ISSN (e): 2250-3021, ISSN (p): 2278-8719
PP 06-12

Machine Learning In Python

[1]Ms. Prajakta Ambalkar, [2]Mr. Dhiraj Rane
[1]Department of Computer Science, MCA-sem5, RTMNU, Nagpur, Maharashtra.
[2]Department of Computer Science, MCA, RTMNU, Nagpur, Maharashtra.

Abstract: *Scikit-learn is an increasingly popular machine learning library. Written in Python, it is designed to be simple and efficient, accessible to non-experts, and reusable in various contexts. In this paper, we present and discuss the libraries that arenumpy, pandas,scipy also discuss the design choices for the application programming interface (API) of the project. In particular, we describe the simple and elegant interface shared by all learning and processing units in the library and then discuss its advantages in terms of composition and reusability. Scikit-learn is the package focuses on bringing machine learning to non-specialists using a general-purpose high-level language. Emphasis is put on ease of use, performance, documentation, and API consistency. It has minimal dependencies and is distributed under the simplified BSD license, encouraging its use in both academic and commercial settings.*

Keywords:–*API, NumPy, Pandas,SciPy, Algorithms, Models, and Modules.*

I. Introduction

Machine learning is a type of artificial intelligence (AI) that provides computers with the ability to learn without being explicitly programmed. Machine learning focuses on the development of Computer Programs that can change when exposed to new data. In this article, we'll see basics of Machine Learning, and implementation of a simple machine learning algorithm using python. Python community has developed many modules to help programmers implement machine learning. The Python programming language is establishing itself as one of the most popular languages for scientific computing. Thanks to its high-level interactive nature and its maturing ecosystem of scientific libraries, it is an appealing choice for algorithmic development and exploratory data analysis (Dubois, 2007; Milmann andAvaizis, 2011).

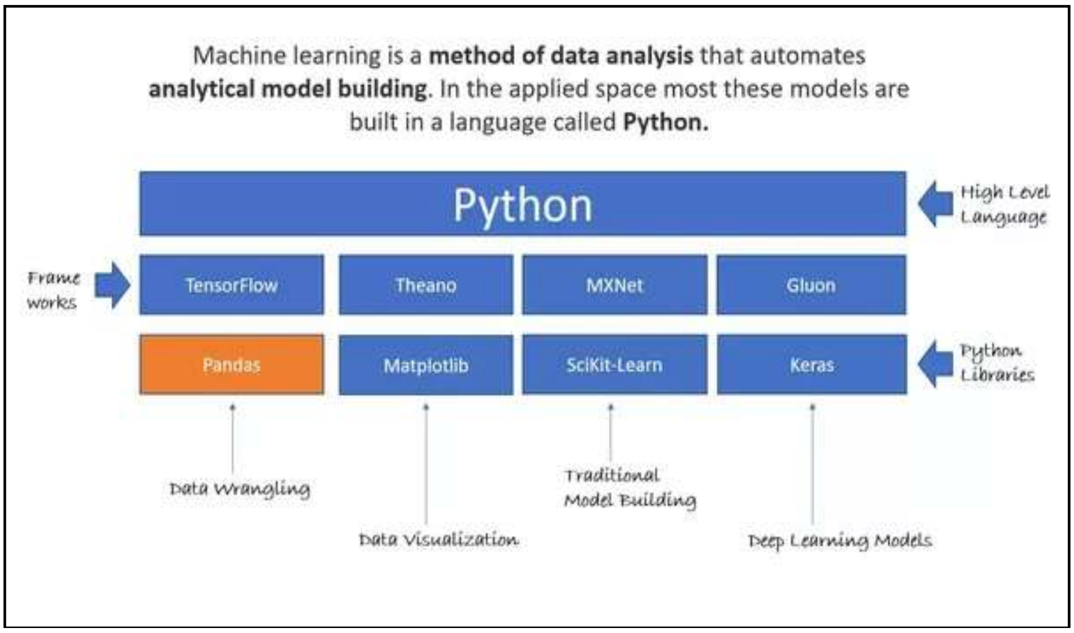
Machine learning is a **method of data analysis** that automates **analytical model building**. In the applied space most these models are built in a language called **Python.**
Python
High Level Language
Frame works
TensorFlow
Theano
MXNet
Gluon
Pandas
Matplotlib
SciKit-Learn
Keras
Python Libraries
Data Wrangling
Data Visualization
Traditional Model Building
Deep Learning Models

Fig: 1.1 Models in Python

Yet, as a general-purpose language, it is increasingly used not only in academic settings but also in industry. Scikit-learn harnesses this rich environment to provide state-of-the-art implementations of many well-known machine learning algorithms, while maintaining an easy-to-use interface tightly integrated with the Python language. This answers the growing need for statistical data analysis by non-specialists in the software and web industries, as well as in fields outside of computer-science, such as biology or physics. *Scikit-*

*learn*differs from other machine learning toolboxes in Python for various reasons: *i)* it is distributed under the BSD license *ii)* it incorporates compiled code for efficiency, unlike MDP (Zito et al., 2008) and pybrain (Schaul et al., 2010), *iii)* it depends only on numpy and scipy to facilitate easy distribution, unlike pymvpa (Hanke et al., 2009) that has optional dependencies such as R and shogun, and *iv)* it focuses on imperative programming, unlike pybrain which uses a data-flow framework. While the package is mostly written in Python, it incorporates the C++ libraries LibSVM (Chang and Lin, 2001) and LibLinear (Fan et al., 2008) that provide referenceimplementations of SVMs and generalized linear models with compatible licenses. Binary packages are available on a rich set of platforms including Windows and any POSIX platforms.Furthermore, thanks to its liberal license, it has been widely distributed as part of major free software distributions such as Ubuntu, Debian, Mandriva, NetBSD and Macports and in commercial distributions such as the "Enthought Python Distribution".

II. Project Vision

Code quality. Rather than providing as many features as possible, the project's goal has been toprovide solid implementations. Code quality is ensured with unit tests—as of release 0.8, testcoverage is 81%—and the use of static analysis tools such as pyflakes and pep8. Finally, westrive to use consistent naming for the functions and parameters used throughout a strict adherenceto the Python coding guidelines and numpy style documentation. *SD licensing.* Most of the Python ecosystem is licensed with non-copy left licenses. While suchpolicy is beneficial for adoption of these tools by commercial projects, it does impose some restrictions: we are unable to use some existing scientific code, such as the GSL.*Bare-bone design and API.* To lower the barrier of entry, we avoid framework code and keep thenumber of different objects to a minimum, relying on numpy arrays for data containers. *Community-driven development.* We base our development on collaborative tools such as git, githuband public mailing lists. External contributions are welcome and encouraged.*Documentation. Scikit-learn* provides a _300 page user guide including narrative documentation,class references, a tutorial, installation instructions, as well as more than 60 examples, some featuringreal-world applications. We try to minimize the use of machine-learning jargon, while maintainingprecision with regards to the algorithms employed.

III. Underlying Technologies

Numpy:the base data structure used for data and model parameters. Input data is presented asnumpy arrays, thus integrating seamlessly with other scientific Python libraries. Numpy's view basedmemory model limits copies, even when binding with compiled code (Van der Walt et al.,2011). It also provides basic arithmetic operations.

Scipy:efficient algorithms for linear algebra, sparse matrix representation, special functions andbasic statistical functions. Scipy has bindings for many Fortran-based standard numerical packages,such as LAPACK. This is important for ease of installation and portability, as providing librariesaround FORTRAN code can prove challenging on various platforms.

Cython:a language for combining C in Python. Cython makes it easy to reach the performanceof compiled languages with Python-like syntax and high-level operations. It is also used to bindcompiled libraries, eliminating the boilerplate code of Python/C extensions.

A. NumPy :

NumPy derives from an old library called Numeric, which was the first array object built for Python. It was quite successful and was used in a variety of applications before being phased out. NumPy also incorporates features introduced by a library called Numarray, which was written after Numeric but before NumPy. When NumPy was first written, it wasn't actually called "NumPy". For about 6 months at the end of 2005, NumPy was called SciPy Core (not to be confused with the full SciPy package which remains a separate package). However, it was decided in January 2006 to go with the historical name of NumPy for the new package.

Inclusion of a Numpy in Python's standard library: In the opinion of many involved in the Numpy development, anN-dimensional array interface should be part of the Python standard libraries. Hence, a PEP was started to describe what exactly is meant by an array interface, and a webpage was set up with useful information. At the SciPy conference in 2006, Guido and Travis discussed which parts of NumPy should go into Python. They decided that the best course to pursue is to write a series of PEPs to get

1. the data-type object into Python
2. Extend the buffer interface with the array interface.

B.SciPy :

In the 1990s, Python was extended to include an array type for numerical computing called Numeric (This package was eventually replaced by Travis Oliphant who wrote NumPy in 2006 as a blending of Numeric and Numarray which had been started in 2001). As of 2000, there was a growing number of extension modules and increasing interest in creating a complete environment for scientific and technical computing. In 2001, Travis Oliphant, Eric Jones, and Pearu Peterson merged code they had written and called the resulting package SciPy. The newly created package provided a standard collection of common numerical operations on top of the Numeric array data structure. Shortly thereafter, Fernando Pérez released IPython, an enhanced interactive shell widely used in the technical computing community, and John Hunter released the first version of Matplotlib, the 2D plotting library for technical computing. Since then the SciPy environment has continued to grow with more packages and tools for technical computing. Several people used *Numeric* as a base for their scientific code and developed their own modules. Around 2001, Travis Oliphant, Eric Jones and Pearu Peterson merged their modules in one scientific super package: **SciPy** was born.

Data structures:The basic data structure used by SciPy is a multidimensional array provided by the NumPy module. NumPy provides some functions for linear algebra, Fourier transforms, and random number generation, but not with the generality of the equivalent functions in SciPy. NumPy can also be used as an efficient multidimensional container of data with arbitrary datatypes. This allows NumPy to seamlessly and speedily integrate with a wide variety of databases. Older versions of SciPy used Numeric as an array type,which is now deprecated in favour of the newer NumPy array code.

C. Pandas :

Pandas is hands down one of the best libraries of python. It supports reading and writing excel spreadsheets, CVS's and a whole lot of manipulation. It is more like a mandatory library you need to know if you're dealing with datasets from excel files and CSV files. I.e. for Machine learning and data science.Machine Learning with Python. Machine learning is a branch in computer science that studies the design of algorithms that can learn. Typical tasks are concept learning, function learning or "predictive modelling", clustering and finding predictive patterns.

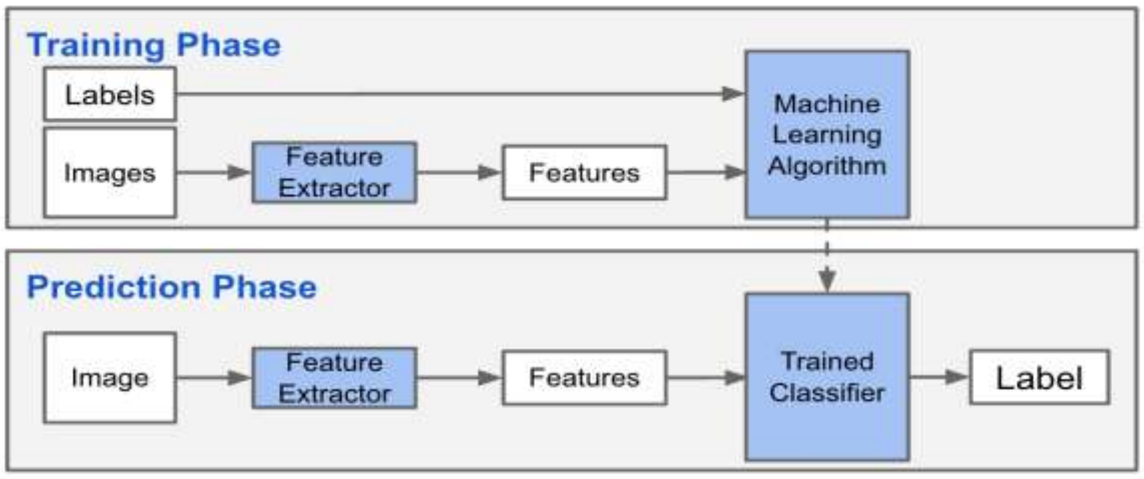

Figure 2:*Phases of Machine Learning*

IV. Core API

"In simple words, an API is a (hypothetical) contract between 2 Softwares saying if the user software provides input in a pre-defined format, the later with extend its functionality and provide the outcome to the user software."All objects within scikit-learn share a uniform common basic API consisting ofthree complementary interfaces: an estimator interface for building and fittingmodels, a predictor interface for making predictions and a transformer interfacefor converting data. In this section, we describe these three interfaces, afterreviewing our general principles and data representation choices.

A. General principles

As much as possible, our design choices have been guided so as to avoid theproliferation of framework code. Wetry to adopt simple conventions and tolimit to a minimum the number of methods an object must implement. **The APIis designed to adhere to the following broad principles:**

Consistency. All objects (basic or composite) share a consistent interface composedof a limited set of methods. This interface is documented in a consistentmanner for all objects. Inspection. Constructor parameters and parameter values determined by learningalgorithms are stored and exposed as public attributes.Non-proliferation of classes. Learning algorithms are the only objects to berepresented using custom classes. Datasets are represented as NumPy arraysor SciPy sparse matrices. Hyper-parameter names and values are representedas standard Python strings or numbers whenever possible. This keeps scikitlearneasy to use and easy to combine with other libraries. Composition. Many machine learning tasks are expressible as sequences orcombinations of transformations to data. Some learning algorithms are alsonaturally viewed asmeta-algorithms parametrized on other algorithms. Wheneverfeasible, such algorithms are implemented and composed from existing building blocks. Sensible defaults. Whenever an operation requires a user-defined parameter,an appropriate default value is defined by the library. The default valueshould cause the operation to be performed in a sensible way (giving a baselinesolution for the task at hand).

B. Data representation:

In most machine learning tasks, data is modelled as a set of variables. For example,in a supervised learning task, the goal is to find a mapping from inputvariables X1, . . .Xp, called features, to some output variables Y . A sample isthen defined as a pair of values ([x1, . . . , xp]T, y) of these variables. A widelyused representation of a dataset, a collection of such samples, is a pair of matriceswith numerical values: one for the input values and one for the output values. Each row of these matrices corresponds to one sample of the dataset andeach column to one variable of the problem. In scikit-learn, we chose a representation of data that is as close as possibleto the matrix representation: datasets are encoded as NumPy multidimensionalarrays for dense data and as SciPy sparse matrices for sparse data. While thesemay seem rather unsophisticated data representations when compared to moreobject-oriented constructs, such as the ones used by Weka (Hall et al., 2009),they bring the prime advantage of allowing us to rely on efficient NumPy andSciPy vectorised operations while keeping the code short and readable. Thisdesign choice has also been motivated by the fact that, given their pervasiveness

In many other scientific Python packages, many scientific users of Python arealready familiar with NumPy dense arrays and SciPy sparse matrices. From apractical point of view, these formats also provide a collection of data loading andconversion tools which make them very easy to use in many contexts. Moreover,for tasks where the inputs are text files or semi-structured objects, we providevectorizer objects that efficiently convert such data to the NumPy or SciPyformats. For efficiency reasons, the public interface is oriented towards processingbatches of samples rather than single samples per API call. While classificationand regression algorithms can indeed make predictions for single samples,scikit-learn objects are not optimized for this use case. (The few online learningalgorithms implemented are intended to take mini-batches.) Batch processingmakes optimal use of NumPy and SciPy by preventing the overhead inherent to Python function calls or due to per-element dynamic type checking. Althoughthis might seem to be an artefact of the Python language, and therefore an implementationdetail that leaks into the API, we argue that APIs should be designedso as not to tie a library to a suboptimal implementation strategy. As such, batchprocessing enables fast implementations in lower-level languages (where memoryhierarchy effects and the possibility of internal parallelization come into play).

```
from s k l e a r n . l ine a r mode l import Lo g i s t i cRe g r e s s i o n
c l f = Lo g i s t i cRe g r e s s i o n ( pena l ty=” l 1 ” )
c l f . f i t ( X t rain , y t r a i n )
```

In this snippet, a LogisticRegression estimator is first initialized by setting thepenalty hyper-parameter to "l1" for ℓ1 regularization. Other hyper-parameters(such as C, the strength of the regularization) are not explicitly given and thusset to the default values. Upon calling fit, a model is learned from the trainingarrays X train and y train, and stored within the object for later use. Sinceall estimators share the same interface, using a different learning algorithm is assimple as replacing the constructor (the class name); to build a random forest onthe same data, one would simply replace Logistic Regression (penalty="l1") in the snippet above by RandomForestClassifier ().In scikit-learn, classical learning algorithms are not the only objects to beimplemented as estimators. For example, pre-processing routines (e.g., scaling offeatures) or feature extraction techniques (e.g., vectorization of text documents)also implement the estimator interface. Even stateless processing steps, that donot require the fit method to perform useful work, implement the estimatorinterface. As we will illustrate in the next sections, this design pattern is indeedof prime importance for consistency, composition and model selection reasons.

V. Predictors

The predictor interface extends the notion of an estimator by adding a predictmethod that takes an array X test and produces predictions for X test, based onthe learned parameters of the estimator (we call the input to predict “X test”in order to emphasize that predict generalizes to new data). In the case ofsupervised learning estimators, this method typically returns the predicted labelsor values computed by the model. Continuing with the previous example,predicted labels for X test can be obtained using the following snippet:

```
y pr ed = c l f . p r e d i c t ( X t e s t )
```

VI. Estimators

The estimator interface is at the core of the library. It defines instantiationmechanisms of objects and exposes a fit method for learning a model fromtraining data. All supervised and unsupervised learning algorithms (e.g., for

Classification, regression or clustering) are offered as objects implementing thisinterface. Machine learning tasks like feature extraction, feature selection ordimensionality reduction are also provided as estimators. Estimator initialization and actual learning are strictly separated, in a waythat is similar to partial function application: an estimator is initialized from aset of named constant hyper-parameter values (e.g., the C constant in SVMs) and can be considered as a function that maps these values to actual learningalgorithms. The constructor of an estimator does not see any actual data, nordoes it perform any actual learning. All it does is attach the given parametersto the object. For the sake of convenient model inspection, hyper-parametersare set as public attributes, which is especially important in model selection settings. For ease of use, default hyper-parameter values are also provided for allbuilt-in estimators. These default values are set to be relevant in many commonsituations in order to make estimators as effective as possible out-of-box fornon-experts. Actual learning is performed by the fit method. This method is called withtraining data (e.g., supplied as two arrays X train and y train in supervisedlearning estimators). Its task is to run a learning algorithm and to determinemodel-specific parameters from the training data and set these as attributes onthe estimator object. As a convention, the parameters learned by an estimatorare exposed as public attributes with names suffixed with a trailing underscore (e.g., coef for the learned coefficients of a linear model), again to facilitatemodel inspection. In the partial application view, fit is a function from datato a model of that data. It always returns the estimator object it was called on,which now serves as a model of its input and can be used to perform predictions or transformations of input data.From the start, the choice to let a single object serve dual purpose as estimatorand model has mostly been driven by usability and technical considerations.From the user point of view, having two coupled instances (i.e., an estimatorobject, used as a factory, and a model object, produced by the estimator) indeeddecreases the ease of use and is also more likely to unnecessarily confusenewcomers. From the developer point of view, decoupling estimators from modelsalso creates parallel class hierarchies and increases the overall maintenancecomplexity of the project. For these practical reasons, we believe that decouplingestimators from models is not worth the effort. A good reason for decouplinghowever, would be that it makes it possible to ship a model in a new environmentwithout having to deal with potentially complex software dependencies. Such afeature could however still be implemented in scikit-learn by making estimatorsAble to export a fitted model, using the information from its public attributes,to an agnostic model description such as PMML (Guazzelli et al., 2009).To illustrate the initialize-fit sequence, let us consider a supervised learningtask using logistic regression. Given the API defined above, solving this problem is as simple as the following example.

VII. Model Selection

As introduced in Section 2, hyper-parameters set in the constructor of anestimator determine the behaviour of the learning algorithm and hence the performanceof the resulting model on unseen data. The problem of model selectionis therefore to find, within some hyper-parameter space, the best combination of hyper-parameters,with respect to some user-specified criterion. For example, adecision tree with too small a value for the maximal tree depth parameter willtend to under fit, while too large a value will make it overfIt.In scikit-learn, model selection is supported in two distinct meta-estimators,GridSearchCV and RandomizedSearchCV.They take as input an estimator (basicor composite), whose hyper-parameters must be optimized, and a set ofhyperparametersettings to search through. This set is represented as a mapping ofparameter names to a set of discrete choices in the case of grid search, whichexhaustively enumerates the “grid” (Cartesian product) of complete parametercombinations. Randomized search is a smarter algorithm that avoids the combinatorialexplosion in grid search by sampling a fixed number of times from itsparameter distributions (see Bergstra and Bengio, 2012).

VIII. Implementation

Our implementation guidelines emphasize writing efficient but readable code. Inparticular, we focus on making the codebase easily maintainable and understandablein order to favour external contributions. Whenever practicable, algorithmsimplemented in scikit-learn are written in Python, using NumPy vector operations for numerical work. This allows for the code to remain concise, readable andefficient. For critical algorithms that cannot be easily and efficiently expressed asNumPy operations, we rely on Cython (Behnel et al., 2011) to achieve competitiveperformance and scalability. Cython is a compiled programming languagethat extends Python with static typing. It produces efficient C extension modulesthat are directly importable from the Python run-time system. Examplesof algorithms written in Cython include stochastic gradient descent for linearmodels, some graph-based clustering algorithms and decision trees. To facilitate the installation and thus adoption of scikit-learn, the set ofexternal dependencies is kept to a bare minimum: only Python, NumPy andSciPy are required for a functioning installation. Binary distributions of these areavailable for the major platforms. Visualization functionality depends on Matplotlib(Hunter, 2007) and/or Graphviz (Gansner and North, 2000), but neitheris required to perform machine learning or prediction. When feasible, externallibraries are integrated into the codebase. In particular, scikit-learn includesmodified versions of LIBSVM and LIBLINEAR (Chang and Lin, 2011; Fan et al.,2008), both written in C++ and wrapped using Cython modules.

IX. Conclusion

The foremost target of ML researchers is to design more efficient (in terms of both time and space) and practicalgeneral purpose learning methods that can perform better over a widespread domain. In the context of ML, the

Efficiency with which a method utilises data resources that is also an important performance paradigm along with time and space complexity. Higher accuracy of prediction and humanly interpretable prediction rules are also of high importance. Being completely data-driven and having the ability to examine a large amount of data in smaller intervals of time,ML algorithms has an edge over manual or direct programming. Also they are often more accurate and not prone to human bias.We have discussed the scikit-learn API and the way it maps machine learningconcepts and tasks onto objects and operations in the Python programminglanguage. We have shown how a consistent API across the package makes scikitlearnvery usable in practice: experimenting with different learning algorithm is as simple as substituting a new class definition. Through composition interfacessuch as Pipelines, Feature Unions, and Metaestimators, these simple buildingblocks lead to an API which is powerful, and can accomplish a wide variety oflearning tasks within a small amount of easy-to-read code. Through duck-typing, the consistent API leads to a library that is easily extensible, and allows user definedestimators to be incorporated into the scikit-learn workflow without anyexplicit object inheritance.While part of the scikit-learn API is necessarily Python-specific, core conceptsmay be applicable to machine learning applications and toolkits writtenin other (dynamic) programming languages. The power, and extensibility of thescikit-learn API is evidenced by the large and growing user-base, its use tosolve real problems across a wide array of fields, as well as the appearance ofthird-party packages that follow the scikit-learn conventions. ML provide software the flexibility and adaptability when necessary. In spite of some application (e.g., to write matrix multiplication programs) where ML may fail to be beneficial, with increase of data resources and increasing demand in personalised customisable software, ML will thrive in near future. Besides software development, MLwill probably but help reformthe generaloutlook of Computer Science. By changing the defining question from “how to program a computer” to “how to empowered to program itself,” ML priories thedevelopment of devicesthat are self- monitoring, self-diagnosing and self-repairing, and the utilises of the data flow available within the program rather than just processing it. Likewise, it will help reform Statistical rules, byprovidingmore computational stance. Obviously, both Statistics and Computer Science will also embellish ML as they develop and contributemore advancedtheories to modify the way of learning.

Reference

[1]. Springer article_available:https://link.springer.com/article/10.1007%2FBF00116251

[2]. Total LLC [US] (2010 - 2019) _available_https://www.toptal.com/python/python-machine-learning-flask-example

[3]. Intro to Pandas: -1: An absolute beginners guide to Machine Learning and Data science._”HACKER NOON”_ available:https://hackernoon.com/intro-to-pandas-1-an-absolute-beginners-guide-to-machine-learning-and-data-science-a1fed3a6f0f3

[4]. This is an archival dump of old wiki content --- see scipy.org for current material_ available:https://scipy.github.io/old-wiki/pages/History_of_SciPy

[5]. SPRINGER NATURE_ Kluwer Academic Publishers 1986 (2018)Introduction of Decision Tree_ available: https://scipy.github.io/old-wiki/pages/History_of_SciPy

[6]. A Gentle Introduction to Scikit-Learn: A Python Machine Learning Library (2019) Machine Learning Mastery_available:https://machinelearningmastery.com/a-gentle-introduction-to-scikit-learn-a-python-machine-learning-library/

[7]. Coursera (2019) Videos: _ available: https://www.coursera.org/learn/machine-learning-with-python

[8]. Book: Hands-On Machine Learning with Scikit-Learn and Tensor Flow[Concepts, Tools, and Techniques to Build Intelligent Systems] (2017) _by AurelianGerona. Published by_O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.