**Optimization and Parallelization of Machine Learning Algorithms for DNA Classification**

***This project report is submitted to***

### Yeshwantrao Chavan College of Engineering

***(An Autonomous Institution Affiliated to Rashtrasant Tukdoji Maharaj Nagpur University)***

#### In partial fulfillment of the requirement For the award of the degree

#### Of

**Bachelor of Engineering in Information Technology**

#### By

# Ms. Mayuri Askar

# Mr. Prashant Mishra

# Mr. Sanket Tinkhade

# Mr. Tushar Zade

## **VIII Sem. B.E. (IT)**

#### Under the guidance of

### *Mr. Amol Gaikwad*



## **DEPARTMENT OF Information Technology**

**Nagar Yuwak Shikshan Sanstha’s**

# YESHWANTRAO CHAVAN COLLEGE OF ENGINEERING,

**(An autonomous institution affiliated to Rashtrasant Tukadoji Maharaj Nagpur University, Nagpur)**

# NAGPUR – 441 110

**2021-2022**

**CERTIFICATE OF APPROVAL**

Certified that the project report entitled “**Optimization and Parallelization of Machine Learning Algorithms for DNA Classification**” has been successfully completed by **Ms. Mayuri Askar, Mr. Prashant Mishra, Mr. Sanket tinkhade, Mr. Tushar Zade** under the guidance of **Prof. Amol Gaikhwad from department** in recognition to the partial fulfillment for the award of the degree of Bachelors of Engineering in Information Technology, **Yeshwantrao Chavan College of Engineering *(An Autonomous Institution Affiliated to Rashtrasant Tukdoji Maharaj Nagpur University).***

### Prof . Amol Gaikwade

Guide

Department of Information Technology

Y.C.C.E ,Nagpur

|  |  |
| --- | --- |
| **Prof. B. U. Bawankar**  In-charge, Projects Department of Information Technology  Y.C.C.E, Nagpur | **Dr. R. C. Dharmik Head,**  Department of Information Technology Y.C.C.E, Nagpur |

Signature of External Examiner Name:

Date of Examination:

## **DECLARATION**

I/WE certify that

* 1. The work contained in this project has been done by me under the guidance of my supervisor(s).
  2. The work has not been submitted to any other Institute for any degree or diploma.
  3. I have followed the guidelines provided by the Institute in preparing the project report.
  4. I have conformed to the norms and guidelines given in the Ethical Code of Conduct of the Institute.
  5. Whenever I have used materials (data, theoretical analysis, figures, and text) from other sources, I have given due credit to them by citing them in the text of the report and giving their details in the references. Further, I have taken permission from the copyright owners of the sources, whenever necessary.

Signature of the Student

Mayuri Askar

Prashant Mishra

Sanket Tinkhade

Tushar Zade

# ACKNOWLEDGEMENT

We take this opportunity to express a deep sense of gratitude towards our guide

**Prof. Amol Gaikwade** for providing excellent guidance, encouragement, and inspiration throughout the project work. Without their valuable guidance, this work would never have been a successful one. He is one of the best mentor, we will always be thankful to them.

We would like to extend our special thanks to our Project Coordinator **Mr. Bhushan U. Bawankar** who has helped us directly or indirectly to complete this project work.

We would like to thank **Prof. R. C. Dharmik**, Head of the department of Information Technology. He was very helpful and encouraging while doing research work.

We would like to thank **Dr. U. P. Waghe**, Principal (YCCE) who has provided all institutional facilities as and when needed.

We would also like to thank all our classmates for their valuable suggestions and helpful discussions and our thanks goes to all the people who have supported us to complete the research work directly or indirectly.

Finally, we are extremely grateful to our parents for their love, prayers, care and sacrifices for educating and preparing us for our future.

**CONTENTS Page No.**

Title Page i

[Certificate of Approval ii](#_TOC_250001)

[DECLARATION iii](#_TOC_250000)

Acknowledgement iv

Table of Contents v

|  |  |  |
| --- | --- | --- |
| List of Figures  Abstract  **Chapter 1 Introduction**   * 1. Overview   2. Problem Statement   3. Thesis Objectives   **Chapter 2 Literature Review**  **Chapter 3 Work done**  **Chapter 4 Results and Discussions**  **Chapter 5 Summary and Conclusions**  **Chapter 6 Appendix**  **Chapter 7 References** |  | vi  vii  1  2  2  3-9    10-11  12-15  16  17-20  21-23 |
|  |  |  |
|  |  |  |
| **LIST OF FIGURES** |
| |  |  |  | | --- | --- | --- | | **Figure No.** | **Figure Title** | **Page No.** | | 3.1 | Flow chart | 17 | | 4.1 | Data Feeding | 19 | | 4.2 | Data Extraction | 19 | | 4.3 | Data visualisation | 20 | | 4.4 | Parameters after passing through ML algo | 21 | | 4.5 | Various parameters comparison of different ML algorithms | 22 | | 4.6 | Visualisation of performance & Time comparison | 22 | |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**ABSTRACT**

DNA classification is a process of determining whether the sequence belongs to a particular pre -existential class .This classification needs huge amount of time and manpower to classify if done manually.This Report uses machine learning algorithms to classify DNA sequences significantly reducing the time and effort.

To Further reduce the time and obtain accurate classification results the ML algorithms are optimized and parallelized using appropriate techniques and NLP(natural language processing).

**Keywords:** Classification,Optimization,Parallelization, DNA, ML, NLP.

**Introduction**

**1.INTRODUCTION**

**1.1 Overview**

At its most basic, Machine Learning uses pre-programmed algorithms that receive and analyze input data to predict output values within an acceptable range. As new data is fed to these algorithms, so to optimize and reduce their time complexity and for gaining the highest accuracy to improve performance gains, we use parallelizing common tasks which may be shared among numerous algorithms for better operations.

Parallel Processing simply means algorithms are deployed across the multiple processors . A parallel algorithm is an algorithm that can execute several instructions simultaneously on different processing devices and then combine all the individual outputs to produce the final result.

Any system that has more than one CPU can perform parallel processing, as well as multi-core processors which are commonly found on computers today. Multi-core processors are IC chips that contain two or more processors for better performance, reduced power consumption and more efficient processing of multiple tasks.

Breaking up different parts of a task among multiple processors will help reduce the amount of time to run a program. The number of hardware increases with parallel processing and with it, the value of the system improves.

But, technological developments have decreased hardware costs to the point where parallel processing methods are economically possible. Parallel processing at a greater level of complexity can be managed by having a multiplicity of functional units that implement identical or various operations together.

**1.2 Problem Statement**

In basic terms, this project classifies DNA sequences in their particular classes whilst demonstrating the accuracy difference between optimized and non optimized machine learning algorithms .It also focuses on time consumption for executing algorithms by running those on different threads rather than on single thread.

In one statement:

Optimization and parallelization of ML algorithms for DNA classification.

**1.3 Thesis Objective**

The objectives of our project are:

1.To classify the DNA sequences in their appropriate class.

2.To show accuracy difference between different ML algorithms.

3.To show the accuracy difference between optimized and non-optimized ML algorithms

4.To show the time difference between parallel and non parallel approach for executing ML algorithms**.**

**Literature Review**

1. **Literature Survey**

In recent years, the Python programming language has gained momentum for scientific computing. Often conventional tools like MatLab are replaced. [1]. It is open to all at no cost because Python is open source, and its portability makes its usability possible on many platforms. The language itself is lightweight, abridged, and highly suitable for quick prototyping, although it is strong enough to write significant applications. Some people don't give it enough credit for its usability and flexibility.

Python can very well be integrated with the C/C++ so that external performance or code-based modules can be easily invoked. Besides, it offers a wide range of scientific libraries, e.g. Processing and analyzing data, plotting and graphical user interfaces [2][3][4]. All these feature makes Python attractive to the scientific public but it has to be parallel to the languages used in large projects. Python is the default implementation and most commonly used. [5], due to the global look in the interpreter, several threads cannot be run at once.

Parallel Processing is another term for speeding up the efficiency of running a program by dividing it into smaller pieces that can be performed simultaneously on multiple processors. [6], In general, each component has its processor. A program running on Q processors can complete Q times faster than a program running on one processor. [7].

D. Meunier et al., [8] NetLogo can be controlled through the programming language of Python. Given Python's growing popularity in computer science, modelers and analysts now have more choices. PyNetLogo features include controlling one of NetLogo's example models from an integrated Python environment and performing a global sensitivity analysis with parallel processing.

Y. Babuji et al., [9] Parsl is a parallel script library that extends Python through fast, scalable, and adaptable encoding parallels. Experimental results on computing in Blue Waters show that Python scripts can run components of just 5 MS overhead, scale to over 250,000 employees across more than 8,000 nodes, and process up to 1200 tasks a second. It has shown multitasking, collaborative, web-based, and machine learning skills in biology, cosmology, and materials science.

D. S. Wahyuni [10] The BayesFactorFMRI tool, written in R and Python, was presented to enable Bayesian second-level analysis and Bayesian meta-analysis for multiprocessing fMRI image data by neuroimaging researchers. This tool accelerates computer-intensive Bayesian fMRI analysis by using multiprocessing. Its graphical user interface (GUI) enables researchers to conduct Bayesian fMRI analysis without the need for computer programming expertise. BayesFactorFMRI can be downloaded for free from Zenodo and GitHub. Neuroimaging researchers who wish to analyze their fMRI data with Bayesian analysis will usually use it, as it is more sensitive than conventional analysis and increases efficiency by spreading analytical tasks across multiple processors.

G. Heine, T et al., [11] Introduced a method for asynchronous streaming. Stream subscriptions are proposed as a tool for monitoring public opinion. A prototype is presented that integrates Twitter sources, Python text processing, and Cassandra storage methods, with three main points elaborated on: 1) A comparison of results in writing techniques. 2) Data parallelization and asynchronous concurrent database writes are used in multiprocessing procedures. 3) Monitoring of public opinion by noun extraction.

D. Datta et al., [12] The performance of parallelized CPUs was compared. Python's Ray library is used to parallelize multicore CPUs. In this project, the benchmark image classification algorithm used is based Convolutional Neural Network. The author attempted to demonstrate the Parallelization of a CPU's multicores which allows for faster training of a model. In this paper, a comparison analysis was conducted between three different Convolutional Neural Network models.

T. Shaffer et al., [13] Native Python functions were proposed at scale, and techniques for dynamically evaluating a minimal collection of dependencies and assembling a lightweight function monitor (LFM) that captures the software environment and manages resources at the granularity of single functions were introduced. The author tests these approaches in various settings, from a campus cluster to a supercomputer, and demonstrates that their advanced dependency management planning and complex resource management strategies outperform the competition.

E. Jonas et al., [14] Introduced the MPI Python connect Academic Journal of Nawroz University (AJNU), Vol.10, No.3, 2021 350 with the standard MPI communication API, known as mpiPython. The author discussed the design issues associated with the implementation of the mpiPython API in this paper. The second part of the paper addressed the node/parallel output to compare mpiPython with other MPI bindings on a Linux cluster.

Galvez et al., [15] CharmPy was introduced as a parallel programming model and application based on the Python programming language. It had many distinguishing features, including a simpler model and API, improved flexibility, and writing anything in Python. Another example is a general-purpose distributed map function that can run independent jobs on multiple nodes simultaneously and supports load balancing. The authors also demonstrated how to use CharmPy to write parallel Python applications that scale to massive core counts on supercomputers and perform similarly to MPI or C++ versions.

R. Eggen et al,. [16] The effect of the global python interpreter lock (GIL) has been examined. To show the effect of GIL, the authors analyze a comparison of python threads to python processes. The GIL leads to sequential execution of threads, while concurrent processing is executed. Processes need more start-up time; it answers the amount of data needed to execute processes faster than threads.

M. R. Rizqullah et al., [17] The middleware in this paper was developed using the Python parallel programming language and installed on a Raspberry Pi 3. The console frame was designed to help people learn the basics of IoT through the transmission and receipt of control data to access sensors or actuators. This middleware transforms a command line for running or accessing the various IoT module features. In order to increase program operating time performance, Python employs multiprocessing or multithreading.

V. Skorpil et al., [18] The paper discussed various methods for parallelizing genetic algorithms with subsequent implementation. For example purposes, the Python programming language is used. Various models of genetic algorithm parallel processing are also provided and described. The Python implementations of the models are then defined and compared using iteration count as a criterion. While individual model output can only be compared to a certain degree, all parallel models outperform the simple serial model.

H. Jan et al., [19] In this article, the NetLogo connector was initially introduced, which connects the NetLogo modeling agent to a Python environment. This was illustrated with one of NetLogo's sample versions. The library SALib Python was used as an example of the more complex tests given in a Python GUI in Sobol's variance-based structural reliability analysis of the model. For better results in the study, the ipyparallel library was used to parallel sequential simulations.

Zhang et al., [20] This paper proposed Quant Cloud, a program that integrates a parallel Python framework with a C++-coded Big Data system. This extensive data framework is built in C++, and the user methods are written in Python. A coprocessor-based parallel strategy underpins the automatic parallel execution of Python code. They have put the program into two popular algorithms: moving window and selfadjusting average movements (ARMA). The Intel Xeon E5 and Xeon Phi processors are thoroughly compared. Their approach to parallelization is almost linear and is suitable for today's multicore processors. The findings show that their method is almost linear.

Sindhu et al.,[21] A Python multi-processing library has implemented a simultaneous implementation of the Max-p problem. The author achieves speeds up to 12 and 19 times with the best sequential algorithm for developing and improving phases utilizing an intuitive multi-lock data structure. In order to validate the algorithms, the author provides detailed experiential results. Real et al.,

[22] This paper has presented Auto Parallel Academic Journal of Nawroz University (AJNU), Vol.10, No.3, 2021 351 which is a Python module that facilitates parallelism and runs on distributed infrastructures. It is built on top of PyCOMP and is sequential, This helps in making it easy to scale up to hundreds of cores for creative purposes. Users can specify the affine loop on sequential methods using the @parallel annotation instead of testifying sequential python code. As it turns out, the generated codes for Choles, LU, and QR algorithms can achieve similar performance without any effort from the programmer. Thus, taking the Auto Pip parallelizes distributed systems one step further.

Z. Rinkevicius et al ., [23] Prsened an Open source software named VeloxChem that was created to measure electronic complicated, real linear response functions for functional theories of Hartree–Fock and Kohn–Sham density. Points to an objective software framework written in Python/C++ layered fashion, VeloxChem enables the time-efficiently prototyping of new techniques without cooperating computational achievement.

V. Canh Vu et al,. [24] In parallel, a genetic programming technique for classifying data patterns for wireless attack detection was presented. The author performed tests on the same computer system configuration, parameters and datasets in order to associate the performance of Karoo GP and standard GP. Karoo GP was, however, implemented alongside the high-speed GPU processing mechanism when the mainstream GP for multi-core CPUs has been used. Karoo GP is much faster than its average GP, according to performance.

S. Khan and A. Latif [25] Proposed solution eliminates this constraint and allows a single machine to run several instances. The SIME method for the measurement of critical clearance time (CCT) and the stability of the rotor angles is measured on a piece of single infinite system equipment (SIME). This method reduces computational time as a parallel factor and dramatically improves the handling and aggregation of the tasks. The approach is generic and possible.

A. V. M. Barone et al., [26] Introduce a broad and diverse Parallel Corpus with its documentation strings ("docstrings") created by scrapping open source repositories on GitHub, with a hundred thousand Python functionalities. The paper defined the fundamental results in neural machine-created translations for the code documentation and code generation tasks.

**Python Parallel Processing and Multiprocessing**

Parallel and multiprocessing algorithms break down significant numerical problems into smaller subtasks, reducing the total computing time on multiprocessor and multicore computers. Parallel programming is well supported in proven programming languages such as C and Python, which are well suited to “heavy-duty” computational tasks. Historically, Python has been

regarded as a strong supporter of parallel programming due to the global interpreter lock (GIL).However, times have changed. Parallel programming in Python is supported by the creation of a diverse set of libraries and packages. This review focused on Python libraries that support parallel processing and multiprocessing, intending to accelerate computation in various fields,

including multimedia, attack detection, supercomputers, and genetic algorithms. Furthermore, we discussed some Python libraries that can be used for this purpose.

**Survey and Performance Test of Python-based Libraries for Parallel Processing.**

By the Fourth Industrial Revolution and the 10 strategic technology of the Gartner Group, Artificial Intelligence(AI) technology was important and affected many areas. One of the ways to accelerate AI services is the Python-based parallel processing library. High-level programming languages such as Python are increasingly used to provide intuitive interfaces to libraries written in lower-level languages and for assembling applications from various components. This migration towards

orchestration rather than implementation, coupled with the growing need for parallel computing (e.g., due to big data and the end of Moore's law), necessitates rethinking how parallelism is expressed in programs.[1] In this paper, take a look at a Pythonbased distributed parallel processing library, one of the ways to accelerate AI services, and use it to compare serial and parallel processing times.

**High-Performance Parallel Computations Using Python as High-Level Language.**

High-performance and parallel computations have always represented a challenge in terms of code optimization and memory usage, and have typically been tackled with languages that allow a low-level management of resources, like Fortran, C and C++. Nowadays, most of the implementation effort goes into constructing the bookkeeping logic that binds together functionalities taken from standard libraries. Because

of the increasing complexity of this kind of codes, it becomes more and

more necessary to keep it well organized through proper software engineering practices. Indeed, in the presence of chaotic implementations, reasoning about correctness is difficult, even when limited to specific aspects like concurrency; moreover, due to the lack in flexibility of the code, making substantial changes for experimentation becomes a grand challenge.

**Work Done**

**3.Work Done**

The workflow of our program is like this

Diagram

Description automatically generated

**Fig 3.1 Flow chart**

**Steps Description**

**Step 1**:- A DNA sequence is given as raw input (in csv or txt format).

**Step 2: -** Data extraction is done. Where K-mers sequences of length six are extracted.

**Step 3**:- Here the extracted data is fed to four machine algorithms. KNN, SVM, RANDOM FOREST, DECISION TREE Linearly, without any parallel processing

**Step 4:-** After the above process is done, This same ML model is processed parallelly by creating individual processes at the same time and running them in the background.

**Step 5:-** Then apply ML models to select the best optimal implementation of a particular application that yields the highest performance based on the execution time of the implementation.

**Step 6:-** After all the processes are done, we have to compare the individual time and result of each algorithm to find out the best algorithm out of all of them.

**Results and Discussions**

**4. Results and Discussions**

Graphical user interface, application, Teams

Description automatically generatedFig 4.1 Data Feeding

Table

Description automatically generated

Fig 4.2 Data Extraction

Chart, pie chart

Description automatically generated

Fig 4.3 Data visualisation

Text

Description automatically generatedGraphical user interface, text, application

Description automatically generated

Fig 4.4 Parameters after passing through ML algo

Chart, line chart

Description automatically generated

Fig 4.5 Various parameters comparison of different ML algorithms

Graphical user interface

Description automatically generated

Fig 4.6 Visualisation of performance & Time comparison

**Future scope and conclusions**

**5.** Future scope and conclusions

This work concludes that optimization of machine learning algorithms include tweaking of hyper-parameters via brute force technique which leads to better results than non optimized implementation of algorithms.

Time for execution of these algorithms is reduced by the use of parallel processing which means to run all algorithms on different threads instead of a single thread, this results in less time consumption as compared to a linear approach.

**Appendix**

**6. Appendix:**

**1. TIME** - time() The time() function returns the number of seconds passed since epoch. For Unix system, January 1, 1970, 00:00:00 at UTC is epoch (the point where time begins). import time seconds = time.time() print("Seconds since epoch =", seconds)

**2. PANDA**- Pandas is built on top of two core Python libraries—matplotlib for data visualization and NumPy for mathematical operations. Pandas acts as a wrapper over these libraries, allowing you to access many of matplotlib's and NumPy's methods with less code. For instance, pandas' .



**3. STREAMLIT**- Streamlit is an open source app framework in Python language. It helps us create web apps for data science and machine learning in a short time. It is compatible with major Python libraries such as scikit-learn, Keras, PyTorch, SymPy(latex), NumPy, pandas, Matplotlib etc.

Logo, company name

Description automatically generated

**4. MATPLOTLIB**- Pyplot is a collection of functions in the popular visualization package Matplotlib. Its functions manipulate elements of a figure, such as creating a figure, creating a plotting area, plotting lines, adding plot labels, etc.

5. SEABORN- Seaborn is a library for making statistical graphics in Python. It builds on top of matplotlib and integrates closely with pandas data structures. Seaborn helps you explore and understand your data.

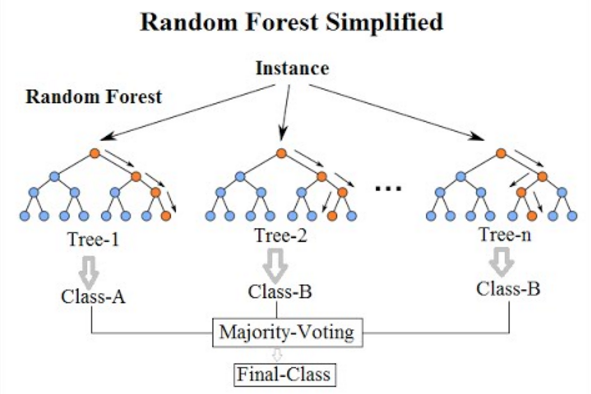
A picture containing text, clipart

Description automatically generated

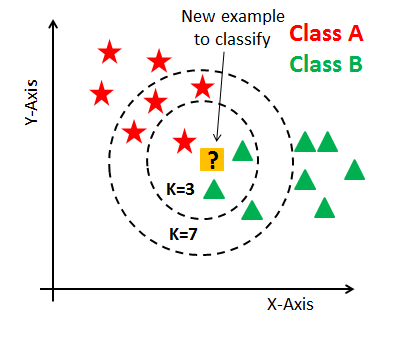
6. COUNTVECTORIZER - CountVectorizer is a great tool provided by the scikit-learn library in Python. It is used to transform a given text into a vector on the basis of the frequency (count) of each word that occurs in the entire text.

7. A random forest classifier.

A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. The sub-sample size is controlled with the max\_samples parameter if bootstrap=True (default), otherwise the whole dataset is used to build each tree.

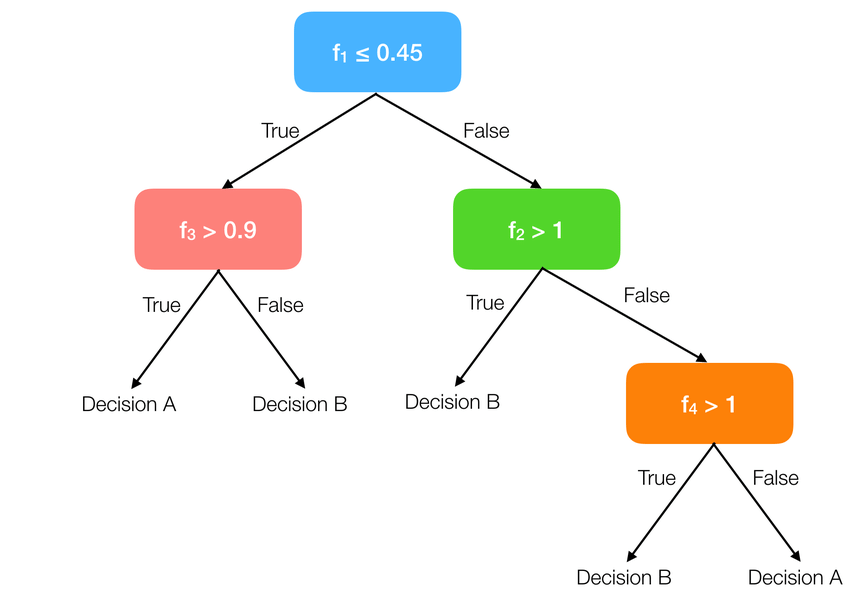


8 . K-Nearest Neighbors, or KNN for short, is one of the simplest machine learning algorithms and is used in a wide array of institutions. KNN is a **non-parametric, lazy**learning algorithm. When we say a technique is non-parametric, it means that it does not make any assumptions about the underlying data. In other words, it makes its selection based off of the proximity to other data points regardless of what feature the numerical values represent. Being a lazylearning algorithmimplies that there is little to no training phase. Therefore, we can immediately classify new data points as they present themselves.



9. Decision tree

Decision tree is **a type of supervised learning algorithm that can be used for both regression and classification problems**. The algorithm uses training data to create rules that can be represented by a tree structure. Like any other tree representation, it has a root node, internal nodes, and leaf nodes.



10. Support Vector Machine(SVM)

Support Vector Machine(SVM) is **a supervised machine learning algorithm used for both classification and regression**. Though we say regression problems as well its best suited for classification. The objective of SVM algorithm is to find a hyperplane in an N-dimensional space that distinctly classifies the data points.

Chart, scatter chart

Description automatically generated

**References**

**7. References:**

1]. S. M. Lim, A. B. M. Sultan, M. N. Sulaiman, A. Mustapha, and K. Y. Leong, “Crossover and mutation operators of genetic algorithms,” Int. J. Mach. Learn. Comput., vol. 7, no. 1, pp. 9–12, 2017, doi: 10.18178/ijmlc.2017.7.1.611.

2]. I. V. Kotenko, I. B. Saenko, and A. G. Kushnerevich, “Architecture of the parallel big data processing system for security monitoring of internet of things networks,” SPIIRAS Proc., vol. 4, no. 59, pp. 5–30, 2018, doi: 10.15622/sp.59.1.

3]. A. Ebrahim, J. A. Lerman, B. O. Palsson, and D. R. Hyduke, “COBRApy: COnstraints-Based Reconstruction and Analysis for Python,” BMC Syst. Biol., vol. 7, 2013, doi: 10.1186/1752- 0509-7-74.

4]. L. D. Dalcin, R. R. Paz, P. A. Kler, and A. Cosimo, “Parallel distributed computing using Python,” Adv. Water Resour., vol. 34, no. 9, pp. 1124–1139, 2011, doi: 10.1016/j.advwatres.2011.04.013.

5]. A. Rogohult, “Benchmarking Python Interpreters,” KTH, Sk. for datavetenskap och Kommun. (CSC), 2016, p. 119, 2016.

6]. S. R. M. Zebari and N. O. Yaseen, “Effects of Parallel Processing Implementation on Balanced Load-Division Depending on Distributed Memory Systems Client / Server Principles,” vol. 5, no. 3, 2011.

7] . M. Wilde, M. Hategan, J. M. Wozniak, B. Clifford, D. S. Katz, and I. Foster, “Swift: A language for distributed parallel scripting,” Parallel Comput., vol. 37, no. 9, pp. 633–652, 2011, doi: 10.1016/j.parco.2011.05.005.

8] . J. Kready, S. A. Shimray, M. N. Hussain, and N. Agarwal, “YouTube data collection using parallel processing,” Proc. - 2020 IEEE 34th Int. Parallel Distrib. Process. Symp. Work. IPDPSW 2020, pp. 1119–1122, 2020, doi: 10.1109/IPDPSW50202.2020.00185.

9] . J. Niruthika and S. Pranavan, “222 implementation of parallel aho-corasick algorithm in python,” no. November, pp. 222–233, 2019.

10] . Y. Babuji et al., “Parsl: Pervasive parallel programming in Python,” HPDC 2019- Proc. 28th Int. Symp. High-Performance Parallel Distrib. Comput., pp. 25–36, 2019, doi: 10.1145/3307681.3325400.

11] . H. Han, “BayesFactorFMRI: Implementing Bayesian SecondLevel fMRI Analysis with Multiple Comparison Correction and Bayesian Meta-Analysis of fMRI Images with Multiprocessing,” J. Open Res. Softw., vol. 9, no. 1, pp. 1–7, 2021, doi: 10.5334/jors.328.

12] . G. Heine, T. Woltron, and W. Alexander, “Towards a Scalable Data-Intensive Text Processing Architecture with Python and Towards a Scalable Data-Intensive Text Processing Architecture with Python and Cassandra,” no. November, 2018.

13]. . D. Datta, D. Mittal, N. P. Mathew, and J. Sairabanu, “Comparison of Performance of Parallel Computation of CPU Cores on CNN model,” Int. Conf. Emerg. Trends Inf. Technol. Eng. ic-ETITE 2020, pp. 1–8, 2020, doi: 10.1109/icETITE47903.2020.142.

14]. . T. Shaffer, Z. Li, B. Tovar, and Y. Babuji, “Lightweight Function Monitors for Fine-Grained Management in Large Scale Python Applications,” pp. 1–11.

15] . H. Park, J. Denio, J. Choi, and H. Lee, “MpiPython: A robust python MPI binding,” Proc. - 3rd Int. Conf. Inf. Comput. Technol. ICICT 2020, pp. 96–101, 2020, doi: 10.1109/ICICT50521.2020.00023.

16] . J. J. Galvez, K. Senthil, and L. Kale, “CharmPy: A Python Parallel Programming Model,” Proc. - IEEE Int. Conf. Clust. Comput. ICCC, vol. 2018-Septe, pp. 423–433, 2018, doi: 10.1109/CLUSTER.2018.00059.

17] . R. Eggen and E. M. Eggen, “Thread and Process Efficiency in Python,” pp. 32–36.

18] . M. R. Rizqullah, A. R. Anom Besari, I. Kurnianto Wibowo, R. Setiawan, and D. Agata, “Design and implementation of middleware system for IoT devices based on raspberry Pi,” Int. Electron. Symp. Knowl. Creat. Intell. Comput. IES-KCIC 2018 - Proc., pp. 229–234, 2019, doi: 10.1109/KCIC.2018.8628528.

19] . V. Skorpil, V. Oujezsky, P. Cika, and M. Tuleja, “Parallel Processing of Genetic Algorithms in Python Language,” Prog. Academic Journal of Nawroz University (AJNU), Vol.10, No.3, 2021 354 Electromagn. Res. Symp., vol. 2019-June, pp. 3727–3731, 2019, doi: 10.1109/PIERS-Spring46901.2019.9017332.

20] . H. Jan, J. H. Pynetlogo, L. Netlogo, M. Jaxa-rozen, and J. H. Kwakkel, “Article PyNetLogo : Linking NetLogo with Python Reference PyNetLogo : Linking NetLogo with Python,” vol. 21, no. 2.

21] . P. Zhang, Y. Gao, and X. Shi, “QuantCloud: A software with automated parallel python for Quantitative Finance applications,” Proc. - 2018 IEEE 18th Int. Conf. Softw. Qual. Reliab. Secur. QRS 2018, pp. 388–396, 2018, doi: 10.1109/QRS.2018.00052.

22] . V. Sindhu, “ScholarWorks @ Georgia State University Exploring Parallel Efficiency and Synergy for Max-P Region Problem Using Python,” 2018.

23] . F. Real, A. Batou, T. Ritto, and C. Desceliers, “Stochastic modeling for hysteretic bit–rock interaction of a drill string under torsional vibrations,” J. Vib. Control, p. 107754631982824, 2019, doi: 10.1177/ToBeAssigned.

24] . Z. Rinkevicius et al., “VeloxChem: A Python-driven densityfunctional theory program for spectroscopy simulations in high-performance computing environments,” Wiley Interdiscip. Rev. Comput. Mol. Sci., vol. 10, no. 5, pp. 1–14, 2020, doi: 10.1002/wcms.1457.

25] . V. Canh Vu and T. H. Hoang, “Detect Wi-Fi Network Attacks Using Parallel Genetic Programming,” Proc. 2018 10th Int. Conf. Knowl. Syst. Eng. KSE 2018, pp. 370–375, 2018, doi: 10.1109/KSE.2018.8573378. 26] . S. Khan and A. Latif, “Python based scenario design and parallel simulation method for transient rotor angle stability assessment in PowerFactory,” 2019 IEEE Milan PowerTech, PowerTech 2019, pp. 1–6, 2019, doi: 10.1109/PTC.2019.8810949.

26] . A. V. M. Barone and R. Sennrich, “A parallel corpus of Python functions and documentation strings for automated code documentation and code generation,” arXiv, 2017