaging

- If you want to reduce the overfitting or variance of your model, you use bagging. If you are looking to reduce underfitting or bias, you use boosting. If you want to increase predictive accuracy, use stacking.
- Bagging and boosting both works with homogeneous weak learners. Stacking works using heterogeneous solid learners.
- All three of these methods can work with either classification or regression problems.

- One disadvantage of boosting is that it is prone to variance or overfitting. It is thus not advisable to use boosting for reducing variance. Boosting will do a worse job in reducing variance as compared to bagging.
- On the other hand, the converse is true. It is not advisable to use bagging to reduce bias or underfitting. This is because bagging is more prone to bias and does not help reduce bias.
- Stacked models have the advantage of better prediction accuracy than bagging or boosting. But because they combine bagged or boosted models, they have the disadvantage of needing much more time and computational power. If you are looking for faster results, it's advisable not to use stacking. However, stacking is the way to go if you're looking for high accuracy.

Conclusion

- Ensemble learning combines multiple machine learning models into a single model. The aim is to increase the performance of the model.
- Bagging aims to decrease variance, boosting aims to decrease bias, and stacking aims to improve prediction accuracy.
- Bagging and boosting combine homogenous weak learners. Stacking combines heterogeneous solid learners.
- Bagging trains models in parallel and boosting trains the models sequentially.
- Stacking creates a meta-model

Big Data Analysis using Pandas

Samarth Godara

ICAR-Indian Agricultural Statistics Research Institute, New Delhi - 110 012

samarth.godara@icar.gov.in

Introduction to Big Data Analytics in Agriculture

Agriculture is one of the oldest and most important industries in the world. It is the backbone of many economies, providing food, fiber, and other essential resources to the global population. However, as the world's population continues to grow, the agriculture industry is facing new challenges. One of the biggest challenges is the need to increase production while reducing the environmental impact of farming practices.

Big data analytics has the potential to revolutionize the agriculture industry by providing farmers and researchers with valuable insights into crop yields, weather patterns, soil conditions, and other critical factors that affect crop growth. By analyzing large amounts of data, farmers and researchers can make more informed decisions about planting, harvesting, and managing crops, which can lead to increased yields, reduced costs, and a more sustainable future for the agriculture industry.

In this lecture, we will discuss the role of big data analytics in agriculture and the various data sources that are used to inform agricultural decision making. We will also review the most popular techniques and algorithms used in big data analytics, such as machine learning, deep learning, and predictive modeling.

Data Sources for Big Data Analytics in Agriculture

The agriculture industry generates a vast amount of data from various sources, including weather stations, drones, satellites, and IoT sensors. This data can be used to inform agricultural decision making, such as planting and harvesting times, irrigation, fertilization, and pest management.

Weather data is one of the most important data sources for agriculture, providing information on temperature, precipitation, wind speed, and other factors that affect crop growth. Weather stations and weather satellites provide accurate and detailed data that can be used to predict weather patterns and make informed decisions about planting and harvesting times.

Drones and satellites are also becoming increasingly popular in agriculture, providing high-resolution images and data that can be used to analyze crop health, soil conditions, and other factors that affect crop growth. Drones can also be used to spray pesticides and fertilizers, reducing the need for manual labor.

IoT sensors are also becoming increasingly popular in agriculture, providing data on soil moisture, pH levels, temperature, and other factors that affect crop growth. This

data can be used to improve irrigation and fertilization practices, leading to increased yields and reduced costs.

Techniques and Algorithms in Big Data Analytics

Machine learning and deep learning are the most popular techniques used in big data analytics in agriculture. Machine learning algorithms can be used to analyze large amounts of data and make predictions about crop yields, weather patterns, and soil conditions.

Deep learning algorithms, such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs), can be used to analyze images and data from drones and satellites. These algorithms can be used to detect patterns and anomalies in crop growth, such as disease and pests, and predict crop yields.

Predictive modeling is also commonly used in agriculture to predict crop yields, weather patterns, and other factors that affect crop growth. Predictive models can be used to make informed decisions about planting, harvesting, and managing crops, leading to increased yields and reduced costs.

Applications of Big Data Analytics in Agriculture

Big data analytics can be used to improve various aspects of the agriculture industry, such as crop yields, sustainability, and efficiency.

One of the most important applications of big data analytics in agriculture is improving crop yields. By analyzing weather patterns, soil conditions, and other factors that affect crop growth, farmers and researchers can make more informed decisions about planting, harvesting, and managing crops, leading to increased yields. Big data analytics can also be used to improve the sustainability of agricultural practices. By analyzing data on soil conditions, water usage, and other factors, farmers and researchers can identify ways to reduce the environmental impact of farming practices. This can include reducing the use of fertilizers and pesticides, improving irrigation practices, and identifying alternative crops that are better suited to specific regions and weather patterns.

Another important application of big data analytics in agriculture is improving efficiency. By analyzing data on crop yields, weather patterns, and other factors, farmers and researchers can identify ways to reduce costs and increase productivity. This can include optimizing planting and harvesting times, improving irrigation and fertilization practices, and identifying new technologies and techniques that can be used to improve crop yields.

Big data analytics can also be used to improve the traceability and transparency of agricultural products. By analyzing data on crop growth, weather patterns, and other factors, farmers and researchers can ensure that agricultural products are safe, high-quality, and sustainable. This can include identifying risks such as pests and diseases,

and ensuring that agricultural products are produced in a way that is safe for the environment and for human health.

Summary

Big data analytics has the potential to revolutionize the agriculture industry by providing farmers and researchers with valuable insights into crop yields, weather patterns, soil conditions, and other critical factors that affect crop growth. By analyzing large amounts of data, farmers and researchers can make more informed decisions about planting, harvesting, and managing crops, which can lead to increased yields, reduced costs, and a more sustainable future for the agriculture industry. This chapter provides a general overview of the role and applications of big data analytics in agriculture and the various data sources, techniques and algorithms used in big data analytics in agriculture.

Big Data Analytics on Kisan Call Center Data

Kisan Call Center (KCC) is a toll-free service provided by the government of India to assist farmers with information on various government schemes, weather forecast, market prices, etc. The call center receives a large number of calls on a daily basis, and the data generated from these calls can be analyzed to gain valuable insights.

In this lecture, we will discuss the process of performing big data analytics on KCC data using Python programming language. We will cover the following topics:

1. Data Collection
2. Data Cleaning and Preprocessing
3. Data Exploration
4. Data Analysis
5. Data Visualization
6. Data Collection:

The first step in the big data analytics process is to collect the data. In the case of KCC data, the data can be collected from the call center's database. The data can be in the form of call logs, which includes information such as the caller's address, the date and time of the call, the duration of the call, etc.

Example:

```
import pandas as pd

# Read the data from a CSV file
data = pd.read_csv("KCC_data.csv")
```

2. Data Cleaning and Preprocessing:

The next step is to clean and preprocess the data. This step is essential to ensure that the data is in a format that can be easily analyzed. Data cleaning can include tasks such as removing missing or duplicate data, and converting data into a consistent

format. Data preprocessing can include tasks such as normalizing the data and removing irrelevant information.

Example:

```
# Removing missing values
data = data.dropna()

# Removing duplicate rows
data = data.drop_duplicates()
```

3. Data Exploration:

Once the data is cleaned and preprocessed, the next step is to explore the data. This step is essential to understand the structure and characteristics of the data. Data exploration can include tasks such as calculating summary statistics, identifying patterns and trends, and identifying outliers.

Example:

```
# Calculating summary statistics
data.describe()

# Identifying patterns and trends
data.groupby("call_reason").size()

# Identifying outliers
data[data.duration > data.duration.mean() + 3*data.duration.std()]
```

4. Data Analysis:

The next step is to perform data analysis. This step is essential to extract insights and information from the data. Data analysis can include tasks such as building predictive models, performing statistical tests, and identifying correlations.

Example:

```
# Building a predictive model
from sklearn.linear_model import LinearRegression

X = data[["duration"]]
y = data["call_reason"]

model = LinearRegression()
model.fit(X, y)

# Perform statistical tests
from scipy.stats import ttest_ind

call_reason1 = data[data["call_reason"] == "Weather Forecast"]
call_reason2 = data[data["call_reason"] == "Market Prices"]

ttest_ind(call_reason1["duration"], call_reason2["duration"])

# Identifying correlations
data.corr()
```

5. Data Visualization:

The final step is to visualize the data. This step is essential to communicate the insights and information obtained from the data analysis. Data visualization can include tasks such as creating charts, graphs, and maps. There are many libraries in python such as Matplotlib, seaborn, and plotly which can be used for data visualization.

Example:

```
import matplotlib.pyplot as plt

# Creating a bar chart
data.groupby("call_reason").size().plot(kind="bar")
plt.xlabel("Call Reason")
plt.ylabel("Number of Calls")
plt.show()

# Creating a scatter plot
plt.scatter(data["duration"], data["call_reason"])
plt.xlabel("Call Duration (in seconds)")
plt.ylabel("Call Reason")
plt.show()
```

Conclusion:

In this lecture, we discussed the process of performing big data analytics on Kisan Call Center data using python programming language. We covered the steps of data collection, cleaning, preprocessing, exploration, analysis and visualization. By utilizing the python libraries and techniques discussed in this lecture, we can gain valuable insights and make informed decisions based on the KCC data.

Web Development and API binding using Flask Framework

Sanchita Naha

ICAR-Indian Agricultural Statistics Research Institute, New Delhi - 110 012

sanchita.naha@icar.gov.in

Introduction:

Python is not only used by data analysts for AI or machine learning, but its flexibility makes it popular among the developers. It has been used for developing many powerful and professional web applications which are quick-loading and secure for example, Netflix, You Tube, Dropbox, Reddit, Instagram etc. For web development using Python, Django and flask are the two most popular frameworks. In this tutorial we will be restricted to only Flask framework for web development.

Web development is basically creation of applications that runs over the Internet i.e., websites. It involves both building and maintenance of websites. To understand Web development in a clearer way let us first look at the following client server architecture. The client-server architecture or model has other systems connected over a network where resources are shared among the different computers.

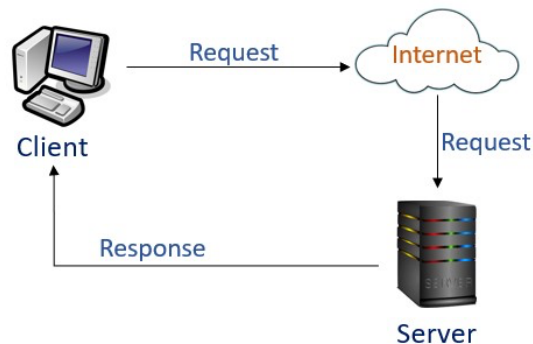


Figure 1: Client Server Web Architecture

Client server architecture is a computing model in which the client request for resources which is provided by the server as a response to the request. The client-server architecture has other systems connected over a network where resources are shared among the different computers. Clients are often situated at personal computers and servers are located elsewhere in the network, usually on a more powerful machine. All requests and services are delivered over a network that is connected to the Internet. Clients access these servers through the browser. User client types URL (Uniform Resource Locator) of a website in the browser and the browser sends over an HTTP/HTTPS request to WEB Server's IP. Then the web server send over the requested files to the client and the client side browser render the files and displays the website. This client-side browser interface is generally designed by the markup language HTML, styled with CSS and JavaScript. For the server-side scripting web frameworks of respective languages are used e.g., Java, C++, PHP,

Python etc. Flask is used for python-based web development as the server-side scripting language.

Flask is a microframework for developers, designed to enable them to create and scale web apps quickly and simply. It has a built-in development server and a debugger. Python 2.6 version or higher is required for installation of Flask. First, we must start with creation of a virtual environment. Virtual environment allows the user to create multiple environments simultaneously so as to avoid compatibility issues between the different versions of the libraries. The following command is used for installing the virtual environment:

```
pip install virtualenv
```

Use the following command to activate the environment.

```
venv\scripts\activate
```

Once the virtual environment is activated, system is ready for Flask installation. In PyCharm editor create a new project and select the virtual environment then go to 'Terminal' and run the following command:

```
pip install flask
```

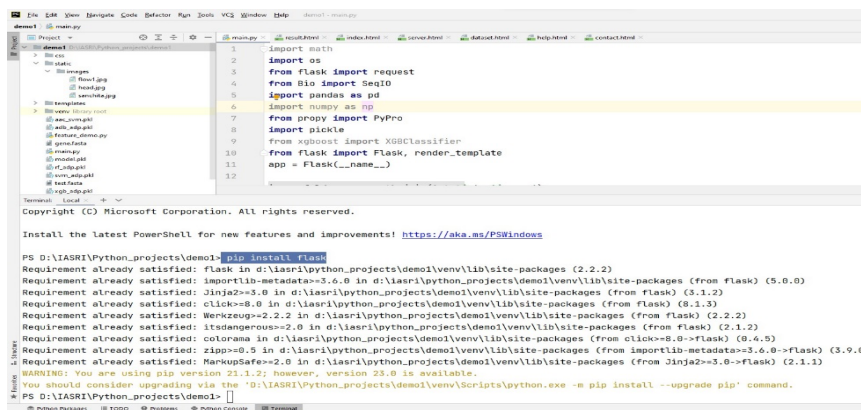


Figure 2: Screenshot of PyCharm for Flask installation

The directory structure of the project will look like the following:

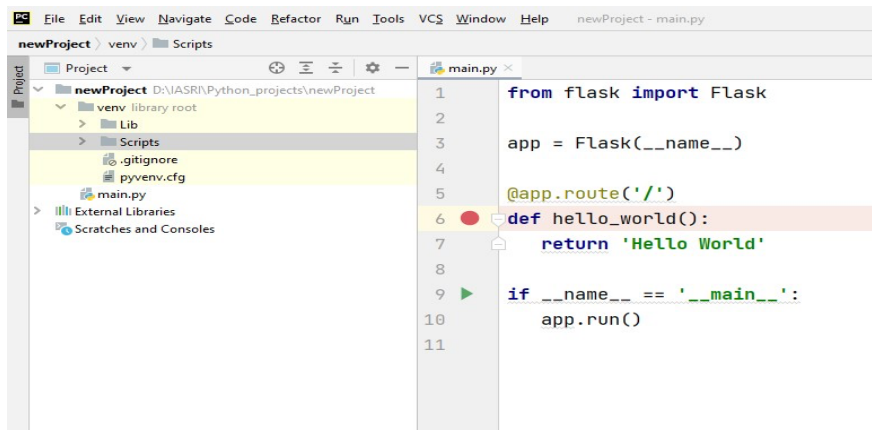


Figure 3: Screenshot of PyCharm project directory with virtual environment

Then in the main.py file execute the following code:

```
from flask import Flask
app = Flask(__name__)
@app.route('/')
def hello_world():
    return 'Hello World'
if __name__ == '__main__':
    app.run()
```

If flask is properly installed in the system, the output looks like following:

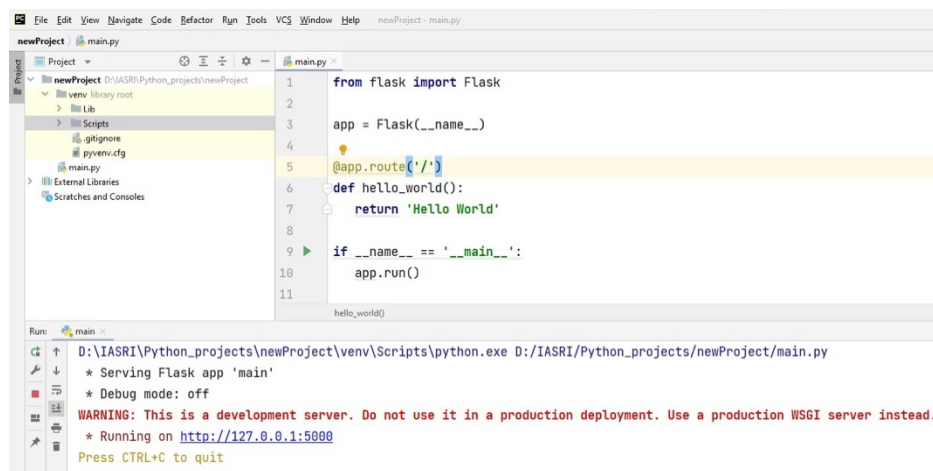


Figure 4: Screenshot of PyCharm showing successful debugging of a web project.

Click on the above URL (localhost:5000) and it will open in the default web browser. 'Hello World' message will be displayed on it.

Flask application starts with the `app.run()` method within the main method. While developing the server, after every change the server must be manually restarted to reflect the change. To avoid this inconvenience, enable debug support. The server will then reload itself if the code changes. It will also provide a useful debugger to track the errors if any, in the application as following:

```
if __name__ == '__main__':
    app.run(debug=True)
```

The Debug mode is enabled by setting the debug property to True, before running or passing the debug parameter to the `run()` method.

The `@app.route('/')` decorator allows to bind the URL to a function. It is useful to access the desired page directly without having to navigate from the home page. For example:

```
@app.route('/hello')
```

```
def hello_world():  
    return 'hello world'
```

Here, URL `‘/hello’` is bound to the `hello_world()` function. As a result, if a user visit `http://localhost:5000/hello`, the output of the `hello_world()` function will be rendered in the browser which is in this case the message `‘hello world’`. In the URL we can also bind dynamic variable values. This variable part is marked as `<variable_name>`. It is passed as a keyword argument to the function with which the rule is associated. As in the following example, the rule parameter of `route()` decorator contains `<name>` variable part attached to URL `‘/hello’`. Hence, if the `http://localhost:5000/hello/john` is entered as a URL in the browser, then `‘john’` will be supplied to `hello()` function as argument and it will display the corresponding message.

```
from flask import Flask  
app = Flask(__name__)  
@app.route('/hello/<name>')  
def hello_name(name):  
    return 'Hello %s!' % name  
if __name__ == '__main__':  
    app.run(debug = True)
```

The code and the output will look as following:

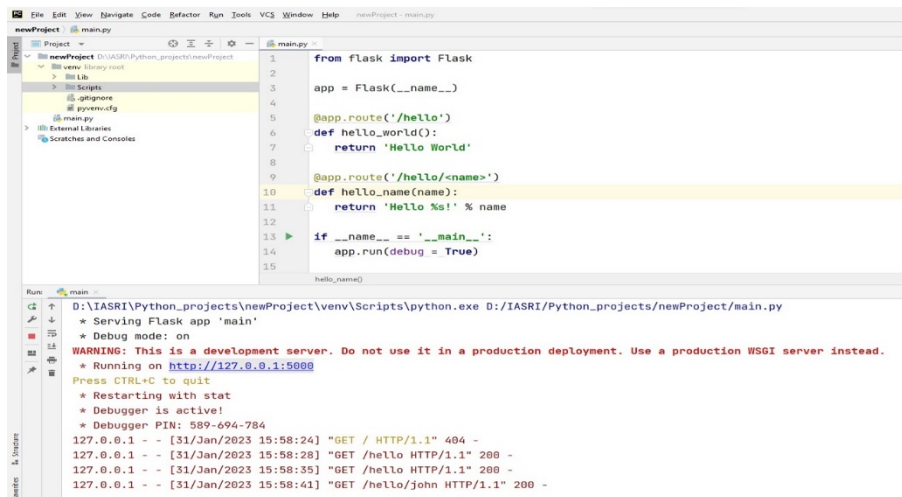


Figure 5: Screenshot of PyCharm editor displaying URL binding

Output:

Type `‘127.0.0.1:5000/hello’` in the browser’s address bar, it will show the following output:

`‘Hello World’`

Type '127.0.0.1:5000/hello/john' in the browser's address bar, it will show the following output:

'Hello john'

HTML (Hyper Text Markup Language) is the standard language for web designing which follows the HTTP (Hyper Text Transfer Protocol) protocol for data communication. To develop dynamic web pages, static HTML pages have to be inserted into the .py files. Python uses two HTTP request methods GET and POST requests for client server communication. GET method is used to request data from the server and POST method is used to submit data to be processed at the server.

At first create a folder called 'templates' within the project directory for storing all the .html files. Within the templates folder create 'login.html' file for the GET method.

login.html

```
<html>
  <body>
    <form action = "http://localhost:5000/data" method =
"get">
      <table>
        <tr><td>Name</td>
        <td><input type="text" name="uname"></td></tr>
        <tr><td>Password</td>
        <td><input type = "password" name = "pass"></td></tr>
        <tr><td><input type = "submit"></td></tr>
      </table>
    </form>
  </body>
</html>
```

With the above code 'login.html' will display a login page for accepting two user inputs called name and password. To call this html page we need to bind this html page with a python function using the @app.route() decorator and render the html page within the same function like following.

main.py:

```
from flask import render_template
@app.route('/login',methods = ['GET'])
def login():
    return render_template('login.html')
```

To display the user input data after form submission, another function has to be created. Suppose we name the function as data and add it to the URL binder using `@app.route()` as following:

main.py:

```
@app.route('/data,methods = ['GET'])
def data():
    uname=request.args.get('uname')
    passwd=request.args.get('pass')
    if uname=="john" and passwd=="smith":
        return "Welcome %s" %uname
```

Output:

URL: <http://127.0.0.1:5000/data?uname=john&pass=smith>

Welcome john

In the above output it is observed that using the GET method user parameter values are placed in the URL which can cause security issues. On the client side, browser history files containing sensitive GET requests are easily recoverable, especially when an unauthorized user gains access to the system. To make it more secure POST request is to be used. The POST request is to be used as the following code and every other URL binding will remain same. To make it more clear previous login.html has been modified as login_p.html and as well the data() has been modified as data_p():

login_p.html:

```
<!DOCTYPE html>
<html>
    <body>
        <center>
            <form action="http://localhost:5000/data_p"
method="post">
                <p>Enter Name:</p>
```

```
<p><input type = "text" name = "nm" /></p>
<p><input type = "submit" value = "submit" /></p>
</form>
</center>
</body>
</html>
```

main.py:

```
from flask import Flask, render_template, redirect, url_for
from flask import request
@app.route('/data_p', methods = ['POST', 'GET'])
def data_p():
    if request.method == 'POST':
        user = request.form['nm']
        return "Welcome %s" %user
```

Similarly we can link other .html pages using the URL binding method where we have to add the desirable .html page to any function which has to called by the `@app.route('/')`.

