GraphicEra
Deemed to be University
Accredited by NAAC with Grade A
NBA Accredited Programs in ECE, CSE & ME
Approved by AICTE, Ministry of HRD, Govt. of India
DEHRADUN, UTTARAKHAND, INDIA

Machine Learning with Python

Machine Learning (ML) is basically that field of computer science with the help of which computer systems can provide sense to data in much the same way as human beings do. In simple words, ML is a type of artificial intelligence that extract patterns out of raw data by using an algorithm or method. The key focus of ML is to allow computer systems to learn from experience without being explicitly programmed or human intervention.

Audience

This tutorial will be useful for graduates, postgraduates, and research students who either have an interest in this subject or have this subject as a part of their curriculum. The reader can be a beginner or an advanced learner. This tutorial has been prepared for the students as well as professionals to ramp up quickly. This tutorial is a stepping stone to your Machine Learning journey.

Prerequisites

The reader must have basic knowledge of artificial intelligence. He/she should also be aware of Python, NumPy, Scikit-learn, Scipy, Matplotlib. If you are new to any of these concepts, we recommend you to take up tutorials concerning these topics, before you dig further into this tutorial.

We are living in the ‘age of data’ that is enriched with better computational power and more storage resources,. This data or information is increasing day by day, but the real challenge is to make sense of all the data. Businesses & organizations are trying to deal with it by building intelligent systems using the concepts and methodologies from Data science, Data Mining and Machine learning. Among them, machine learning is the most exciting field of computer science. It would not be wrong if we call machine learning the application and science of algorithms that provides sense to the data.

What is Machine Learning?

Machine Learning (ML) is that field of computer science with the help of which computer systems can provide sense to data in much the same way as human beings do.

In simple words, ML is a type of artificial intelligence that extract patterns out of raw data by using an algorithm or method. The main focus of ML is to allow computer systems learn from experience without being explicitly programmed or human intervention.

Need for Machine Learning

Human beings, at this moment, are the most intelligent and advanced species on earth because they can think, evaluate and solve complex problems. On the other side, AI is still in its initial stage and haven’t surpassed human intelligence in many aspects. Then the question is that what is the need to make machine learn? The most suitable reason for doing this is, “to make decisions, based on data, with efficiency and scale”.

Lately, organizations are investing heavily in newer technologies like Artificial Intelligence, Machine Learning and Deep Learning to get the key information from data to perform several real-world tasks and solve problems. We can call it data-driven decisions taken by machines, particularly to automate the process. These data-driven decisions can be used, instead of using programing logic, in the problems that cannot be programmed inherently. The fact is that we can’t do without human intelligence, but other aspect is that we all need to solve real-world problems with efficiency at a huge scale. That is why the need for machine learning arises.

Why & When to Make Machines Learn?

We have already discussed the need for machine learning, but another question arises that in what scenarios we must make the machine learn? There can be several circumstances where we need machines to take data-driven decisions with efficiency and at a huge scale. The followings are some of such circumstances where making machines learn would be more effective −

Lack of human expertise

The very first scenario in which we want a machine to learn and take data-driven decisions, can be the domain where there is a lack of human expertise. The examples can be navigations in unknown territories or spatial planets.

Dynamic scenarios

There are some scenarios which are dynamic in nature i.e. they keep changing over time. In case of these scenarios and behaviors, we want a machine to learn and take data-driven decisions. Some of the examples can be network connectivity and availability of infrastructure in an organization.

Difficulty in translating expertise into computational tasks

There can be various domains in which humans have their expertise,; however, they are unable to translate this expertise into computational tasks. In such circumstances we want machine learning. The examples can be the domains of speech recognition, cognitive tasks etc.

Machine Learning Model

Before discussing the machine learning model, we must need to understand the following formal definition of ML given by professor Mitchell −

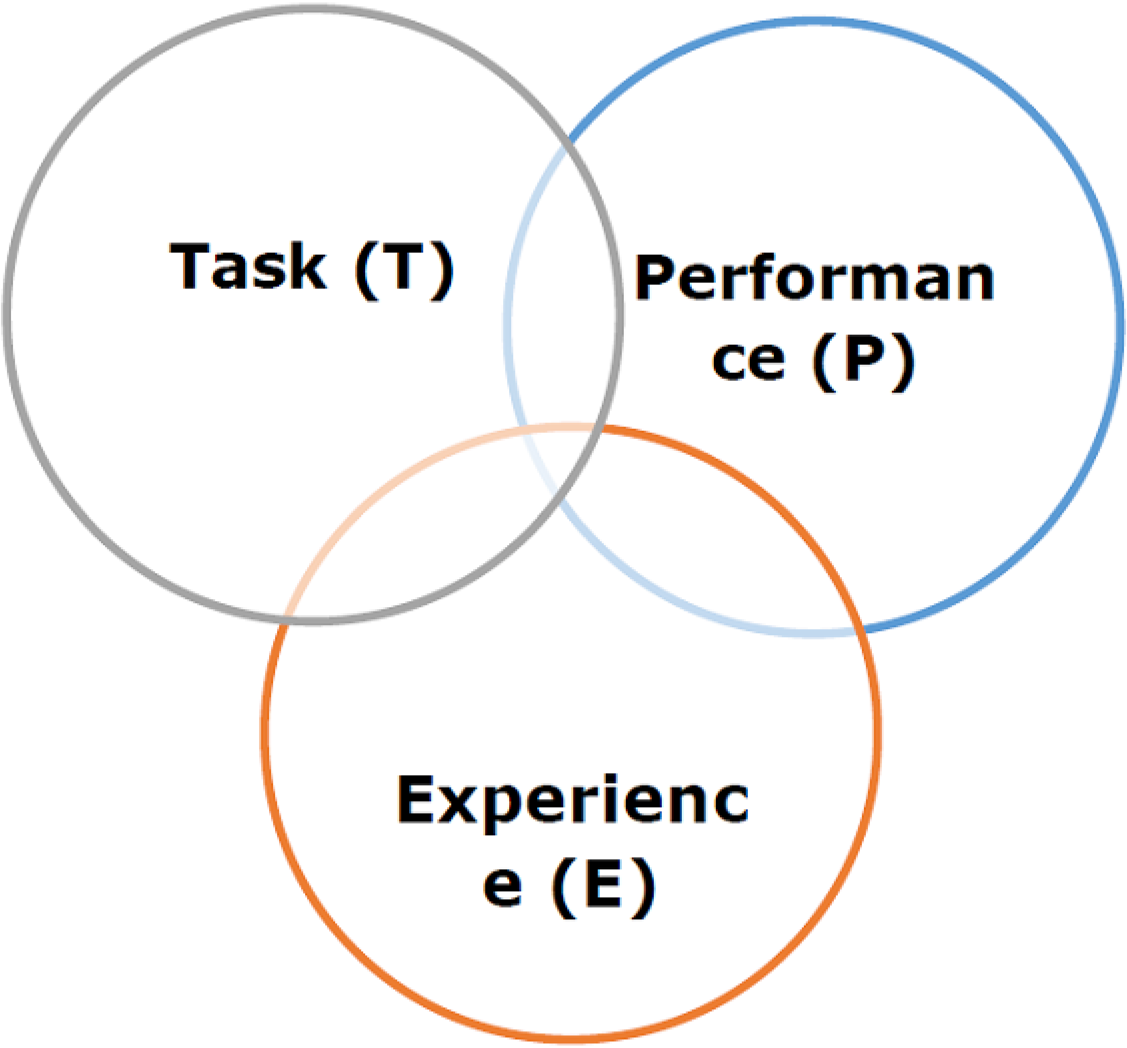
“A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.”

The above definition is basically focusing on three parameters, also the main components of any learning algorithm, namely Task(T), Performance(P) and experience (E). In this context, we can simplify this definition as −

ML is a field of AI consisting of learning algorithms that −

- Improve their performance (P)
- At executing some task (T)
- Over time with experience (E)

Based on the above, the following diagram represents a Machine Learning Model −



Let us discuss them more in detail now −

Task(T)

From the perspective of problem, we may define the task T as the real-world problem to be solved. The problem can be anything like finding best house price in a specific location or to find best marketing strategy etc. On the other hand, if we talk about machine learning, the definition of task is different because it is difficult to solve ML based tasks by conventional programming approach.

A task T is said to be a ML based task when it is based on the process and the system must follow for operating on data points. The examples of ML based tasks are Classification, Regression, Structured annotation, Clustering, Transcription etc.

Experience (E)

As name suggests, it is the knowledge gained from data points provided to the algorithm or model. Once provided with the dataset, the model will run iteratively and will learn some inherent pattern. The learning thus acquired is called experience(E). Making an analogy with human learning, we can think of this situation as in which a human being is learning or gaining some experience from various attributes like situation, relationships etc. Supervised, unsupervised and reinforcement learning are some ways to learn or gain experience. The experience gained by out ML model or algorithm will be used to solve the task T.

Performance (P)

An ML algorithm is supposed to perform task and gain experience with the passage of time. The measure which tells whether ML algorithm is performing as per expectation or not is its performance (P). P is basically a quantitative metric that tells how a model is performing the task, T, using its experience, E. There are many metrics that help to understand the ML performance, such as accuracy score, F1 score, confusion matrix, precision, recall, sensitivity etc.

Challenges in Machines Learning

While Machine Learning is rapidly evolving, making significant strides with cybersecurity and autonomous cars, this segment of AI as whole still has a long way to go. The reason behind is that ML has not been able to overcome number of challenges. The challenges that ML is facing currently are −

Quality of data − Having good-quality data for ML algorithms is one of the biggest challenges. Use of low-quality data leads to the problems related to data preprocessing and feature extraction.

Time-Consuming task − Another challenge faced by ML models is the consumption of time especially for data acquisition, feature extraction and retrieval.

Lack of specialist persons − As ML technology is still in its infancy stage, availability of expert resources is a tough job.

No clear objective for formulating business problems − Having no clear objective and well-defined goal for business problems is another key challenge for ML because this technology is not that mature yet.

Issue of overfitting & underfitting − If the model is overfitting or underfitting, it cannot be represented well for the problem.

Curse of dimensionality − Another challenge ML model faces is too many features of data points. This can be a real hindrance.

Difficulty in deployment − Complexity of the ML model makes it quite difficult to be deployed in real life.

Applications of Machines Learning

Machine Learning is the most rapidly growing technology and according to researchers we are in the golden year of AI and ML. It is used to solve many real-world complex problems which cannot be solved with traditional approach. Following are some real-world applications of ML −

- Emotion analysis
- Sentiment analysis
- Error detection and prevention
- Weather forecasting and prediction
- Stock market analysis and forecasting
- Speech synthesis
- Speech recognition
- Customer segmentation
- Object recognition
- Fraud detection
- Fraud prevention
- Recommendation of products to customer in online shopping.

An Introduction to Python

Python is a popular object-oriented programing language having the capabilities of high-level programming language. Its easy to learn syntax and portability capability makes it popular these days. The followings facts gives us the introduction to Python −

- Python was developed by Guido van Rossum at Stichting Mathematisch Centrum in the Netherlands.
- It was written as the successor of programming language named ‘ABC’.
- It’s first version was released in 1991.
- The name Python was picked by Guido van Rossum from a TV show named Monty Python’s Flying Circus.
- It is an open source programming language which means that we can freely download it and use it to develop programs. It can be downloaded from www.python.org.
- Python programming language is having the features of Java and C both. It is having the elegant ‘C’ code and on the other hand, it is having classes and objects like Java for object-oriented programming.
- It is an interpreted language, which means the source code of Python program would be first converted into bytecode and then executed by Python virtual machine.

Strengths and Weaknesses of Python

Every programming language has some strengths as well as weaknesses, so does Python too.

Strengths

According to studies and surveys, Python is the fifth most important language as well as the most popular language for machine learning and data science. It is because of the following strengths that Python has −

Easy to learn and understand − The syntax of Python is simpler; hence it is relatively easy, even for beginners also, to learn and understand the language.

Multi-purpose language − Python is a multi-purpose programming language because it supports structured programming, object-oriented programming as well as functional programming.

Huge number of modules − Python has huge number of modules for covering every aspect of programming. These modules are easily available for use hence making Python an extensible language.

Support of open source community − As being open source programming language, Python is supported by a very large developer community. Due to this, the bugs are easily fixed by the Python community. This characteristic makes Python very robust and adaptive.

Scalability − Python is a scalable programming language because it provides an improved structure for supporting large programs than shell-scripts.

Weakness

Although Python is a popular and powerful programming language, it has its own weakness of slow execution speed.

The execution speed of Python is slow as compared to compiled languages because Python is an interpreted language. This can be the major area of improvement for Python community.

Installing Python

For working in Python, we must first have to install it. You can perform the installation of Python in any of the following two ways −

- Installing Python individually
- Using Pre-packaged Python distribution − Anaconda

Let us discuss these each in detail.

Installing Python Individually

If you want to install Python on your computer, then then you need to download only the binary code applicable for your platform. Python distribution is available for Windows, Linux and Mac platforms.

The following is a quick overview of installing Python on the above-mentioned platforms −

On Unix and Linux platform

With the help of following steps, we can install Python on Unix and Linux platform −

- First, go to www.python.org/downloads/.
- Next, click on the link to download zipped source code available for Unix/Linux.
- Now, Download and extract files.
- Next, we can edit the Modules/Setup file if we want to customize some options.
    - Next, write the command run ./configure script
    - make
    - make install

On Windows platform

With the help of following steps, we can install Python on Windows platform −

- First, go to www.python.org/downloads/.
- Next, click on the link for Windows installer python-XYZ.msi file. Here XYZ is the version we wish to install.
- Now, we must run the file that is downloaded. It will take us to the Python install wizard, which is easy to use. Now, accept the default settings and wait until the install is finished.

On Macintosh platform

For Mac OS X, Homebrew, a great and easy to use package installer is recommended to install Python 3. In case if you don't have Homebrew, you can install it with the help of following command −

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

It can be updated with the command below −

```
$ brew update
```

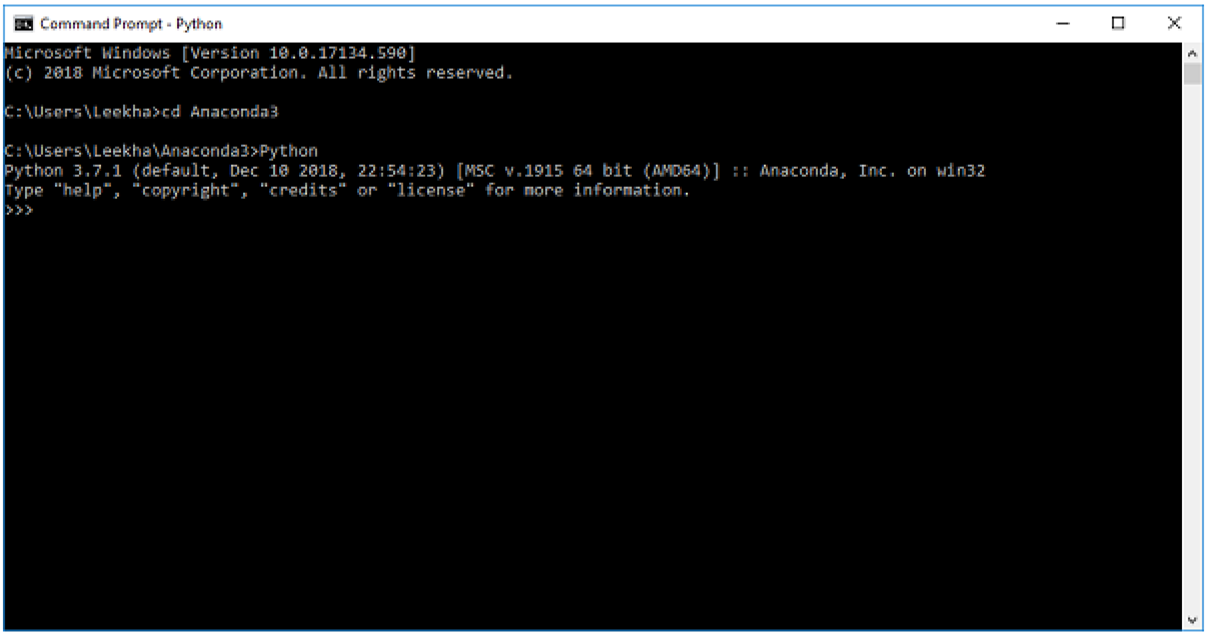
Now, to install Python3 on your system, we need to run the following command −

```
$ brew install python3
```

Using Pre-packaged Python Distribution: Anaconda

Anaconda is a packaged compilation of Python which have all the libraries widely used in Data science. We can follow the following steps to setup Python environment using Anaconda −

- Step 1 − First, we need to download the required installation package from Anaconda distribution. The link for the same is www.anaconda.com/distribution/. You can choose from Windows, Mac and Linux OS as per your requirement.
- Step 2 − Next, select the Python version you want to install on your machine. The latest Python version is 3.7. There you will get the options for 64-bit and 32-bit Graphical installer both.
- Step 3 − After selecting the OS and Python version, it will download the Anaconda installer on your computer. Now, double click the file and the installer will install Anaconda package.
- Step 4 − For checking whether it is installed or not, open a command prompt and type Python as follows −



```
Microsoft Windows [Version 10.0.17134.590]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\Leekha>cd Anaconda3

C:\Users\Leekha\Anaconda3>Python
Python 3.7.1 (default, Dec 10 2018, 22:54:23) [MSC v.1915 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

You can also check this in detailed video lecture at www.tutorialspoint.com/python_essentials_online_training/getting_started_with_anaconda.asp.

Why Python for Data Science?

Python is the fifth most important language as well as most popular language for Machine learning and data science. The following are the features of Python that makes it the preferred choice of language for data science −

Extensive set of packages

Python has an extensive and powerful set of packages which are ready to be used in various domains. It also has packages like numpy, scipy, pandas, scikit-learn etc. which are required for machine learning and data science.

Easy prototyping

Another important feature of Python that makes it the choice of language for data science is the easy and fast prototyping. This feature is useful for developing new algorithm.

Collaboration feature

The field of data science basically needs good collaboration and Python provides many useful tools that make this extremely.

One language for many domains

A typical data science project includes various domains like data extraction, data manipulation, data analysis, feature extraction, modelling, evaluation, deployment and updating the solution. As Python is a multi-purpose language, it allows the data scientist to address all these domains from a common platform.

Components of Python ML Ecosystem

In this section, let us discuss some core Data Science libraries that form the components of Python Machine learning ecosystem. These useful components make Python an important language for Data Science. Though there are many such components, let us discuss some of the importance components of Python ecosystem here −

Jupyter Notebook

Jupyter notebooks basically provides an interactive computational environment for developing Python based Data Science applications. They are formerly known as ipython notebooks. The following are some of the features of Jupyter notebooks that makes it one of the best components of Python ML ecosystem −

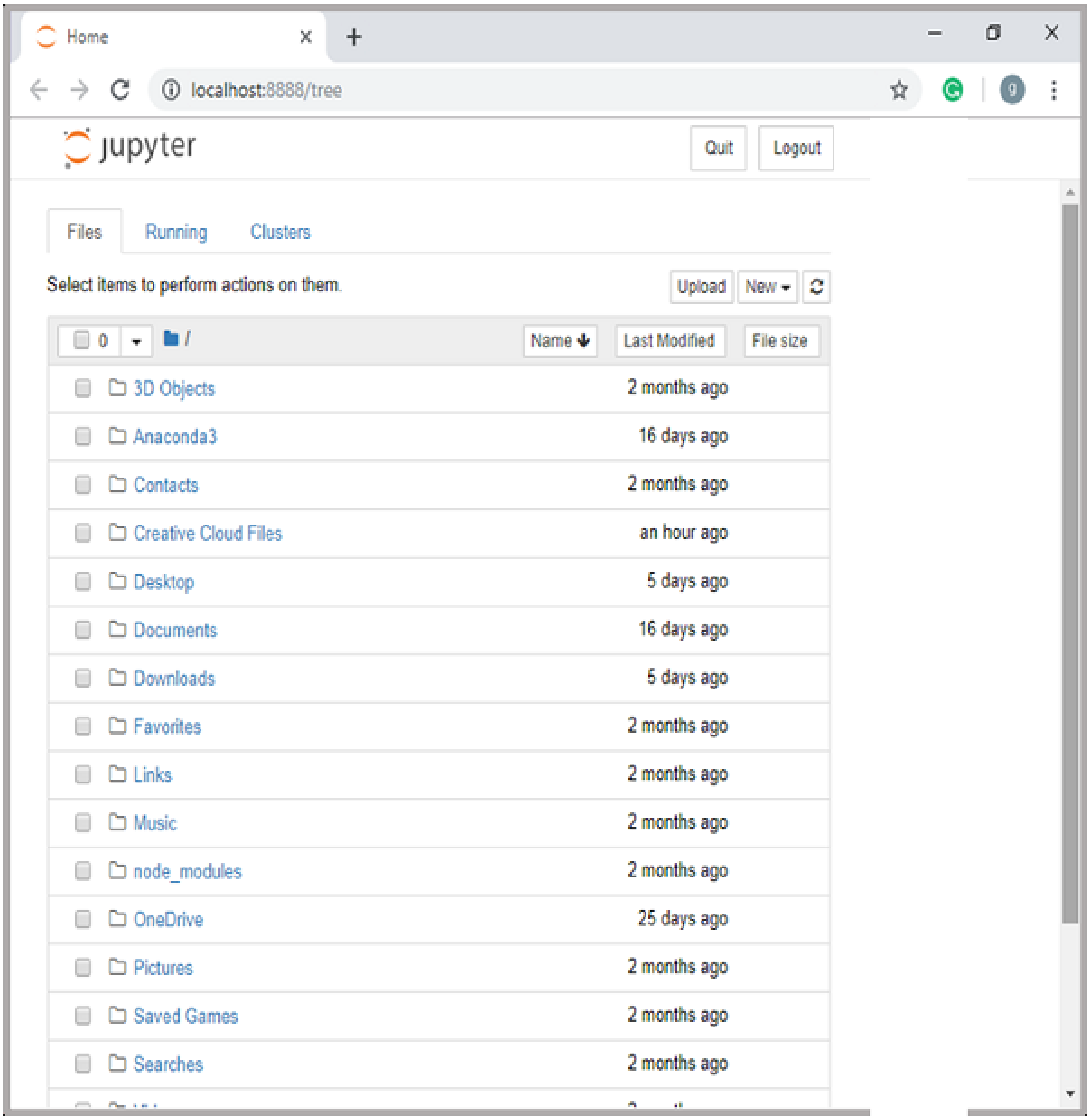
- Jupyter notebooks can illustrate the analysis process step by step by arranging the stuff like code, images, text, output etc. in a step by step manner.
- It helps a data scientist to document the thought process while developing the analysis process.
- One can also capture the result as the part of the notebook.
- With the help of jupyter notebooks, we can share our work with a peer also.

Installation and Execution

If you are using Anaconda distribution, then you need not install jupyter notebook separately as it is already installed with it. You just need to go to Anaconda Prompt and type the following command −

```
C:\>jupyter notebook
```

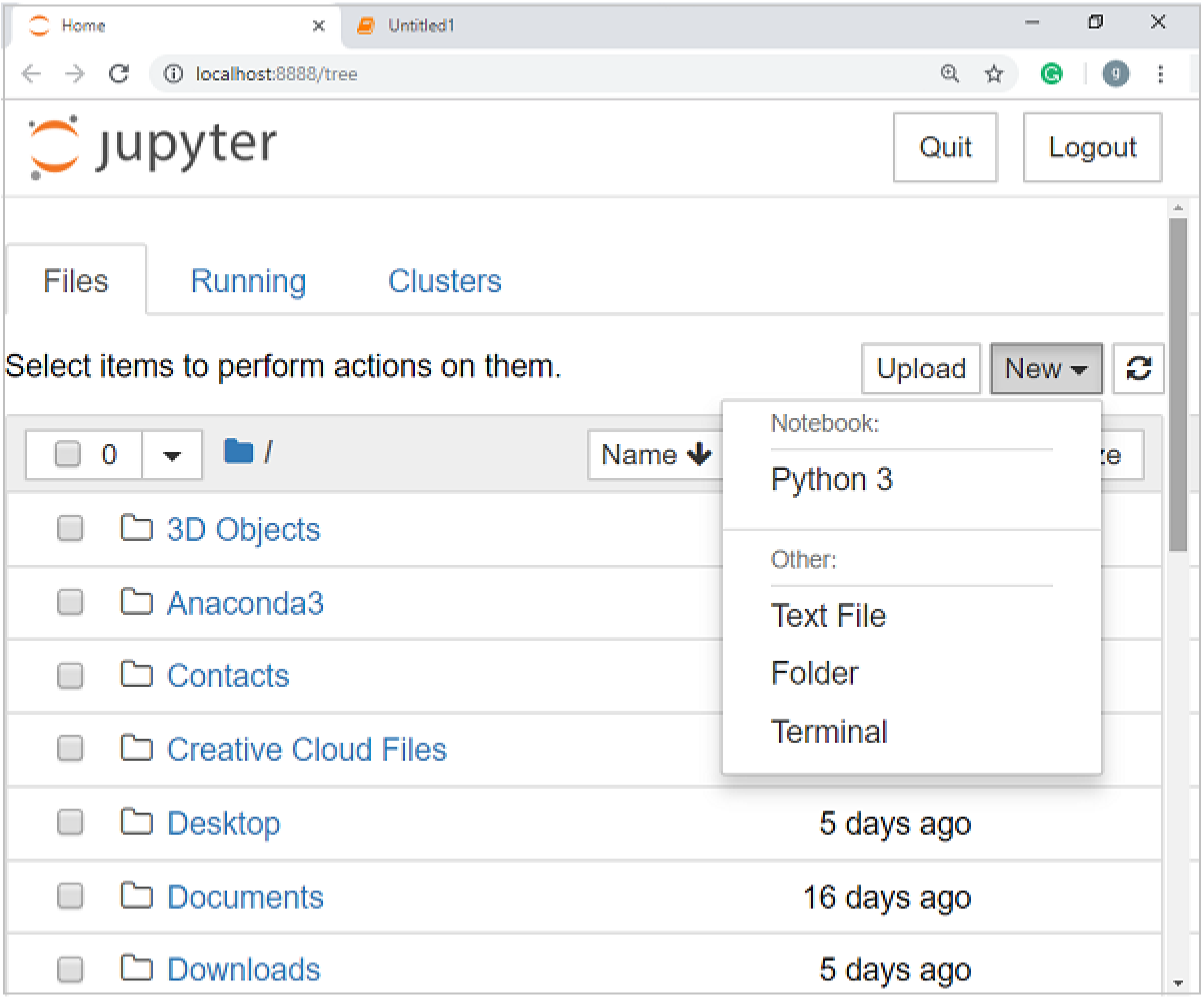
After pressing enter, it will start a notebook server at localhost:8888 of your computer. It is shown in the following screen shot −



Select items to perform actions on them.

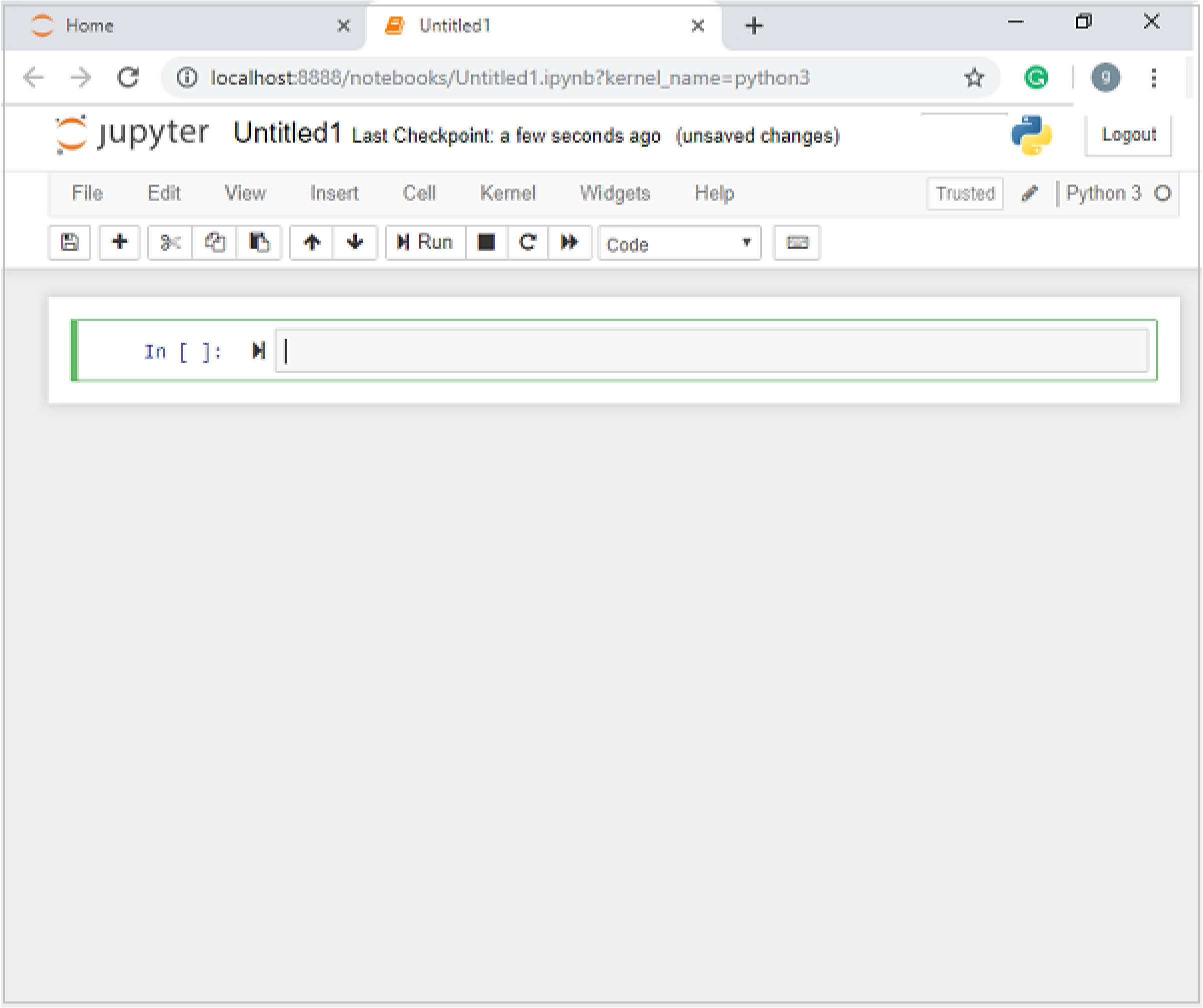
Name	Last Modified	File size
□ 3D Objects	2 months ago	
□ Anaconda3	16 days ago	
□ Contacts	2 months ago	
□ Creative Cloud Files	an hour ago	
□ Desktop	5 days ago	
□ Documents	16 days ago	
□ Downloads	5 days ago	
□ Favorites	2 months ago	
□ Links	2 months ago	
□ Music	2 months ago	
□ node_modules	2 months ago	
□ OneDrive	25 days ago	
□ Pictures	2 months ago	
□ Saved Games	2 months ago	
□ Searches	2 months ago	

Now, after clicking the New tab, you will get a list of options. Select Python 3 and it will take you to the new notebook for start working in it. You will get a glimpse of it in the following screenshots −



Select items to perform actions on them.

3D Objects	
Anaconda3	
Contacts	
Creative Cloud Files	
Desktop	5 days ago
Documents	16 days ago
Downloads	5 days ago


Home
Untitled1
localhost:8888/notebooks/Untitled1.ipynb?kernel_name=python3
jupyter
Untitled1 Last Checkpoint: a few seconds ago (unsaved changes)
Logout
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Trusted
Python 3
Run
Code
In [ ]:

On the other hand, if you are using standard Python distribution then jupyter notebook can be installed using popular python package installer, pip.

```
pip install jupyter
```

Types of Cells in Jupyter Notebook

The following are the three types of cells in a jupyter notebook −

Code cells − As the name suggests, we can use these cells to write code. After writing the code/content, it will send it to the kernel that is associated with the notebook.

Markdown cells − We can use these cells for notating the computation process. They can contain the stuff like text, images, Latex equations, HTML tags etc.

Raw cells − The text written in them is displayed as it is. These cells are basically used to add the text that we do not wish to be converted by the automatic conversion mechanism of jupyter notebook.

For more detailed study of jupyter notebook, you can go to the link www.tutorialspoint.com/jupyter/index.htm.

NumPy

It is another useful component that makes Python as one of the favorite languages for Data Science. It basically stands for Numerical Python and consists of multidimensional array objects. By using NumPy, we can perform the following important operations −

- Mathematical and logical operations on arrays.
- Fourier transformation
- Operations associated with linear algebra.

We can also see NumPy as the replacement of MatLab because NumPy is mostly used along with Scipy (Scientific Python) and Mat-plotlib (plotting library).

Installation and Execution

If you are using Anaconda distribution, then no need to install NumPy separately as it is already installed with it. You just need to import the package into your Python script with the help of following −

```
import numpy as np
```

On the other hand, if you are using standard Python distribution then NumPy can be installed using popular python package installer, pip.

```
pip install NumPy
```

For more detailed study of NumPy, you can go to the link www.tutorialspoint.com/numpy/index.htm.

Pandas

It is another useful Python library that makes Python one of the favorite languages for Data Science. Pandas is basically used for data manipulation, wrangling and analysis. It was developed by Wes McKinney in 2008. With the help of Pandas, in data processing we can accomplish the following five steps −

- Load
- Prepare
- Manipulate
- Model
- Analyze

Data representation in Pandas

The entire representation of data in Pandas is done with the help of following three data structures −

Series − It is basically a one-dimensional ndarray with an axis label which means it is like a simple array with homogeneous data. For example, the following series is a collection of integers 1,5,10,15,24,25...

1	5	10	15	24	25	28	36	40	89

Data frame − It is the most useful data structure and used for almost all kind of data representation and manipulation in pandas. It is basically a two-dimensional data structure which can contain heterogeneous data. Generally, tabular data is represented by using data frames. For example, the following table shows the data of students having their names and roll numbers, age and gender −

Name	Roll number	Age	Gender
Aarav	1	15	Male
Harshit	2	14	Male
Kanika	3	16	Female
Mayank	4	15	Male

Panel − It is a 3-dimensional data structure containing heterogeneous data. It is very difficult to represent the panel in graphical representation, but it can be illustrated as a container of DataFrame.

The following table gives us the dimension and description about above mentioned data structures used in Pandas −

Data Structure	Dimension	Description
Series	1-D	Size immutable, 1-D homogeneous data

DataFrames	2-D	Size Mutable, Heterogeneous data in tabular form
Panel	3-D	Size-mutable array, container of DataFrame.

We can understand these data structures as the higher dimensional data structure is the container of lower dimensional data structure.

Installation and Execution

If you are using Anaconda distribution, then no need to install Pandas separately as it is already installed with it. You just need to import the package into your Python script with the help of following −

```
import pandas as pd
```

On the other hand, if you are using standard Python distribution then Pandas can be installed using popular python package installer, pip.

```
pip install Pandas
```

After installing Pandas, you can import it into your Python script as did above.

Example

The following is an example of creating a series from ndarray by using Pandas −

```
In [1]: import pandas as pd

In [2]: import numpy as np

In [3]: data = np.array(['g','a','u','r','a','v'])

In [4]: s = pd.Series(data)

In [5]: print (s)

0 g
1 a
2 u
3 r
4 a
5 v
```

```
dtype: object
```

For more detailed study of Pandas you can go to the link www.tutorialspoint.com/python_pandas/index.htm.

Scikit-learn

Another useful and most important python library for Data Science and machine learning in Python is Scikit-learn. The following are some features of Scikit-learn that makes it so useful −

- It is built on NumPy, SciPy, and Matplotlib.
- It is an open source and can be reused under BSD license.
- It is accessible to everybody and can be reused in various contexts.
- Wide range of machine learning algorithms covering major areas of ML like classification, clustering, regression, dimensionality reduction, model selection etc. can be implemented with the help of it.

Installation and Execution

If you are using Anaconda distribution, then no need to install Scikit-learn separately as it is already installed with it. You just need to use the package into your Python script. For example, with following line of script we are importing dataset of breast cancer patients from Scikit-learn −

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

On the other hand, if you are using standard Python distribution and having NumPy and SciPy then Scikit-learn can be installed using popular python package installer, pip.

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

After installing Scikit-learn, you can use it into your Python script as you have done above.

There are various ML algorithms, techniques and methods that can be used to build models for solving real-life problems by using data. In this chapter, we are going to discuss such different kinds of methods.

Different Types of Methods

The following are various ML methods based on some broad categories −

Based on human supervision

In the learning process, some of the methods that are based on human supervision are as follows −

Supervised Learning

Supervised learning algorithms or methods are the most commonly used ML algorithms. This method or learning algorithm take the data sample i.e. the training data and its associated output i.e. labels or responses with each data samples during the training process.

The main objective of supervised learning algorithms is to learn an association between input data samples and corresponding outputs after performing multiple training data instances.

For example, we have

x: Input variables and

Y: Output variable

Now, apply an algorithm to learn the mapping function from the input to output as follows −

Y=f(x)

Now, the main objective would be to approximate the mapping function so well that even when we have new input data (x), we can easily predict the output variable (Y) for that new input data.

It is called supervised because the whole process of learning can be thought as it is being supervised by a teacher or supervisor. Examples of supervised machine learning algorithms includes Decision tree, Random Forest, KNN, Logistic Regression etc.

Based on the ML tasks, supervised learning algorithms can be divided into following two broad classes −

- Classification
- Regression

Classification

The key objective of classification-based tasks is to predict categorial output labels or responses for the given input data. The output will be based on what the model has learned in training phase. As we know that the categorial output responses means unordered and discrete values, hence each output response will belong to a specific class or category. We will discuss Classification and associated algorithms in detail in the upcoming chapters also.

Regression

The key objective of regression-based tasks is to predict output labels or responses which are continues numeric values, for the given input data. The output will be based on what the model has learned in its training phase. Basically, regression models use the input data features (independent variables) and their corresponding continuous numeric output values (dependent or outcome variables) to learn specific association between inputs and corresponding outputs. We will discuss regression and associated algorithms in detail in further chapters also.

Unsupervised Learning

As the name suggests, it is opposite to supervised ML methods or algorithms which means in unsupervised machine learning algorithms we do not have any supervisor to provide any sort of guidance. Unsupervised learning algorithms are handy in the scenario in which we do not have the liberty, like in supervised learning algorithms, of having pre-labeled training data and we want to extract useful pattern from input data.

For example, it can be understood as follows −

Suppose we have −

x: Input variables, then there would be no corresponding output variable and the algorithms need to discover the interesting pattern in data for learning.

Examples of unsupervised machine learning algorithms includes K-means clustering, K-nearest neighbors etc.

Based on the ML tasks, unsupervised learning algorithms can be divided into following broad classes −

- Clustering
- Association
- Dimensionality Reduction

Clustering

Clustering methods are one of the most useful unsupervised ML methods. These algorithms used to find similarity as well as relationship patterns among data samples and then cluster those samples into groups having similarity based on features. The real-world example of clustering is to group the customers by their purchasing behavior.

Association

Another useful unsupervised ML method is Association which is used to analyze large dataset to find patterns which further represents the interesting relationships between various items. It is also termed as Association Rule Mining or Market basket analysis which is mainly used to analyze customer shopping patterns.

Dimensionality Reduction

This unsupervised ML method is used to reduce the number of feature variables for each data sample by selecting set of principal or representative features. A question arises here is that why we need to reduce the dimensionality? The reason behind is the problem of feature space complexity which arises when we start analyzing and extracting millions of features from data samples. This problem generally refers to "curse of dimensionality". PCA (Principal Component Analysis), K-nearest neighbors and discriminant analysis are some of the popular algorithms for this purpose.

Anomaly Detection

This unsupervised ML method is used to find out the occurrences of rare events or observations that generally do not occur. By using the learned knowledge, anomaly detection methods would be able to differentiate between anomalous or a normal data point. Some of the unsupervised algorithms like clustering, KNN can detect anomalies based on the data and its features.

Semi-supervised Learning

Such kind of algorithms or methods are neither fully supervised nor fully unsupervised. They basically fall between the two i.e. supervised and unsupervised learning methods. These kinds of algorithms generally use small supervised learning component i.e. small amount of pre-labeled annotated data and large unsupervised learning component i.e. lots of unlabeled data for training. We can follow any of the following approaches for implementing semi-supervised learning methods −

- The first and simple approach is to build the supervised model based on small amount of labeled and annotated data and then build the unsupervised model by applying the same to the large amounts of unlabeled data to get more labeled samples. Now, train the model on them and repeat the process.
- The second approach needs some extra efforts. In this approach, we can first use the unsupervised methods to cluster similar data samples, annotate these groups and then use a combination of this information to train the model.

Reinforcement Learning

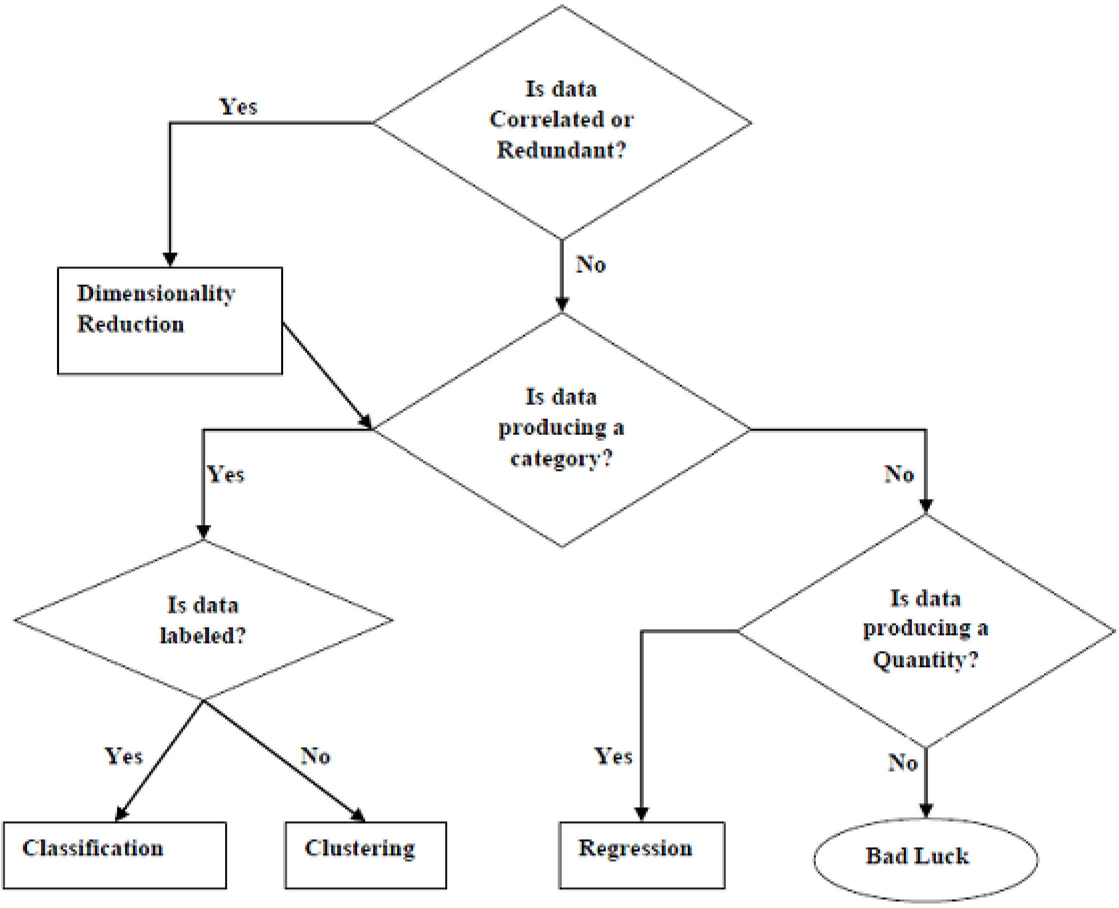
These methods are different from previously studied methods and very rarely used also. In this kind of learning algorithms, there would be an agent that we want to train over a period of time so that it can interact with a specific environment. The agent will follow a set of strategies for interacting with the environment and then after observing the environment it will take actions regards the current state of the environment. The following are the main steps of reinforcement learning methods −

- Step 1 − First, we need to prepare an agent with some initial set of strategies.
- Step 2 − Then observe the environment and its current state.
- Step 3 − Next, select the optimal policy regards the current state of the environment and perform important action.

- Step 4 − Now, the agent can get corresponding reward or penalty as per accordance with the action taken by it in previous step.
- Step 5 − Now, we can update the strategies if it is required so.
- Step 6 − At last, repeat steps 2-5 until the agent got to learn and adopt the optimal policies.

Tasks Suited for Machine Learning

The following diagram shows what type of task is appropriate for various ML problems −



Based on learning ability

In the learning process, the following are some methods that are based on learning ability −

Batch Learning

In many cases, we have end-to-end Machine Learning systems in which we need to train the model in one go by using whole available training data. Such kind of learning method or algorithm is called Batch or Offline learning. It is called Batch or Offline learning because it is a one-time procedure and the model will be trained with data in one single batch. The following are the main steps of Batch learning methods −

- Step 1 − First, we need to collect all the training data for start training the model.
- Step 2 − Now, start the training of model by providing whole training data in one go.
- Step 3 − Next, stop learning/training process once you got satisfactory results/performance.
- Step 4 − Finally, deploy this trained model into production. Here, it will predict the output for new data sample.

Online Learning

It is completely opposite to the batch or offline learning methods. In these learning methods, the training data is supplied in multiple incremental batches, called mini-batches, to the algorithm. Followings are the main steps of Online learning methods −

- Step 1 − First, we need to collect all the training data for starting training of the model.
- Step 2 − Now, start the training of model by providing a mini-batch of training data to the algorithm.
- Step 3 − Next, we need to provide the mini-batches of training data in multiple increments to the algorithm.
- Step 4 − As it will not stop like batch learning hence after providing whole training data in mini-batches, provide new data samples also to it.
- Step 5 − Finally, it will keep learning over a period of time based on the new data samples.

Based on Generalization Approach

In the learning process, followings are some methods that are based on generalization approaches −

Instance based Learning

Instance based learning method is one of the useful methods that build the ML models by doing generalization based on the input data. It is opposite to the previously studied learning methods in the way that this kind of learning involves ML systems as well as methods that uses the raw data points themselves to draw the outcomes for newer data samples without building an explicit model on training data.

In simple words, instance-based learning basically starts working by looking at the input data points and then using a similarity metric, it will generalize and predict the new data points.

Model based Learning

In Model based learning methods, an iterative process takes place on the ML models that are built based on various model parameters, called hyperparameters and in which input data is used to extract the features. In this learning, hyperparameters are optimized based on various model validation techniques. That is why we can say that Model based learning methods uses more traditional ML approach towards generalization.

Suppose if you want to start a ML project then what is the first and most important thing you would require? It is the data that we need to load for starting any of the ML project. With respect to data, the most common format of data for ML projects is CSV (comma-separated values).

Basically, CSV is a simple file format which is used to store tabular data (number and text) such as a spreadsheet in plain text. In Python, we can load CSV data into with different ways but before loading CSV data we must have to take care about some considerations.

Consideration While Loading CSV data

CSV data format is the most common format for ML data, but we need to take care about following major considerations while loading the same into our ML projects −

File Header

In CSV data files, the header contains the information for each field. We must use the same delimiter for the header file and for data file because it is the header file that specifies how should data fields be interpreted.

The following are the two cases related to CSV file header which must be considered −

- Case-I: When Data file is having a file header − It will automatically assign the names to each column of data if data file is having a file header.
- Case-II: When Data file is not having a file header − We need to assign the names to each column of data manually if data file is not having a file header.

In both the cases, we must need to specify explicitly weather our CSV file contains header or not.

Comments

Comments in any data file are having their significance. In CSV data file, comments are indicated by a hash (#) at the start of the line. We need to consider comments while loading CSV data into ML projects because if we are having comments in the file then we may need to indicate, depends upon the method we choose for loading, whether to expect those comments or not.

Delimiter

In CSV data files, comma (,) character is the standard delimiter. The role of delimiter is to separate the values in the fields. It is important to consider the role of delimiter while uploading the CSV file into ML projects because we can also use a different delimiter such as a tab or white space. But in the case of using a different delimiter than standard one, we must have to specify it explicitly.

Quotes

In CSV data files, double quotation (“ ”) mark is the default quote character. It is important to consider the role of quotes while uploading the CSV file into ML projects because we can also use other quote character than double quotation mark. But in case of using a different quote character than standard one, we must have to specify it explicitly.

Methods to Load CSV Data File

While working with ML projects, the most crucial task is to load the data properly into it. The most common data format for ML projects is CSV and it comes in various flavors and varying difficulties to parse. In this section, we are going to discuss about three common approaches in Python to load CSV data file −

Load CSV with Python Standard Library

The first and most used approach to load CSV data file is the use of Python standard library which provides us a variety of built-in modules namely csv module and the reader()function. The following is an example of loading CSV data file with the help of it −

Example

In this example, we are using the iris flower data set which can be downloaded into our local directory. After loading the data file, we can convert it into NumPy array and use it for ML projects. Following is the Python script for loading CSV data file −

First, we need to import the csv module provided by Python standard library as follows −

```
import csv
```

Next, we need to import Numpy module for converting the loaded data into NumPy array.

```
import numpy as np
```

Now, provide the full path of the file, stored on our local directory, having the CSV data file −

```
path = r"c:\iris.csv"
```

Next, use the csv.reader()function to read data from CSV file −

```
with open(path,'r') as f:
   reader = csv.reader(f,delimiter = ',')
   headers = next(reader)
   data = list(reader)
   data = np.array(data).astype(float)
```

We can print the names of the headers with the following line of script −

```
print(headers)
```

The following line of script will print the shape of the data i.e. number of rows & columns in the file −

```
print(data.shape)
```

Next script line will give the first three line of data file −

```
print(data[:3])
```

Output

```
['sepal_length', 'sepal_width', 'petal_length', 'petal_width']
(150, 4)
[  [5.1  3.5  1.4  0.2]
   [4.9  3.   1.4  0.2]
   [4.7  3.2  1.3  0.2]
]
```

Load CSV with NumPy

Another approach to load CSV data file is NumPy and numpy.loadtxt() function. The following is an example of loading CSV data file with the help of it −

Example

In this example, we are using the Pima Indians Dataset having the data of diabetic patients. This dataset is a numeric dataset with no header. It can also be downloaded into our local directory. After loading the data file, we can convert it into NumPy array and use it for ML projects. The following is the Python script for loading CSV data file −

```
from numpy import loadtxt
path = r"C:\pima-indians-diabetes.csv"
```

```
datapath= open(path, 'r')
data = loadtxt(datapath, delimiter=",")
print(data.shape)
print(data[:3])
```

Output

```
(768, 9)
[  [ 6.   148.   72.   35.   0.   33.6   0.627   50.  1.]
   [ 1.   85.    66.   29.   0.   26.6   0.351   31.  0.]
   [ 8.   183.   64.   0.    0.   23.3   0.672   32.  1.]
]
```

Load CSV with Pandas

Another approach to load CSV data file is by Pandas and pandas.read_csv()function. This is the very flexible function that returns a pandas.DataFrame which can be used immediately for plotting. The following is an example of loading CSV data file with the help of it −

Example

Here, we will be implementing two Python scripts, first is with Iris data set having headers and another is by using the Pima Indians Dataset which is a numeric dataset with no header. Both the datasets can be downloaded into local directory.

Script-1

The following is the Python script for loading CSV data file using Pandas on Iris Data set −

```
from pandas import read_csv
path = r"C:\iris.csv"
data = read_csv(path)
print(data.shape)
print(data[:3])
```

```
Output:

(150, 4)
    sepal_length    sepal_width   petal_length    petal_width
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
```

Script-2

The following is the Python script for loading CSV data file, along with providing the headers names too, using Pandas on Pima Indians Diabetes dataset −

```
from pandas import read_csv
path = r"C:\pima-indians-diabetes.csv"
headernames = ['preg', 'plas', 'pres', 'skin', 'test', 'mass',
'pedi', 'age', 'class']
data = read_csv(path, names=headernames)
print(data.shape)
print(data[:3])
```

Output

```
(768, 9)
   preg  plas  pres   skin  test   mass    pedi    age   class
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
```

The difference between above used three approaches for loading CSV data file can easily be understood with the help of given examples.

Introduction

While working with machine learning projects, usually we ignore two most important parts called *mathematics* and *data*. It is because, we know that ML is a data driven approach and our ML model will produce only as good or as bad results as the data we provided to it.

In the previous chapter, we discussed how we can upload CSV data into our ML project, but it would be good to understand the data before uploading it. We can understand the data by two ways, with statistics and with visualization.

In this chapter, with the help of following Python recipes, we are going to understand ML data with statistics.

Looking at Raw Data

The very first recipe is for looking at your raw data. It is important to look at raw data because the insight we will get after looking at raw data will boost our chances to better pre-processing as well as handling of data for ML projects.

Following is a Python script implemented by using head() function of Pandas DataFrame on Pima Indians diabetes dataset to look at the first 50 rows to get better understanding of it −

Example

```
from pandas import read_csv
path = r"C:\pima-indians-diabetes.csv"
headernames = ['preg', 'plas', 'pres', 'skin', 'test', 'mass',
'pedi', 'age', 'class']
data = read_csv(path, names=headernames)
print(data.head(50))
```

Output

```
preg   plas  pres    skin  test  mass   pedi    age      class
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
10     4      110     92      0   0     37.6    0.191    30    0
11    10      168     74      0   0     38.0    0.537    34    1
12    10      139     80      0   0     27.1    1.441    57    0
13     1      189     60     23  846    30.1    0.398    59    1
14     5      166     72     19  175    25.8    0.587    51    1
15     7      100      0      0   0     30.0    0.484    32    1
16     0      118     84     47  230    45.8    0.551    31    1
17     7      107     74      0   0     29.6    0.254    31    1
18     1      103     30     38  83     43.3    0.183    33    0
19     1      115     70     30  96     34.6    0.529    32    1
20     3      126     88     41  235    39.3    0.704    27    0
21     8       99     84      0   0     35.4    0.388    50    0
22     7      196     90      0   0     39.8    0.451    41    1
23     9      119     80     35   0     29.0    0.263    29    1
24    11      143     94     33  146    36.6    0.254    51    1
25    10      125     70     26  115    31.1    0.205    41    1
26     7      147     76      0   0     39.4    0.257    43    1
27     1       97     66     15  140    23.2    0.487    22    0
28    13      145     82     19  110    22.2    0.245    57    0
29     5      117     92      0   0     34.1    0.337    38    0
30     5      109     75     26   0     36.0    0.546    60    0
31     3      158     76     36  245    31.6    0.851    28    1
32     3       88     58     11   54    24.8    0.267    22    0
33     6       92     92      0   0     19.9    0.188    28    0
34    10      122     78     31   0     27.6    0.512    45    0
35     4      103     60     33  192    24.0    0.966    33    0
36    11      138     76      0   0     33.2    0.420    35    0
```

```
37     9    102    76    37    0    32.9    0.665    46    1
38     2     90    68    42    0    38.2    0.503    27    1
39     4    111    72    47  207    37.1    1.390    56    1
40     3    180    64    25   70    34.0    0.271    26    0
41     7    133    84     0    0    40.2    0.696    37    0
42     7    106    92    18    0    22.7    0.235    48    0
43     9    171   110    24  240    45.4    0.721    54    1
44     7    159    64     0    0    27.4    0.294    40    0
45     0    180    66    39    0    42.0    1.893    25    1
46     1    146    56     0    0    29.7    0.564    29    0
47     2     71    70    27    0    28.0    0.586    22    0
48     7    103    66    32    0    39.1    0.344    31    1
49     7    105     0     0    0    0.0     0.305    24    0
```

We can observe from the above output that first column gives the row number which can be very useful for referencing a specific observation.

Checking Dimensions of Data

It is always a good practice to know how much data, in terms of rows and columns, we are having for our ML project. The reasons behind are −

- Suppose if we have too many rows and columns then it would take long time to run the algorithm and train the model.
- Suppose if we have too less rows and columns then it we would not have enough data to well train the model.

Following is a Python script implemented by printing the shape property on Pandas Data Frame. We are going to implement it on iris data set for getting the total number of rows and columns in it.

Example

```
from pandas import read_csv
path = r"C:\iris.csv"
data = read_csv(path)
print(data.shape)
```

Output

```
(150, 4)
```

We can easily observe from the output that iris data set, we are going to use, is having 150 rows and 4 columns.

Getting Each Attribute’s Data Type

It is another good practice to know data type of each attribute. The reason behind is that, as per to the requirement, sometimes we may need to convert one data type to another. For example, we may need to convert string into floating point or int for representing categorial or ordinal values. We can have an idea about the attribute's data type by looking at the raw data, but another way is to use dtypes property of Pandas DataFrame. With the help of dtypes property we can categorize each attributes data type. It can be understood with the help of following Python script −

Example

```
from pandas import read_csv
path = r"C:\iris.csv"
data = read_csv(path)
print(data.dtypes)
```

Output

```
sepal_length  float64
sepal_width   float64
petal_length  float64
petal_width   float64
dtype: object
```

From the above output, we can easily get the datatypes of each attribute.

Statistical Summary of Data

We have discussed Python recipe to get the shape i.e. number of rows and columns, of data but many times we need to review the summaries out of that shape of data. It can be done with the help of describe() function of Pandas DataFrame that further provide the following 8 statistical properties of each & every data attribute −

- Count
- Mean
- Standard Deviation
- Minimum Value
- Maximum value
- 25%
- Median i.e. 50%
- 75%

Example

```
from pandas import read_csv
from pandas import set_option
path = r"C:\pima-indians-diabetes.csv"
```

```
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi',
'age', 'class']
data = read_csv(path, names=names)
set_option('display.width', 100)
set_option('precision', 2)
print(data.shape)
print(data.describe())
```

Output

```
(768, 9)
         preg      plas       pres      skin      test        mass
pedi      age      class
count 768.00      768.00    768.00     768.00    768.00     768.00
768.00    768.00    768.00
mean    3.85      120.89     69.11      20.54     79.80      31.99
0.47     33.24      0.35
std     3.37       31.97     19.36      15.95    115.24       7.88
0.33     11.76      0.48
min     0.00        0.00      0.00       0.00      0.00       0.00
0.08     21.00      0.00
25%     1.00       99.00     62.00       0.00      0.00      27.30
0.24     24.00      0.00
50%     3.00      117.00     72.00      23.00     30.50      32.00
0.37     29.00      0.00
75%     6.00      140.25     80.00      32.00    127.25      36.60
0.63     41.00      1.00
max    17.00      199.00    122.00      99.00    846.00      67.10
2.42     81.00      1.00
```

From the above output, we can observe the statistical summary of the data of Pima Indian Diabetes dataset along with shape of data.

Reviewing Class Distribution

Class distribution statistics is useful in classification problems where we need to know the balance of class values. It is important to know class value distribution because if we have highly imbalanced class distribution i.e. one class is having lots more observations than other class, then it may need special handling at data preparation stage of our ML project. We can easily get class distribution in Python with the help of Pandas DataFrame.

Example

```
from pandas import read_csv
path = r"C:\pima-indians-diabetes.csv"
```

```
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi',
'age', 'class']
data = read_csv(path, names=names)
count_class = data.groupby('class').size()
print(count_class)
```

Output

```
Class
0   500
1   268
dtype: int64
```

From the above output, it can be clearly seen that the number of observations with class 0 are almost double than number of observations with class 1.

Reviewing Correlation between Attributes

The relationship between two variables is called correlation. In statistics, the most common method for calculating correlation is Pearson's Correlation Coefficient. It can have three values as follows −

- Coefficient value = 1 − It represents full positive correlation between variables.
- Coefficient value = -1 − It represents full negative correlation between variables.
- Coefficient value = 0 − It represents no correlation at all between variables.

It is always good for us to review the pairwise correlations of the attributes in our dataset before using it into ML project because some machine learning algorithms such as linear regression and logistic regression will perform poorly if we have highly correlated attributes. In Python, we can easily calculate a correlation matrix of dataset attributes with the help of corr() function on Pandas DataFrame.

Example

```
from pandas import read_csv
from pandas import set_option
path = r"C:\pima-indians-diabetes.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi',
'age', 'class']
data = read_csv(path, names=names)
set_option('display.width', 100)
set_option('precision', 2)
correlations = data.corr(method='pearson')
print(correlations)
```

Output

```
preg      plas      pres      skin      test      mass      pedi
age       class
preg      1.00      0.13      0.14      -0.08     -0.07     0.02
-0.03        0.54    0.22
plas      0.13      1.00      0.15      0.06       0.33     0.22
0.14        0.26    0.47
pres      0.14      0.15      1.00      0.21       0.09     0.28
0.04        0.24    0.07
skin     -0.08      0.06      0.21      1.00       0.44     0.39
0.18       -0.11    0.07
test     -0.07      0.33      0.09      0.44       1.00     0.20
0.19       -0.04    0.13
mass      0.02      0.22      0.28      0.39       0.20     1.00
0.14        0.04    0.29
pedi     -0.03      0.14      0.04      0.18       0.19     0.14
1.00        0.03    0.17
age       0.54      0.26      0.24      -0.11     -0.04     0.04
0.03        1.00    0.24
class     0.22      0.47      0.07      0.07       0.13     0.29
0.17        0.24    1.00
```

The matrix in above output gives the correlation between all the pairs of the attribute in dataset.

Reviewing Skew of Attribute Distribution

Skewness may be defined as the distribution that is assumed to be Gaussian but appears distorted or shifted in one direction or another, or either to the left or right. Reviewing the skewness of attributes is one of the important tasks due to following reasons −

- Presence of skewness in data requires the correction at data preparation stage so that we can get more accuracy from our model.
- Most of the ML algorithms assumes that data has a Gaussian distribution i.e. either normal of bell curved data.

In Python, we can easily calculate the skew of each attribute by using skew() function on Pandas DataFrame.

Example

```
from pandas import read_csv
path = r"C:\pima-indians-diabetes.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi',
'age', 'class']
data = read_csv(path, names=names)
print(data.skew())
```

Output

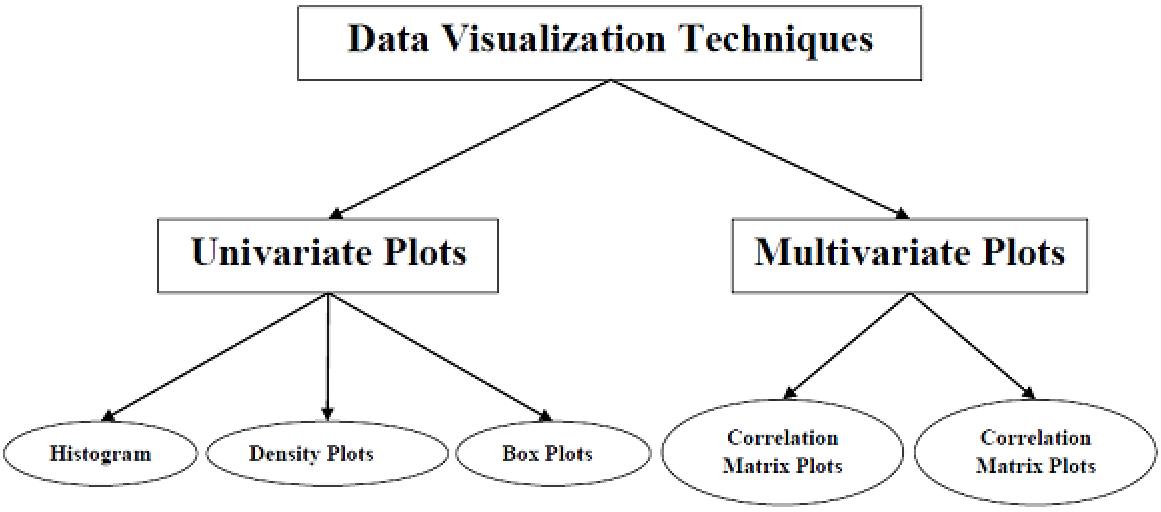
```
preg    0.90
plas    0.17
pres   -1.84
skin    0.11
test    2.27
mass   -0.43
pedi    1.92
age     1.13
class   0.64
dtype: float64
```

From the above output, positive or negative skew can be observed. If the value is closer to zero, then it shows less skew.

Introduction

In the previous chapter, we have discussed the importance of data for Machine Learning algorithms along with some Python recipes to understand the data with statistics. There is another way called Visualization, to understand the data.

With the help of data visualization, we can see how the data looks like and what kind of correlation is held by the attributes of data. It is the fastest way to see if the features correspond to the output. With the help of following Python recipes, we can understand ML data with statistics.



Univariate Plots: Understanding Attributes Independently

The simplest type of visualization is single-variable or “univariate” visualization. With the help of univariate visualization, we can understand each attribute of our dataset independently. The following are some techniques in Python to implement univariate visualization −

Histograms

Histograms group the data in bins and is the fastest way to get idea about the distribution of each attribute in dataset. The following are some of the characteristics of histograms −

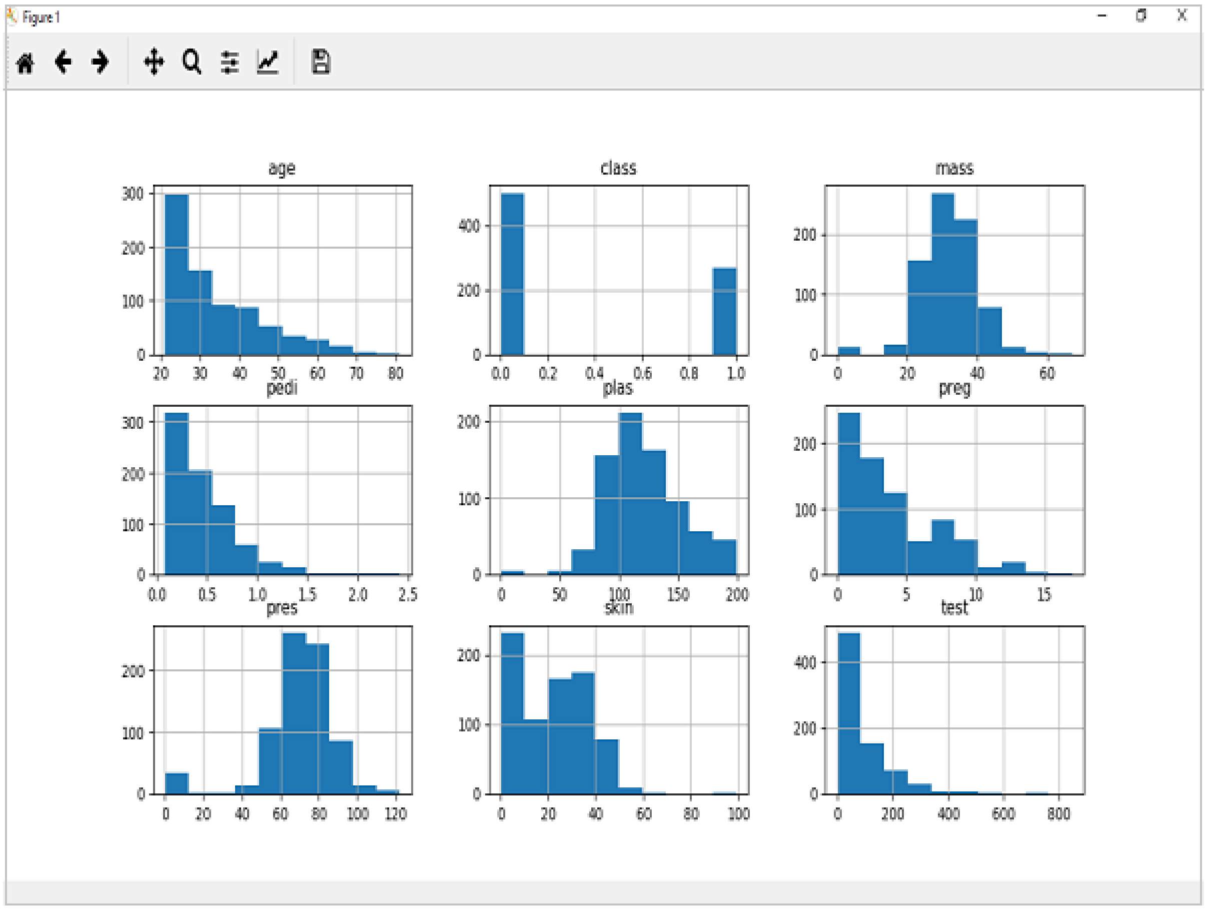
- It provides us a count of the number of observations in each bin created for visualization.
- From the shape of the bin, we can easily observe the distribution i.e. weather it is Gaussian, skewed or exponential.
- Histograms also help us to see possible outliers.

Example

The code shown below is an example of Python script creating the histogram of the attributes of Pima Indian Diabetes dataset. Here, we will be using hist() function on Pandas DataFrame to generate histograms and matplotlib for ploting them.

```
from matplotlib import pyplot
from pandas import read_csv
path = r"C:\pima-indians-diabetes.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi',
'age', 'class']
data = read_csv(path, names=names)
data.hist()
pyplot.show()
```

Output



The above output shows that it created the histogram for each attribute in the dataset. From this, we can observe that perhaps age, pedi and test attribute may have exponential distribution while mass and plas have Gaussian distribution.

Density Plots

Another quick and easy technique for getting each attributes distribution is Density plots. It is also like histogram but having a smooth curve drawn through the top of each bin. We can call them as abstracted histograms.

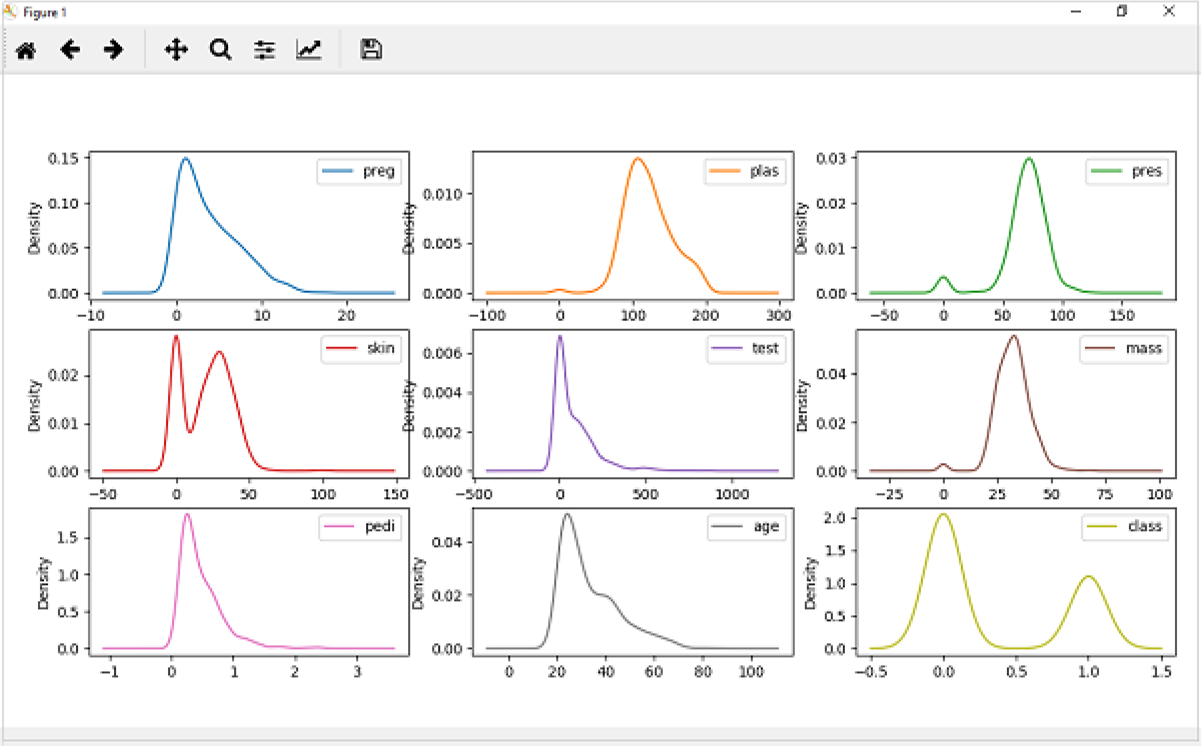
Example

In the following example, Python script will generate Density Plots for the distribution of attributes of Pima Indian Diabetes dataset.

```
from matplotlib import pyplot
from pandas import read_csv
path = r"C:\pima-indians-diabetes.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi',
'age', 'class']
data = read_csv(path, names=names)
data.plot(kind='density', subplots=True, layout=(3,3),
sharex=False)
```

```
pyplot.show()
```

Output



From the above output, the difference between Density plots and Histograms can be easily understood.

Box and Whisker Plots

Box and Whisker plots, also called boxplots in short, is another useful technique to review the distribution of each attribute's distribution. The following are the characteristics of this technique −

- It is univariate in nature and summarizes the distribution of each attribute.
- It draws a line for the middle value i.e. for median.
- It draws a box around the 25% and 75%.
- It also draws whiskers which will give us an idea about the spread of the data.
- The dots outside the whiskers signifies the outlier values. Outlier values would be 1.5 times greater than the size of the spread of the middle data.

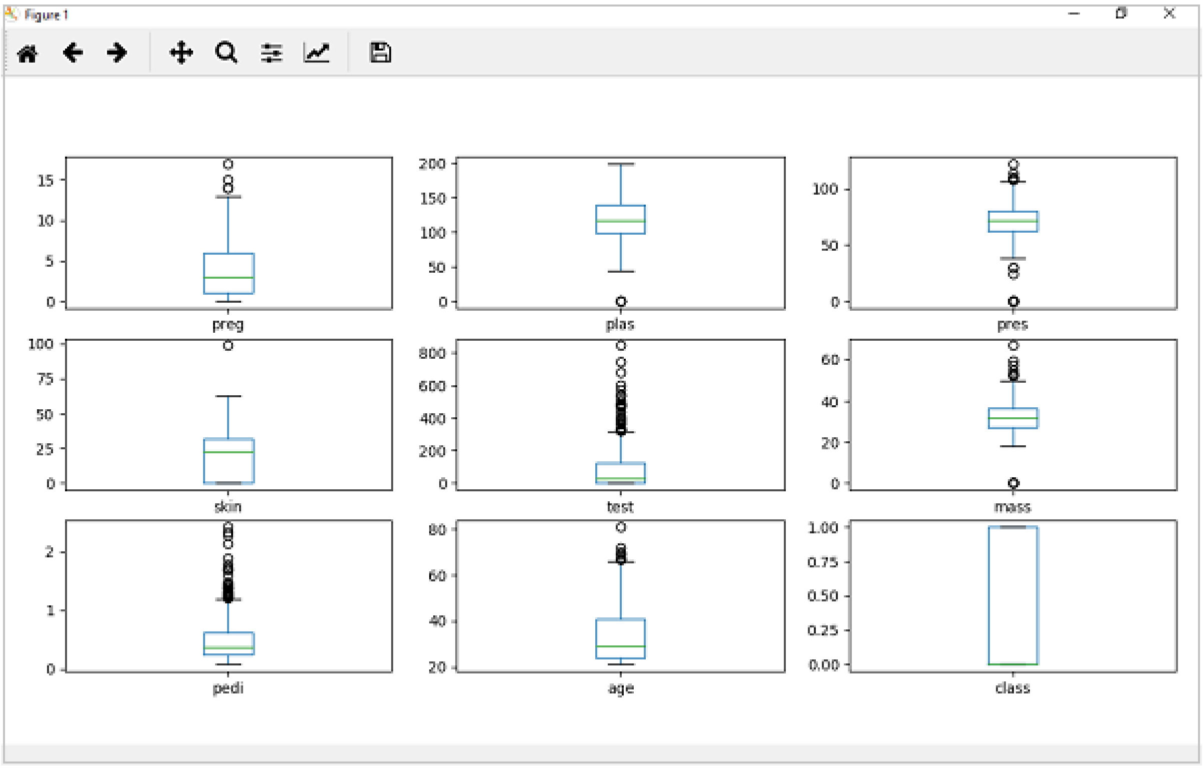
Example

In the following example, Python script will generate Density Plots for the distribution of attributes of Pima Indian Diabetes dataset.

```
from matplotlib import pyplot
from pandas import read_csv
path = r"C:\pima-indians-diabetes.csv"
```

```
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi',
'age', 'class']
data = read_csv(path, names=names)
data.plot(kind='box', subplots=True, layout=(3,3),
sharex=False,sharey=False)
pyplot.show()
```

Output



From the above plot of attribute’s distribution, it can be observed that age, test and skin appear skewed towards smaller values.

Multivariate Plots: Interaction Among Multiple Variables

Another type of visualization is multi-variable or “multivariate” visualization. With the help of multivariate visualization, we can understand interaction between multiple attributes of our dataset. The following are some techniques in Python to implement multivariate visualization −

Correlation Matrix Plot

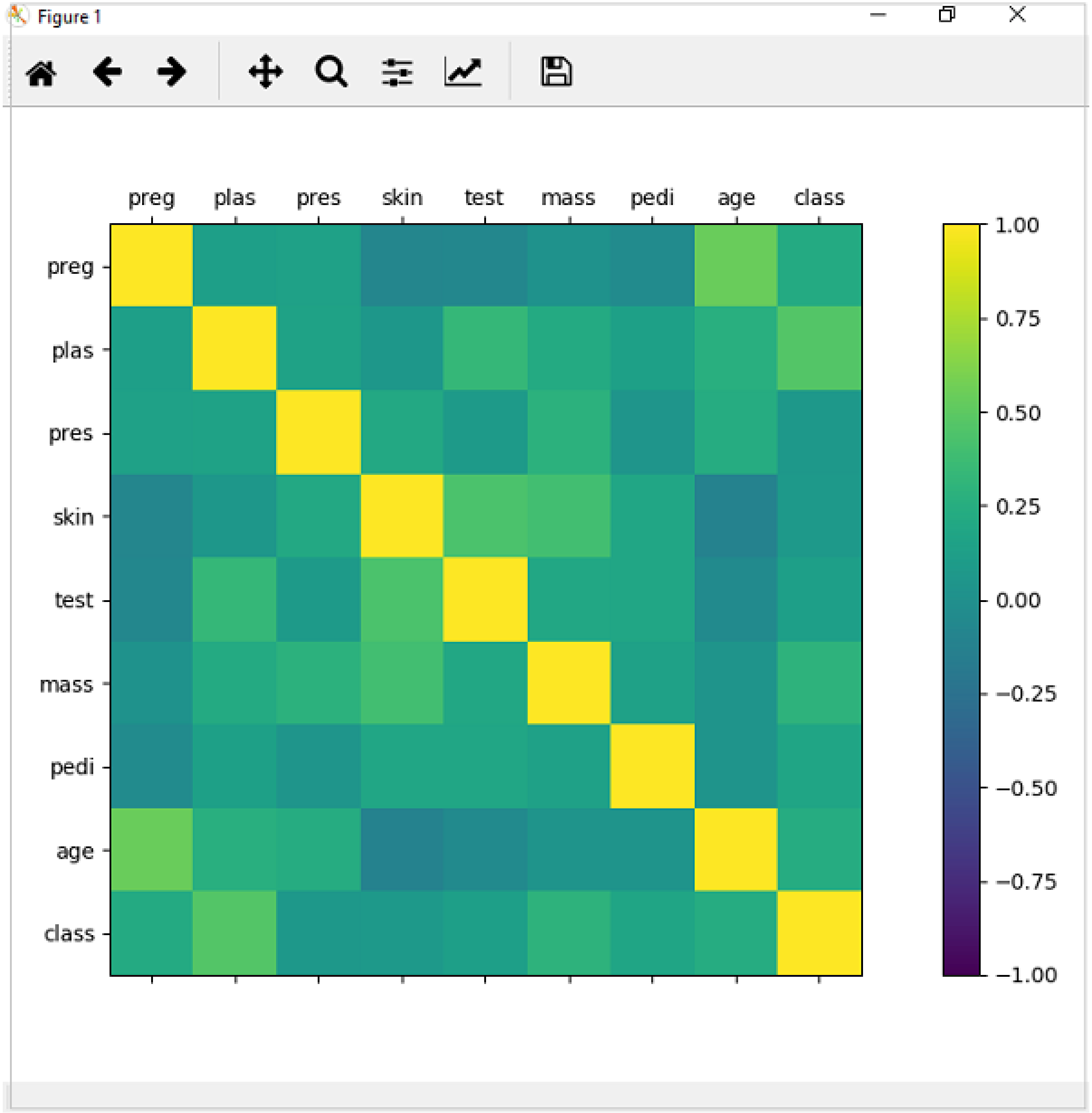
Correlation is an indication about the changes between two variables. In our previous chapters, we have discussed Pearson’s Correlation coefficients and the importance of Correlation too. We can plot correlation matrix to show which variable is having a high or low correlation in respect to another variable.

Example

In the following example, Python script will generate and plot correlation matrix for the Pima Indian Diabetes dataset. It can be generated with the help of corr() function on Pandas DataFrame and plotted with the help of pyplot.

```
from matplotlib import pyplot
from pandas import read_csv
import numpy
Path = r"C:\pima-indians-diabetes.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi',
'age', 'class']
data = read_csv(Path, names=names)
correlations = data.corr()
fig = pyplot.figure()
ax = fig.add_subplot(111)
cax = ax.matshow(correlations, vmin=-1, vmax=1)
fig.colorbar(cax)
ticks = numpy.arange(0,9,1)
ax.set_xticks(ticks)
ax.set_yticks(ticks)
ax.set_xticklabels(names)
ax.set_yticklabels(names)
pyplot.show()
```

Output



From the above output of correlation matrix, we can see that it is symmetrical i.e. the bottom left is same as the top right. It is also observed that each variable is positively correlated with each other.

Scatter Matrix Plot

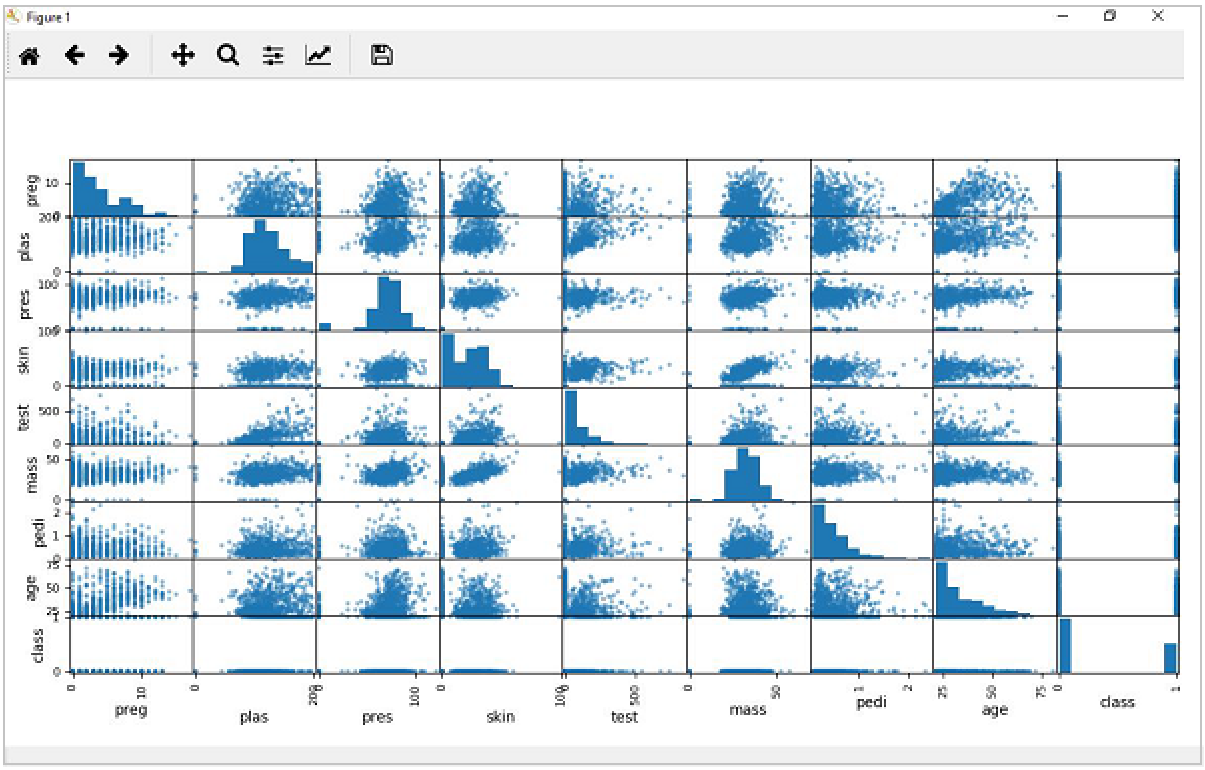
Scatter plots shows how much one variable is affected by another or the relationship between them with the help of dots in two dimensions. Scatter plots are very much like line graphs in the concept that they use horizontal and vertical axes to plot data points.

Example

In the following example, Python script will generate and plot Scatter matrix for the Pima Indian Diabetes dataset. It can be generated with the help of scatter_matrix() function on Pandas DataFrame and plotted with the help of pyplot.

```
from matplotlib import pyplot
from pandas import read_csv
from pandas.tools.plotting import scatter_matrix
path = r"C:\pima-indians-diabetes.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi',
'age', 'class']
data = read_csv(path, names=names)
scatter_matrix(data)
pyplot.show()
```

Output



Introduction

Machine Learning algorithms are completely dependent on data because it is the most crucial aspect that makes model training possible. On the other hand, if we won’t be able to make sense out of that data, before feeding it to ML algorithms, a machine will be useless. In simple words, we always need to feed right data i.e. the data in correct scale, format and containing meaningful features, for the problem we want machine to solve.

This makes data preparation the most important step in ML process. Data preparation may be defined as the procedure that makes our dataset more appropriate for ML process.

Why Data Pre-processing?

After selecting the raw data for ML training, the most important task is data pre-processing. In broad sense, data preprocessing will convert the selected data into a form we can work with or can feed to ML algorithms. We always need to preprocess our data so that it can be as per the expectation of machine learning algorithm.

Data Pre-processing Techniques

We have the following data preprocessing techniques that can be applied on data set to produce data for ML algorithms −

Scaling

Most probably our dataset comprises of the attributes with varying scale, but we cannot provide such data to ML algorithm hence it requires rescaling. Data rescaling makes sure that attributes are at same scale. Generally, attributes are rescaled into the range of 0 and 1. ML algorithms like gradient descent and k-Nearest Neighbors requires scaled data. We can rescale the data with the help of MinMaxScaler class of scikit-learn Python library.

Example

In this example we will rescale the data of Pima Indians Diabetes dataset which we used earlier. First, the CSV data will be loaded (as done in the previous chapters) and then with the help of MinMaxScaler class, it will be rescaled in the range of 0 and 1.

The first few lines of the following script are same as we have written in previous chapters while loading CSV data.

```
from pandas import read_csv
from numpy import set_printoptions
from sklearn import preprocessing
path = r'C:\pima-indians-diabetes.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi',
'age', 'class']
dataframe = read_csv(path, names=names)
array = dataframe.values
```

Now, we can use MinMaxScaler class to rescale the data in the range of 0 and 1.

```
data_scaler = preprocessing.MinMaxScaler(feature_range=(0,1))
```

```
data_rescaled = data_scaler.fit_transform(array)
```

We can also summarize the data for output as per our choice. Here, we are setting the precision to 1 and showing the first 10 rows in the output.

```
set_printoptions(precision=1)
print ("\nScaled data:\n", data_rescaled[0:10])
```

Output

```
Scaled data:
[
   [0.4 0.7 0.6 0.4 0.  0.5 0.2 0.5 1. ]
   [0.1 0.4 0.5 0.3 0.  0.4 0.1 0.2 0. ]
   [0.5 0.9 0.5 0.  0.  0.3 0.3 0.2 1. ]
   [0.1 0.4 0.5 0.2 0.1 0.4 0.  0.  0. ]
   [0.  0.7 0.3 0.4 0.2 0.6 0.9 0.2 1. ]
   [0.3 0.6 0.6 0.  0.  0.4 0.1 0.2 0. ]
   [0.2 0.4 0.4 0.3 0.1 0.5 0.1 0.1 1. ]
   [0.6 0.6 0.  0.  0.  0.5 0.  0.1 0. ]
   [0.1 1.  0.6 0.5 0.6 0.5 0.  0.5 1. ]
   [0.5 0.6 0.8 0.  0.  0.  0.1 0.6 1. ]
]
```

From the above output, all the data got rescaled into the range of 0 and 1.

Normalization

Another useful data preprocessing technique is Normalization. This is used to rescale each row of data to have a length of 1. It is mainly useful in Sparse dataset where we have lots of zeros. We can rescale the data with the help of Normalizer class of scikit-learn Python library.

Types of Normalization

In machine learning, there are two types of normalization preprocessing techniques as follows −

L1 Normalization

It may be defined as the normalization technique that modifies the dataset values in a way that in each row the sum of the absolute values will always be up to 1. It is also called Least Absolute Deviations.

Example

In this example, we use L1 Normalize technique to normalize the data of Pima Indians Diabetes dataset which we used earlier. First, the CSV data will be loaded and then with the help of Normalizer class it will be normalized.

The first few lines of following script are same as we have written in previous chapters while loading CSV data.

```
from pandas import read_csv
from numpy import set_printoptions
from sklearn.preprocessing import Normalizer
path = r'C:\pima-indians-diabetes.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi',
'age', 'class']
dataframe = read_csv (path, names=names)
array = dataframe.values
```

Now, we can use Normalizer class with L1 to normalize the data.

```
Data_normalizer = Normalizer(norm='l1').fit(array)
Data_normalized = Data_normalizer.transform(array)
```

We can also summarize the data for output as per our choice. Here, we are setting the precision to 2 and showing the first 3 rows in the output.

```
set_printoptions(precision=2)
print ("\nNormalized data:\n", Data_normalized [0:3])
```

Output

```
Normalized data:
[
    [0.02 0.43 0.21 0.1   0.  0.1   0.  0.14 0.  ]
    [0.   0.36 0.28 0.12 0.  0.11 0.  0.13 0.  ]
    [0.03 0.59 0.21 0.    0.  0.07 0.  0.1  0.  ]
]
```

L2 Normalization

It may be defined as the normalization technique that modifies the dataset values in a way that in each row the sum of the squares will always be up to 1. It is also called least squares.

Example

In this example, we use L2 Normalization technique to normalize the data of Pima Indians Diabetes dataset which we used earlier. First, the CSV data will be loaded (as done in previous chapters) and then with the help of Normalizer class it will be normalized.

The first few lines of following script are same as we have written in previous chapters while loading CSV data.

```
from pandas import read_csv
from numpy import set_printoptions
from sklearn.preprocessing import Normalizer
path = r'C:\pima-indians-diabetes.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi',
'age', 'class']
dataframe = read_csv (path, names=names)
array = dataframe.values
```

Now, we can use Normalizer class with L1 to normalize the data.

```
Data_normalizer = Normalizer(norm='l2').fit(array)
Data_normalized = Data_normalizer.transform(array)
```

We can also summarize the data for output as per our choice. Here, we are setting the precision to 2 and showing the first 3 rows in the output.

```
set_printoptions(precision=2)
print ("\nNormalized data:\n", Data_normalized [0:3])
```

Output

```
Normalized data:
[
    [0.03 0.83 0.4  0.2  0.  0.19 0.  0.28 0.01]
    [0.01 0.72 0.56 0.24 0.  0.22 0.  0.26 0.  ]
    [0.04 0.92 0.32 0.   0.  0.12 0.  0.16 0.01]
]
```

Binarization

As the name suggests, this is the technique with the help of which we can make our data binary. We can use a binary threshold for making our data binary. The values above that threshold value will be converted to 1 and below that threshold will be converted to 0. For example, if we choose threshold value = 0.5, then the dataset value above it will become 1 and below this will become 0. That is why we can call it binarizing the data or thresholding the data. This technique is useful when we have probabilities in our dataset and want to convert them into crisp values.

We can binarize the data with the help of Binarizer class of scikit-learn Python library.

Example

In this example, we will rescale the data of Pima Indians Diabetes dataset which we used earlier. First, the CSV data will be loaded and then with the help of Binarizer class it will be converted into binary values i.e. 0 and 1 depending upon the threshold value. We are taking 0.5 as threshold value.

The first few lines of following script are same as we have written in previous chapters while loading CSV data.

```
from pandas import read_csv
from sklearn.preprocessing import Binarizer
path = r'C:\pima-indians-diabetes.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi',
'age', 'class']
dataframe = read_csv(path, names=names)
array = dataframe.values
```

Now, we can use Binarize class to convert the data into binary values.

```
binarizer = Binarizer(threshold=0.5).fit(array)
Data_binarized = binarizer.transform(array)
```

Here, we are showing the first 5 rows in the output.

```
print ("\nBinary data:\n", Data_binarized [0:5])
```

Output

```
Binary data:
[
   [1. 1. 1. 1. 0. 1. 1. 1. 1.]
   [1. 1. 1. 1. 0. 1. 0. 1. 0.]
   [1. 1. 1. 0. 0. 1. 1. 1. 1.]
   [1. 1. 1. 1. 1. 1. 0. 1. 0.]
   [0. 1. 1. 1. 1. 1. 1. 1. 1.]
]
```

Standardization

Another useful data preprocessing technique which is basically used to transform the data attributes with a Gaussian distribution. It differs the mean and SD (Standard Deviation) to a standard Gaussian distribution with a mean of 0 and a SD of 1. This technique is useful in ML algorithms like linear regression, logistic regression that assumes a Gaussian distribution in input dataset and produce better results with rescaled data. We can standardize the data (mean = 0 and SD =1) with the help of StandardScaler class of scikit-learn Python library.

Example

In this example, we will rescale the data of Pima Indians Diabetes dataset which we used earlier. First, the CSV data will be loaded and then with the help of StandardScaler class it will be converted into Gaussian Distribution with mean = 0 and SD = 1.

The first few lines of following script are same as we have written in previous chapters while loading CSV data.

```
from sklearn.preprocessing import StandardScaler
from pandas import read_csv
from numpy import set_printoptions
path = r'C:\pima-indians-diabetes.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi',
'age', 'class']
dataframe = read_csv(path, names=names)
array = dataframe.values
```

Now, we can use StandardScaler class to rescale the data.

```
data_scaler = StandardScaler().fit(array)
data_rescaled = data_scaler.transform(array)
```

We can also summarize the data for output as per our choice. Here, we are setting the precision to 2 and showing the first 5 rows in the output.

```
set_printoptions(precision=2)
print ("\nRescaled data:\n", data_rescaled [0:5])
```

Output

```
Rescaled data:
[
   [ 0.64  0.85  0.15  0.91 -0.69  0.2   0.47  1.43  1.37]
   [-0.84 -1.12 -0.16  0.53 -0.69 -0.68 -0.37 -0.19 -0.73]
   [ 1.23  1.94 -0.26 -1.29 -0.69 -1.1   0.6  -0.11  1.37]
   [-0.84 -1.   -0.16  0.15  0.12 -0.49 -0.92 -1.04 -0.73]
   [-1.14  0.5  -1.5   0.91  0.77  1.41  5.48 -0.02  1.37]
]
```

Data Labeling

We discussed the importance of good fata for ML algorithms as well as some techniques to pre-process the data before sending it to ML algorithms. One more aspect in this regard is data labeling. It is also very important to send the data to ML algorithms having proper labeling. For example, in case of classification problems, lot of labels in the form of words, numbers etc. are there on the data.

What is Label Encoding?

Most of the sklearn functions expect that the data with number labels rather than word labels. Hence, we need to convert such labels into number labels. This process is called label encoding. We can perform label encoding of data with the help of LabelEncoder() function of scikit-learn Python library.

Example

In the following example, Python script will perform the label encoding.

First, import the required Python libraries as follows −

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

Now, we need to provide the input labels as follows −

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

The next line of code will create the label encoder and train it.

```
encoder = preprocessing.LabelEncoder()
encoder.fit(input_labels)
```

The next lines of script will check the performance by encoding the random ordered list −

```
test_labels = ['green','red','black']
encoded_values = encoder.transform(test_labels)
print("\nLabels =", test_labels)
print("Encoded values =", list(encoded_values))
encoded_values = [3,0,4,1]
decoded_list = encoder.inverse_transform(encoded_values)
```

We can get the list of encoded values with the help of following python script −

```
print("\nEncoded values =", encoded_values)
print("\nDecoded labels =", list(decoded_list))
```

Output

```
Labels = ['green', 'red', 'black']
Encoded values = [1, 2, 0]
Encoded values = [3, 0, 4, 1]
Decoded labels = ['white', 'black', 'yellow', 'green']
```

In the previous chapter, we have seen in detail how to preprocess and prepare data for machine learning. In this chapter, let us understand in detail data feature selection and various aspects involved in it.

Importance of Data Feature Selection

The performance of machine learning model is directly proportional to the data features used to train it. The performance of ML model will be affected negatively if the data features provided to it are irrelevant. On the other hand, use of relevant data features can increase the accuracy of your ML model especially linear and logistic regression.

Now the question arise that what is automatic feature selection? It may be defined as the process with the help of which we select those features in our data that are most relevant to the output or prediction variable in which we are interested. It is also called attribute selection.

The following are some of the benefits of automatic feature selection before modeling the data −

- Performing feature selection before data modeling will reduce the overfitting.
- Performing feature selection before data modeling will increases the accuracy of ML model.
- Performing feature selection before data modeling will reduce the training time

Feature Selection Techniques

The followings are automatic feature selection techniques that we can use to model ML data in Python −

Univariate Selection

This feature selection technique is very useful in selecting those features, with the help of statistical testing, having strongest relationship with the prediction variables. We can implement univariate feature selection technique with the help of SelectKBest0class of scikit-learn Python library.

Example

In this example, we will use Pima Indians Diabetes dataset to select 4 of the attributes having best features with the help of chi-square statistical test.

```
from pandas import read_csv
from numpy import set_printoptions
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
```

```
path = r'C:\pima-indians-diabetes.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi',
'age', 'class']
dataframe = read_csv(path, names=names)
array = dataframe.values
```

Next, we will separate array into input and output components −

```
X = array[:,0:8]
Y = array[:,8]
```

The following lines of code will select the best features from dataset −

```
test = SelectKBest(score_func=chi2, k=4)
fit = test.fit(X,Y)
```

We can also summarize the data for output as per our choice. Here, we are setting the precision to 2 and showing the 4 data attributes with best features along with best score of each attribute −

```
set_printoptions(precision=2)
print(fit.scores_)
featured_data = fit.transform(X)
print ("\nFeatured data:\n", featured_data[0:4])
```

Output

```
[ 111.52 1411.89 17.61 53.11 2175.57 127.67 5.39 181.3 ]
Featured data:
[
    [148. 0.   33.6 50. ]
    [ 85. 0.   26.6 31. ]
    [183. 0.   23.3 32. ]
    [ 89. 94.  28.1 21. ]
]
```

Recursive Feature Elimination

As the name suggests, RFE (Recursive feature elimination) feature selection technique removes the attributes recursively and builds the model with remaining attributes. We can implement RFE feature selection technique with the help of RFE class of scikit-learn Python library.

Example

In this example, we will use RFE with logistic regression algorithm to select the best 3 attributes having the best features from Pima Indians Diabetes dataset to.

```
from pandas import read_csv
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression
path = r'C:\pima-indians-diabetes.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 
'age', 'class']
dataframe = read_csv(path, names=names)
array = dataframe.values
```

Next, we will separate the array into its input and output components −

```
X = array[:,0:8]
Y = array[:,8]
```

The following lines of code will select the best features from a dataset −

```
model = LogisticRegression()
rfe = RFE(model, 3)
fit = rfe.fit(X, Y)
print("Number of Features: %d")
print("Selected Features: %s")
print("Feature Ranking: %s")
```

Output

```
Number of Features: 3
Selected Features: [ True False False False False True True False]
Feature Ranking: [1 2 3 5 6 1 1 4]
```

We can see in above output, RFE choose preg, mass and pedi as the first 3 best features. They are marked as 1 in the output.

Principal Component Analysis (PCA)

PCA, generally called data reduction technique, is very useful feature selection technique as it uses linear algebra to transform the dataset into a compressed form. We can implement PCA feature selection technique with the help of PCA class of scikit-learn Python library. We can select number of principal components in the output.

Example

In this example, we will use PCA to select best 3 Principal components from Pima Indians Diabetes dataset.

```
from pandas import read_csv
from sklearn.decomposition import PCA
```

```
path = r'C:\pima-indians-diabetes.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi',
'age', 'class']
dataframe = read_csv(path, names=names)
array = dataframe.values
```

Next, we will separate array into input and output components −

```
X = array[:,0:8]
Y = array[:,8]
```

The following lines of code will extract features from dataset −

```
pca = PCA(n_components=3)
fit = pca.fit(X)
print("Explained Variance: %s") % fit.explained_variance_ratio_
print(fit.components_)
```

Output

```
Explained Variance: [ 0.88854663 0.06159078 0.02579012]
[
   [
      -2.02176587e-03 9.78115765e-02 1.60930503e-02 6.07566861e-02
      9.93110844e-01 1.40108085e-02 5.37167919e-04 -3.56474430e-03
   ]
   [
      2.26488861e-02 9.72210040e-01 1.41909330e-01 -5.78614699e-02
      -9.46266913e-02 4.69729766e-02 8.16804621e-04 1.40168181e-01
   ]
   [
      -2.24649003e-02 1.43428710e-01 -9.22467192e-01
-3.07013055e-01
      2.09773019e-02 -1.32444542e-01 -6.39983017e-04
-1.25454310e-01
   ]
]
```

We can observe from the above output that 3 Principal Components bear little resemblance to the source data.

Feature Importance

As the name suggests, feature importance technique is used to choose the importance features. It basically uses a trained supervised classifier to select features. We can implement this feature selection technique with the help of ExtraTreeClassifier class of scikit-learn Python library.

Example

In this example, we will use ExtraTreeClassifier to select features from Pima Indians Diabetes dataset.

```
from pandas import read_csv
from sklearn.ensemble import ExtraTreesClassifier
path = r'C:\Desktop\pima-indians-diabetes.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi',
'age', 'class']
dataframe = read_csv(data, names=names)
array = dataframe.values
```

Next, we will separate array into input and output components −

```
X = array[:,0:8]
Y = array[:,8]
```

The following lines of code will extract features from dataset −

```
model = ExtraTreesClassifier()
model.fit(X, Y)
print(model.feature_importances_)
```

Output

```
[ 0.11070069 0.2213717 0.08824115 0.08068703 0.07281761 0.14548537
0.12654214 0.15415431]
```

From the output, we can observe that there are scores for each attribute. The higher the score, higher is the importance of that attribute.

....