A Machine Learning Approach for Predictive Selection of Efficient CPU Scheduling Algorithms

THESIS

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE AWARD OF THE DEGREE OF

MASTER OF TECHNOLOGY
in
COMPUTER SCIENCE ENGINEERING

Submitted by

Abhishek kumar
(2K23/CSE/16)

Under the supervision of
Dr. Manoj Sethi

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING
DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Bawana Road, Delhi 110042
MAY, 2025

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING

DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering)

Bawana Road, Delhi-110042

CANDIDATE'S DECLARATION

I, **Abhishek Kumar**, Roll No – **(2K23/CSE/16)** student of M.Tech (**Department of Computer Science Engineering**), hereby declare that the project Dissertation titled **—A MACHINE LEARNING APPROACH FOR PREDICTIVE SELECTION OF EFFICIENT CPU SCHEDULING ALGORITHMS** ‖ which is submitted by me to the **Department of Computer Science Engineering**, Delhi Technological University, Delhi in partial fulfilment of the requirement for the award of degree of Master of Technology, is original and not copied from any source without proper citation. This work has not previously formed the basis for the award of any Degree, Diploma, Associate ship, Fellowship or other similar title or recognition.

Place: Delhi

Abhishek Kumar **Date: 31.05.2025**

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING

DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering)

Bawana Road, Delhi-110042

CERTIFICATE

I hereby certify that the Thesis report titled **"A MACNINE LEARNING APPROACH FOR PRDICTIVE SELECTION OF EFFICIENT CPU SCHEDULING ALGORITHMS"** which is submitted by Abhishek Kumar, Roll No. 2K23/CSE/16, Department of Computer Science Engineering, Delhi Technological University, Delhi in partial fulfilment of the requirement for the award of the degree of Master of Technology, is a record of the project work carried out by the student under my supervision. To the best of my knowledge this work has not been submitted in part or full for any Degree or Diploma to this University or elsewhere.

Place: Delhi **Dr. Manoj Sethi**

Date: 31.05.2025 **SUPERVISOR**

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING

DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering)

Bawana Road, Delhi-110042

ACKNOWLEDGEMENT

I wish to express my sincerest gratitude to **Dr. Manoj Sethi** for his continuous guidance and mentorship that he provided me during the project. He showed me the path to achieve my targets by explaining all the tasks to be done and explained to me the importance of this project as well as its industrial relevance. He was always ready to help me and clear my doubts regarding any hurdles in this project. Without his constant support and motivation, this project would not have been successful.

Place: Delhi **Abhishek Kumar**

Date: 31.05.2025 **(2K23/CSE/16)**

Abstract

Basic for the efficient operation of multi-tasking operating systems is good CPU scheduling .To accomplish this, we have designed and developed a tool called CPU Scheduling Simulator which can implement and compare a variety of scheduling algorithms including the First Come First Serve (FCFS), Shortest Job First (SJF), and Round Robin (RR) among others. Its evaluations of these algorithms are based on key performance indices such as Turnaround Time (TAT), Operations Duration (OPD) and Response Time (Response), offering details down to a level finer than you might think into just how well each one works for any given set of applications or circumstances.

Optimizing the performance of operating systems is crucial, but with modern computational needs this has to be done efficiently. Traditional scheduling algorithms (FCFS, SJF, Priority, Round Robin) can produce inconsistent results in varying system environments. According to this research, a machine learning based framework has been presented to dynamically predict which CPU Scheduling algorithm is best suited for a given process context. By using models such as Support Vector Machine (SVM), Logistic Regression (LR) and Stochastic Gradient Descent (SGD), the system analyzes process attributes such as arrival time, burst time, priority and quantum, and selects the most effective scheduling strategy.

Of all the models tested, SVM achieved the highest prediction accuracy: 94.56%. It is better at catching up on slack sites and in queuing networks. Our results highlight the potential for inserting smart forecasting systems into operating systems to lessen turnaround and save resources. Thus it is in keeping with both sustainable and flexible computer environments.

TABLE OF CONTENTS

List of Tables

List of Figures

List of Abbreviations

ML - Machine Learning

SGD – Stochastic Gradient Decent

LR- Logistic Regression

SVM - Support Vector Machine

ROC - Receiver Operating Curve

AOC - Area under the Curve.

FCFS – First come first serve

SJF – Shortest job first

SRTF- Shortest remaining time first

PSA- Priority scheduling Algorithms.

RR- Round Robin

CHAPTER 1

1. INTRODUCTION

1.1 Background and motivation

In today's computer realm, the operating system is all important in running multiple tasks at once smoothly. Of its core functions, process scheduling stands out as a primitive mechanism to regulate how processes reach the Central Processing Unit (CPU). As computer systems cope with larger numbers of simultaneous processes, this function becomes increasingly important. The efficiency and preference of the scheduler have a direct bearing on major system performance indicators as CPU utilization, process turnaround time, throughput, waiting time and total responsiveness.

In essence, process scheduling means to determine the detailed order and timing with which a CPU process shall be allowed to run. The scheduler must achieve a balance between fairness – insuring no process is ever indefinitely postponed – and efficiency & maximizing the use of CPU cycles in actual productive work. If this balance is not maintained, various bad performance results may ensue, such as unreasonably long waits for processes, wasteful use of CPU time and, at worst, process starvation-where some processes could never get any time on the CPU Over the past several decades, all kinds of classical CPU scheduling methods have been developed and spread throughout operating systems.

- **First-Come-First-Serve (FCFS):** As the simplest scheduling method, FCFS processes arrived in the queue will be executed in the order that they were received, as the most basic techniques do not take execution time or even the priority of the request into consideration. The uncomplicated nature of FCFS can lead to inefficiencies like the "convoy effect," which occurs when shorter processes are queued behind longer ones.
- **Shortest Job First (SJF):** SJF operates on the premise of minimizing waiting times by completing the process of the shortest estimated duration first. Although,

in order to adhere to this logic, advance knowledge of the processes needed to be completed must be available, and that is not realistic.

- **Round Robin (RR):** This is also a time-sharing technique, whereby RR schedules processes in a circular order, and each is allocated a fixed time period or 'quantum.' Responsiveness is the main priority of this approach. On the downside, there will be costs incurred, albeit small, due to interruptions that lead to context switching.
- **Shortest Remaining Time First (SRTF)**: This approach expands SJF pre-emotively, in that SRJF will always select the process that will take the least time to complete relative to the other processes. In some situations this improves efficiency, but that can make the system more complex and the amount of unused "work" increases.
- **Priority Scheduling Algorithm (PSA):** In this type processes are assigned different priority numbers and the CPU is allocated to the process that has the highest priority and is in the ready state. Matching this method to different levels of urgency could lead to starvation of lower priority processes unless aging is used.

These algorithms have shaped the history of CPU scheduling in several operating systems. The clear-cut, deterministic style provides ease of implementation and framework. However, the very static attributes of these systems are their greatest drawback. Most tend to function based on rigid logic which does not change in response to dynamic workload conditions or real-time system states. This lack of flexibility becomes more evident as the complexity and heterogeneity of computing workloads increases.

Contemporary systems need to accommodate a broad spectrum of processes, from time-sensitive real-time applications to interactive ones that require immediate responsiveness, and compute-heavy batch jobs that prefer high throughput. Such diverse and sometimes conflicting requirements cannot be optimally met by rigid static scheduling policies. For instance, a strategy designed for CPU-intensive batch jobs would result in intolerable latency for real-time processes. In addition, the ever-increasing use of virtualization, cloud computing, and multi-core architectures adds yet more layers of complexity and variability to workload behavior.

To meet these challenges, there is a growing effort to create smarter, more flexible scheduling approaches that can adjust decisions based on the system conditions and workload characteristics. Advances in Machine Learning (ML) offers promise for this change. Unlike traditional scheduling algorithms that follow a fixed approach, ML methods examine the data and make use of sophisticated heuristics and provide data-driven, precise predictions.

In the case of CPU scheduling, an ML model can use many process attributes like arrival time, burst time, priority, and even time quantum to calculate the specific scheduling algorithm that will be optimally advantageous. With appropriate training on diverse datasets depicting numerous scheduling situations, ML-based schedulers can choose or suggest algorithms tailored to specific system objectives like minimizing wait time, maximizing throughput, or other desired performance levels.

This change from rigid to flexible scheduling introduces new opportunities such as:

1. **Adaptability:** Responsiveness can be achieved with ML-based models that modify algorithms in an expedited manner according to the dynamic changes in workloads, thus eliminating latency for time-critical tasks.
2. **Enhanced Algorithm Matching:** With better matching algorithms according to processes, resource wasting on the CPU is addressed, hence improving avoidance of performance degradation or buildup.
3. **Innovation Balance:** Intelligent algorithms could use these frameworks to manage task starvation, making sure that resource balancing consideration is exercised without much balance disturbance.
4. **Robustness:** Adjustment of pre-established ML static models requires added work if the complexity of the system rises. Unlike handcrafted methods, ML from experience requires less effort utilizing data.
5. **Agility:** As software and hardware evolve, new features and system metrics become available for the ML scheduler algorithm allowing backward adaptation for the algorithms to be incorporated.

All these benefits notwithstanding, challenges arise from using machine learning for scheduling such as explain ability and verifying system safety, relevant training data, productiveness, and overhead manageability. These problems are best solved using processes of collecting or simulating events, filtering useful data, and supervised learning for predicting the most appropriate scheduler algorithm to the set of processes presented.

1.2 Problem Statement

CPU scheduling is one of the cornerstones of today's operating systems, since it determines how and for what time multiple processes are allowed to access the CPU. Decades-long research work on scheduling algorithms like FCFS, SJF, RR, SRTF, PSA etc. have been the basis of process management for decades . These algorithms are well-defined and deterministic as they direct the CPU on how to assign time slices to competitive processes to improve CPU utilization, average waiting time, turnaround time, and system throughput, among other performance parameters.

These traditional scheduling algorithms have been very successful and are rather simple, yet they work under set, predetermined rules and assumptions. For example, FCFS is based on the order in which processes arrive, while SJF favors processes with shorter burst times. Round Robin rotates between processes using a fixed time quantum, while Priority Scheduling only concerns itself with process priority values. While these approaches have been successful in controlled or reasonably predictable environments they are inflexible and less adaptable to the complexity and variation of modern workloads.

The nature of processes in a computing system, such as a PC or a cloud computing system, is very heterogeneous and the nature of such processes is becoming more dynamic. Processes can vary greatly in terms of arrival statistics, computation time, and priority and responsiveness requirements. As an example, whereas real-time applications are delay sensitive and must meet hard deadlines while executing in a short amount of time, background batch jobs may be delay tolerant but may also have long CPU bursts. Multimedia and interactive applications require a relatively constant allocation of CPU cycles in order to remain responsive, whereas data-intensive scientific computing might include long-running compute-intensive jobs. Static advancing scheduling algorithms have not been effective in handling this type of heterogeneity.

The use of a single, static scheduling policy for all workloads can lead to substantial underperformance and inefficiency. The same scheduling policy may perform well on one class of processes, while performing poorly in different class of workloads. As a case in point, FCFS may lead to increased waiting times for short processes when longer jobs are serving on the CPU, whereas SJF might lead to starvation of longer processes. Also, Round Robin suffers from the fixed time quantum problem: If the time quantum is too small, there is excessive context switching; if too large, response time will be increased. Not being able to decide or modify a scheduling strategy to execute dynamically necessitates a lot of real-time situational adaptation, which is very challenging.

Plus, conventional approaches ignore historical information that could improve system performance and aid in decision-making. Given the growth of data-driven system management concepts, rigid and static rule-based systems that lack learning capabilities are becoming increasingly obsolete. Numerous automatic systems suffer from inability to adapt past obtained results for schedule into new strategies based on already gathered data. Such approaches are certain to miss the possibility available in the changing computer workload environment.

Process features are inputs to the classifiers which are trained on labeled datasets, where each label represents the most optimal scheduling strategy identified through heuristics or empirical approaches. The intention of the model is to achieve good performance across previously unseen workloads so that it enables automatic adaptive algorithm selection based on attributes of the process without human involvement.

The above information outlines the approach, which offers promises of several benefits such as the following:

1. Adaptability: As opposed to fixed scheduling strategies, the model adapts decisions in real time based on previous data patterns and current process attributes. This improves resource utilization with regards to the CPU.

2. Enhanced Efficiency: The system is capable of optimizing waiting time, turnaround time, and even throughput by skillfully predicting ideal algorithms alongside meeting responsiveness and fairness with improved efficiency.

3. Generalized: The system aims to achieve great performance across varying conditions by effective training and validation instead of being constricted to rigid boundaries through defined fixed rules. With bounded complexity and diversity in scenarios, the model becomes scalable, befitting distributed and multi-core computing infrastructures.

1.3 Objectives

The objective of this study is to provide a CPU scheduling in a intelligent adaptive framework based on machine learning techniques. Though these CPU scheduling algorithms have been shown to work well for a targeted purpose, the problem is in the rigidity and inflexibility of these algorithms when used in combination with diverse and complex heterogeneous workloads.

This project intends to address these issues by developing a predictive model that can adaptively select which scheduling algorithm to use based on real time parameters of the processes themselves. This work aims to be defined in both scope and depth by the following specific objectives so that the goal. Can be accomplished in a systematic way.

1.3.1 Generation of Synthetic Dataset Reflecting Process Diversity

An achievable aim is to develop a synthetic dataset that represents the enormous variety of processes and their attributes faced in contemporary computing facilities. Because of the lack of publicly accessible datasets designed for the purpose of predicting scheduling algorithms, the absence of such datasets makes it imperative to create realistic, controlled, and synthetic data. This dataset aims to capture the essential characteristics of the processes such as core features like arrival time, burst duration, priority level, and time quantum which serve as critical parameters for conventional CPU scheduling algorithms as well as multi-level scheduling algorithms. The dataset’s variability is intended to capture not only short, high-priority real time processes but also long-running low-priority background tasks. By incorporating such dataset the designed processes will provide a solid base for training machine learning algorithms meant for generalization across different workload profiles.

1.3.2 The Creation of a Rules-Based Strategy for Identifying Algorithms via Labeling

Supervised learning requires that each instance of data is associated with a desired label that represents the scheduling algorithm appropriate for that process/workload state. A secondary objective involves developing a labeling system that is heuristic based on scheduling principles. These heuristics are based on known algorithm preconditions such as assigning high priority processes to priority scheduling, short job first for serving preemptively and time quantum for round robin to be applicable. This approach to labeling simulates the behavior of a very smart scheduling agent as to be able to have clear ground truths for training machine learning. The rules are the process decisions of the schedule made under different conditions of the process, and thus have the more complex task of matching features for optimal scheduling of different processes.

1.3.3 Implementation and Comparison of Multiple Machine Learning Classifiers

In order to have a complete exploration of the applicability and potential of machine learning in the prediction of scheduling algorithms, this study uses various classifiers with different learning paradigms. A third objective is to train and test three state of the art classification models: Stochastic Gradient Descent (SGD), Logistic Regression and Support Vector Machines (SVM). Each model has its own set of features, for instance SGD is computationally light and allows online learning, Logistic Regression gives interpretable probabilistic prediction, SVM captures complex, non-linear relationships using kernel functions. Such an approach allows for a holistic comparison of their performance, robustness and computational needs in the context of scheduling.

1.3.4 Quantitative Performance Assessment Using Robust Evaluation Metrics

An important aim is to properly test the predictive capabilities of the ML machine learning models through different evaluation metrics. Among these metrics are accuracy as the fraction of correct predictions, log loss to measure confidence in the predictions while applying a cost for predicting with high confidence but being incorrect, confusion matrices that allow to visualize performance of each class individually, and Receiver Operating Characteristic (ROC) curves together with Area Under Curve (AUC) scores which evaluate false positive and true positive rates as a trade-off. These metrics combined give a more complete understanding of the ability of these models to predict the appropriate scheduling algorithm given different process situations. This assessment is useful not only in confirming the models' performance but also in feed backing in to iterative enhancements and hyper parameter tuning.

1.3.5 Visualization of Training Dynamics and Model Convergence

Optimizing a model requires understanding how the machine learning algorithms learn, as this directly affects their efficiency and stability. Over and above, our objective is to visualize certain milestones pertaining to training a model, particularly concerning the SGD classifier because it is heuristic and learns via epochs iteration. The analysis looks into training loss with respect to epochs in order to gain understanding of their convergence behavior, over fitting, under fitting, and proper learning rates. Confusion matrices and ROC curves complement this analysis by visualizing classifier output in relation to label descriptions and thus help explain the boundaries set by the classifiers. Such efforts improve the reliability, understanding, and dissemination of analytical communication results to the technical and non-technical audience alike.

1.3.6 An Example on Integrating Machine Learning for Improvement of Scheduling Decisions in Real-time Systems

With this project, we intend to showcase the feasibility and advantages of incorporating machine learning into operating system schedulers for real-time decisions. The research goal focuses particularly on the intelligent scheduling systems that are capable of automatically selecting the most suitable CPU scheduling algorithm from a wide range of available algorithms to maximize the efficiency of the CPU, minimize waiting and turnaround times, and improve system responsiveness. We hope to offer further pioneering research ideas and methods for next-generation operating systems that autonomously optimize their workload process management in heterogeneous and changing environments.

1.4 Overview of the Methodology

This part describes the entire process of creating and testing a predictive model for selection of CPU scheduling algorithms based on machine learning. The methodology combines simulation of data, feature selection, algorithm assignment, model training, assessment, and representation in order to create a responsive scheduler that adjusts its strategy based on the specifics of the processes.

1.4.1 Developing Synthetic Data

Since no datasets exist publicly which are customized for CPU scheduling prediction, the initial task is to create synthetic workload data that aims to capture a variety of operational scenarios encountered in an OS. This synthetic dataset comprises several process features important to scheduling such as:

- **Arrival Time:** Denotes the time a process arrives into the ready queue, capturing simulated real-time job submission.
- **Burst Time:** Refers to the total time a process will require CPU execution for.
- **Priority Level:** Corresponds to a number expressing the relative importance or urgency of a process and so preference enabled scheduling impact.
- **Time Quantum:** Corresponds to the unit of time given to processes in time sharing algorithms such as Round Robin.

To model the variability present in real life workloads, each feature is assigned int values randomly within specified ranges. For example, arrival times are bounded between 0 - 100 time units to simulate processes arriving at different periods during the course, while burst times range from 1 - 100 time units to accommodate both short and long running jobs. Priority levels range from high (1) to low (5), while time quanta vary from 1 to 20 units simulating different scheduling contexts.

1.4.2 Ground Truth Label Creation with Heuristic Rules

In determining the best fit scheduling algorithm to individual processes, ground truth labels necessitating the greatest accuracy per algorithm are assigned through heuristic rules derived from classical scheduling techniques. These heuristic rules assume realistic scheduler actions based on the specific features of the individual processes:

- Processes that have very high priority values (e.g., priority ≤ 2) are regarded as best fitted for the PSA – priority scheduling algorithm.
- Processes that have burst times below median value are tagged as ideal candidates for shortest job first algorithm because they are likely to benefit from SJF.
- Processes that have time quantum values lower than median value are affiliated with round robin scheduling as lower quantum duration enhances performance.
- If a process has come in before the average, it is made First Come First Served (FCFS).
- Processes that came in later than this are executed in the order of Shortest Remaining Time First (SRTF).

This rule-based method of label generation gives an explicit mapping from process features into scheduling strategies, thus enabling supervised training where models learn from input features and their outputs are these labels.

1.4.3 Feature Transformation and Dataset Preparation

Before data is fed to machine learning algorithms at all scales feature scaling conveniently puts input variables within a range of numerical value and thus helps model convergence zn-score normalization method now Standardizes the characteristics of each feature, so that each has zero mean and unit variance. This ensures that features with large numeric ranges (like burst time) do not overwhelm the training of models when compared to small-scale ones (such

After that, the dataset is divided into a training set and some testing sets, typically about 70%. The training set is used to fit the model.

1.4.4 Training of the Selected Machine Learning Model

To evaluate performance on a broader scope, various methods of machine learning are conducted:

Classifier of Stochastic Gradient Descent (SGD): Classifier of Stochastic Gradient Descent (SGD) is a scalable, efficient strategy for training linear classifies, like logistic regression and support vector machines. Employing such large datasets, SGD outperforms batch gradient descent which computes the gradient using the entire training data set, as it updates the model parameters using a single example or mini-batch during each iteration. This subsampling strategy reduces computations and memory requirements which is critical where data is continuously received or when the amount of input data is massive.

$$\theta := \theta - \eta \cdot \nabla_{\theta}\mathcal{L}(\theta; x_i, y_i)$$

Where:

- θ - model parameters (weights),
- η - learning rate,
- $\mathcal{L}$ - loss function (e.g., logistic loss),
- x_i, y_i Represent the input features and corresponding label.

Important highlights and features:

- Incremental Learning: This feature is useful for dynamic or non-memory data.
- Flexibility: Offers support for multiple loss functions such as hinge, log, and modified Huber.
- Regularization: L1 and L2 regularization are applied to control over fitting.
- Scalability: Suits well with sparse and high dimensional data.

The parameters of the SGD classifier are continuously updated through epochs, which further reduces classification errors. Its convergence is impacted by learning rate, regularization, and batch size.

Logistic Regression: The cornerstone of algorithms in machine learning can be seen in Logistic Regression, which is used primarily for binary classification problems. Unlike linear regression which forecasts a value that is continuous in nature, logistic regression predicts the likelihood of an input belonging to a specific class. This is done using the logistic or sigmoid function, which takes as input a linear combination of features and outputs a probability value between 0 and 1. The definition of sigmoid function is as follows:

$$\sigma(z) = \frac{1}{1 + e^{-z}}, \quad \text{where } z = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n$$

Here, β_i are the model coefficients and x_i are the input features. The model then classifies an instance based on whether the output probability exceeds a threshold (typically 0.5).

Core Functions and Benefits:

- **Probabilistic Output:** Offers probability results that can be set to a threshold for classification or employed in ranking.
- **Interpretability:** Each coefficient indicates the level of influence on the outcome based on the corresponding feature.
- **Optimization:** Uses maximum likelihood estimation (MLE) for parameter calculations and subsequently employs gradient techniques for training**.**
- **Baseline Model:** Remains a reliable benchmark because of the its simplicity and effectiveness on linearly separable data.

Logistic regression has a flexible relationship with feature interactions and the output's log-odds, but it assumes a linear connection. Regardless, it handles many practical problems well and is frequently the first model used in classification tasks.

Support Vector Machine (SVM) with Radial Basis Function (RBF) Kernel:

Support Vector Machine (SVM) is a powerful supervised learning algorithm used for classification and regression tasks. The core principle behind SVM is to find a hyper plane that best separates data points of different classes by maximizing the margin — the distance between the hyper plane and the nearest data points from each class (support vectors).

In cases where the data is not linearly separable in its original space, SVM utilizes kernel functions to transform the input space into a higher-dimensional feature space where a linear separation becomes feasible. One of the most commonly used kernels is the **Radial Basis Function (RBF)** kernel, defined as:

$$K(x_i, x_j) = \exp\left(-\gamma \parallel x_i - x_j \parallel^2\right)$$

Where:

- x_i and x_j are feature vectors,
- γ is a kernel parameter that determines the influence of a single training example.

The RBF kernel enables SVM to capture non-linear relationships by measuring similarity based on the distance between data points. It effectively creates flexible decision boundaries that can adapt to complex patterns in the data.

Key Functions and Advantages:

- **Non-linearity**: Allows modeling of complex, non-linear decision surfaces.
- **High Accuracy**: Often achieves high classification performance, particularly on well-processed datasets.
- **Robustness**: Effective when the number of dimensions is greater than the number of samples and in high-dimensional spaces
- **Regularization**: use parameter C to regulate the trade-off between minimizing classification error and optimizing the margin.
- Each model is trained on the scaled training data, with hyper parameters tuned to achieve the best predictive performance. For instance, the SGD classifier's learning rate and number of epochs are selected to ensure stability, while SVM kernel parameters such as gamma are adjusted to balance bias and variance.

1.4.5 Model Evaluation Metrics

Performance evaluation is conducted using multiple complementary metrics to provide a holistic assessment:

- **Accuracy:** Measures the proportion of correctly classified scheduling algorithms.

- **Log Loss (Cross-Entropy Loss):** Quantifies the uncertainty of predictions, penalizing incorrect confident predictions more heavily.
- **Confusion Matrix:** Visualizes true positives, false positives, true negatives, and false negatives, enabling analysis of class-specific errors.
- **Receiver Operating Characteristic (ROC) Curve and Area Under Curve (AUC):** Assess the trade-off between true positive rate and false positive rate across thresholds, especially useful for multi-class classification with binarized labels.

These metrics are computed on the test set to evaluate how well the models generalize beyond the training data.

1.4.6 Visualization and Interpretation

Visual representations aid in understanding model behavior and training dynamics:

- **Loss Curve:** For the SGD classifier, plotting log loss over training epochs reveals the convergence pattern and helps diagnose under fitting or over fitting.
- **Confusion Matrix Heat maps:** Highlight misclassification trends and potential biases toward certain scheduling algorithms.
- **ROC Curves:** Illustrate classifier discrimination capability and facilitate comparison across models.

Such visual tools provide both quantitative and qualitative insights, supporting iterative refinement of models and feature engineering.

CHAPTER 2

2. LITERATURE REVIEW

2.1 Introduction

In terms of resource management, scheduling is considered as one of the building blocks of an OS as it decides the manner in which processes are executed by the CPU. Throughout the years, numerous classical algorithms have been designed to improve some specific metrics, including CPU usage, waiting time, response time, and turnaround time. With the increasing complexity in system architectures and workloads being more diverse, the weaknesses of such rigid approaches are becoming more apparent. A great deal of recent work has started looking into dynamic and intelligent scheduling strategies, particularly those that incorporate machine learning (ML) for adapting to different runtime conditions. This chapter reviews essential works in both conventional methods of scheduling and the increasing domain of scheduling enabled by machine learning (ML), arguing the merit, gaps, and relevance to the focus of the research I am presenting.

2.2 Traditional CPU Scheduling Techniques

The backbone of an operating system's processes is based on the classical scheduling algorithms which, have stood the test of time. There are a number of techniques, but some which are common include the following:

- **First-Come-First-Serve (FCFS):** This algorithm is non-preemptive and processes get scheduled as they arrive in order. It is simple and fair, but it gets complex when waiting time average gets high, due to the "convoy effect" wherein short tasks are forced to wait behind longer ones.
- **Shortest Job First (SJF):** This method chooses the process which has the shortest burst time. Average waiting time drops significantly using this method, but is considered less efficient as one needs to know how long all tasks will take, runs into starvation for longer tasks, and is not very smart.
- **Round Robin (RR):** Suitable for time-sharing systems, RR allocates CPU time in

a circular manner using a constant time slice. It promotes equity although can hurt efficiency if the time slice is set poorly.

- **Shortest Remaining Time First (SRTF):** SRTF allows preemption of ongoing processes, system resources can be redirected to processes with shorter execution time. It enhances responsiveness but incurs significant costs from constant context switching.
- **Priority Scheduling**: Prioritization is assigned to tasks according to their level of urgency or importance. While beneficial for important tasks, it can leave low-priority tasks vulnerable to starvation.

Even though these strategies have clear theoretical bases, they are constrained by being overly structured and rule bound. Such frameworks do not evolve with the dynamic environment, constraining them further. They work optimally in static surroundings but do not adapt to runtime changes, reasoning workloads that do not vary in nature.

2.3 Shortcomings of Static Scheduling Methods

These static approaches have been shown to be inefficient in numerous ways. It is clear from an extensive body of research that no single scheduling algorithm consistently outperforms others across all situations. Considerable attention needs to be devoted to choosing an optimal strategy because it has to align with specific characteristics of the workload, which includes patterns of process arrivals, burst times, and system load.

Researchers like Silberschatz et al. in Operating System Concepts remark that self-adjustment has not been incorporated in classical scheduling mechanisms. Today’s workloads range from user-interactive processes to compute-intensive background tasks and real-time systems, each having unique scheduling behavior preferences. Having a ‘one-approach-for-all’ solution is problematic as it leads to increased response times or decreased throughput when conditions do not match. Additionally, some hybrid and multilevel queue approaches attempt to solve variability by mixing several scheduling algorithms or classifying jobs into different classes. Even to these frameworks lack the ability to learn and infer the optimal approach from data.

2.4 Rise of the Development of Intelligent Scheduling and Its Integration with ML

Shrubbing the most recent research focuses on the flexible scheduling problem with the use of AI and machine learning. These methods can analyze past data and make projections on which scheduling plan would be optimal for a specific set of processes.

2.4.1 Rule-Based and Heuristic Systems

The first steps toward adaptive scheduling were made using heuristics. For instance, some schedulers implementing SJF or RR switch to a different algorithm based on a system load or peer defined levels. While these models provided a fair dash of adaptability, they also required user-edited rules which meant lower data-based generalization effectiveness than needed.

2.4.2 Machine Learning Models

Some authors suggest that the choice of the best algorithm could be made by using supervised models like support vector machines, neural networks, and decision trees to classify the scheduling scenarios.

A decision tree classifier using process features for training to choose among SJF, RR and PSA has been used by Jain et al. They found that their results were better than the results they would have obtained using any of the static algorithms alone.

A Dynamic Quantum Determination model based on Neural Network was presented by Ahmed et al. in 2020, to estimate the quantum time dynamically in round robin scheduling. Such method was effective because it minimized average waiting time and turnaround time.

Mohan and Verma, 2021 used Support Vector Machines, SVMs to classify workloads into scheduling classes, demonstrating that ML models could be superior to hybrid deterministic models given the right context.

It is concluded by these works that using ML-based models permits effective generalization to other types of workload and allows for real time adaptations of the scheduling policy, greatly improving responsiveness and efficiency of the system.

2.5 Creating Synthetic Data and Simulation for Scheduling Algorithm Research

Most studies focus on synthetic data generation due to the deficit of publicly available datasets on CPU scheduling. Several researchers imitate different processes attributes such as burst time, priority, and arrival time using stochastic distributions or workload models to conduct controlled experiments on ML methods. This framework enables testing under a variety of conditions e.g. light vs. heavy system load or CPU-bound vs. I/O-bound tasks. Regardless, the effectiveness of such experiments is based on the heuristic bounds of the algorithm's best scheduler and synthetic data with which the experiment is conducted.

CHAPTER 3

3. METHODOLOGY

3.1 Overview

In this section, I discuss the CPU scheduling problem and how I have made a machine learning based predictive framework to solve it. The sole purpose is to allow the intelligent system to choose the best algorithm for a given process. The optimization of CPU scheduling impacts the overall system responsiveness, user satisfaction, and system performance. Older scheduling policies operate using static governing algorithms which to some extent do not accommodate shifts in workloads. In this work, I attempt to solve this problem using supervised machine learning to create an adaptable smart recommendation system for scheduling.

The methodology includes multiple stages which are interrelated:

1. Synthetic Data Generation: The synthetic process characteristics are used to create a dataset that is broad and representative of reality.

2. Heuristic Labeling: Scheduling labels are ascribed to each process instance based on specific domain knowledge heuristics.

3. Data Preprocessing: The dataset is scaled and split into training and testing groups.

4. Model Training and Tuning: Three machine learning models are used and optimized: Support Vector Machines (SVM), Logistic Regression (LR), and Stochastic Gradient Descent (SGD).

5. Evaluation and Visualization: Model performance is evaluated with accuracy, ROC curve analysis, confusion matrix metrics, and log loss plots.

3.2 Generation of Synthetic Data

The lack of dataset documenting the process level details of CPU scheduling information necessitated the creation of synthetic data. This was done for processes the system behavior in real life scenarios is diverse, which is a distinguishing feature of many processes. In this regard, the Python library NumPy was used to create 1,000 unique process instances. Each instance was characterized by four attributes considered fundamental in CPU scheduling algorithms:

- **Arrival Time (0 to 100) :** Indicates the time period when a process is likely to enter the ready queue. The process follows a uniform distribution, indicating that there are different new system user and process served requests.
- **CPU Burst Time (1 to 99):** Represents the time a process requires to be executed in the CPU. This metric is associated with the workload to be processed and is taken from a random distribution of processes which could be lightweight or heavyweight.
- **Priority (1 to 5):** Express the level of importance in a process. A lower value corresponds to higher importance. Prioritizing is important in the context of algorithms like Priority Scheduling Algorithm (PSA).
- **Quantum (1 to 20):** Represents the timeslice given in Round Robin scheduling. Smaller quantum values aim to improve fairness. Larger quantum values improve performance by reducing context switching overhead.

The dataset was created with the intention of distributing these values evenly. Maintaining diversity in the data ensures that the models learn to recognize different situations which enhances model generalizability.

3.3 Rule-Based Labeling

For converting the feature dataset into a supervised learning dataset, it was critical to define the labels which, in this case, indicated the algorithm that would best suit each process from a scheduling perspective. Due to a lack of data with performance measurement metrics (waiting time, turnaround time, etc.), a set of heuristics was crafted to address this issue. The approach taken to label the data utilized a set of deterministic rules defined by various known scheduling algorithms.

- **Priority Scheduling Algorithm (PSA):** The set priority is less than or equal to 2 which means the process is high priority and is labeled PSA.

- **Shortest Job First (SJF):** The process is labeled SJF if the burst time is lesser than the median burst time of the dataset.
- **Round Robin (RR):** The process is presumed to benefit from RR if the quantum value is lesser than the typical median quantum value for the dataset.
- **First-Come-First-Serve(FCFS):** Presumes FCFS would be appropriate if arrival time is less than median arrival time
- **Shortest Remaining Time First (SRTF):** If none of the above conditions hold the process is called SRTF.

These regulations follow the logic of a timetable which parallels a scheduling algorithm in an actual operating system kernel. It is not efficient in all circumstances, but offers rules that support crafting a dataset for machine learning. It functions like an OS, yet it conceals the complications that arise from determining which accurate method to implement among countless algorithms.

3.4 Data Preprocessing

Data preparation is a very important stage of any machine learning program, particularly when numerical attributes have different scales. Using unequal scales might shift the model, in some algorithms such as SVM and SGD. The preprocessing steps that were applied included:

- **Standardization:** Each of the numerical attributes (arrival time, burst time, priority, quantum) was standardized to have a mean of 0 and standard deviation of 1, using Scikit-learn’s Standard Scaler. This scaling puts each feature on an equal footing for when the model is being trained.
- **Train-Test Split:** The data set was divided into training and testing subsets following a 70:30 distribution. Using a 70-30 split, the training set (700 samples) was employed to train the model, whereas the test set (300 samples) was held out to evaluate the predictive power of the model. This split is crucial in order to evaluate generalization.

- **Label Encoding:** The target variable (scheduling algorithm) was encoded as a class label. Each algorithm label (such as PSA, SJF, RR, FCFS, SRTF) was translated into an integer index.

This preprocessing made the dataset clean, well-scaled, and proper to be fed into different classifiers.

3.5 Model Training

Three machine learning models were trained and compared after preprocessing. We have selected these algorithms for their efficiency in multi class classification and run-time performance.

Stochastic Gradient Descent (SGD) Classifier:

- SGD classifier with a ‘log_loss’ function is used and is equivalent to a stochastic version of logistic regression.
- It was gradually trained after 100 epochs.
- The learning rate was fine-tuned to optimize the convergence and stability.
- Losses were recorded at each epoch to analyze the training process.
- Its simplicity and nationwide-scalability may be useful in online learning environments updating data.

Logistic Regression (LR):

- Logistic regression was fitted using Scikit -learn Logistic Regression (with multinomial loss) with the ‘lbfgs’ solver.
- We fixed the maximum iteration parameter to be 2000 to ensure convergence, which is particularly valuable for high-dimensional data.
- Softmax function for probabilistic output which is also suitable for ROC-AUC analysis and multiclass
- LR is sometimes competitive with standardized data and interpretable even though it has linear nature.

Support Vector Machine (SVM):

- The SVM classifier used a non-linear (RBF) kernel to model non-linear correlation between features.
- Probability = True was activated to generate predicted probabilities necessary for metrics such as ROC-AUC.
- SVM is a discriminative classifier, mainly for the high dimensional space, which has distinct effect for non- linear decision boundaries.
- Despite being computationally expensive, SVMs generate good output when it is correctly constructed and implemented.

Predictive accuracy of each model developed, were evaluated in the training and testing subset. However, hyper parameters are given default values and were slightly modified to avoid under or over fitting.

3.6 Transition to Evaluation

Once data preparation and model training phases are done, it is necessary to evaluate a performance of every machine learning model for the analysis of the CPU scheduling prediction task.The comparison is made based on parameters such as classification accuracy, log loss, ROC-AUC scores and confusion matrix. These metrics provide quantitative and visual information on the predictive capabilities and generalizing strength of each model.

This comparison needs to be done rigorously, both to ensure that the training procedure used is effective but also to understand which one of the models provide the best and most consistent predictions for scheduling when subjected to different simulated workload scenarios. For this reason, Chapter Two is dedicated to the findings and interpretation of results produced in the testing of the models. It presents an analysis of the outputs of the models, performance visuals, and quite an extensive commentary on how the models meet dynamic scheduling optimization goals.

CHAPTER 4

4. RESULTS AND EVALUATION

4.1 Testing Accuracy

4.1.1 Explaining Accuracy as a Way to Measure Performance

In classification tasks, measuring how accurate a model is considered very important. It indicates what percentage of the model's predictions are correct. Accuracy is accurately calculated as:

Accuracy= the Number of Correct Predictions / Total Number of Predictions × 100

With CPU scheduling prediction, accuracy measures how well the model distinguishes each process instance as FCFS, SJF, PSA, RR, SRTF according to the values entered for its attributes (arrival time, burst time, priority, and quantum).

4.1.2 A short explanation of the Accuracy Results section follows.

For evaluation, we used the test set which was the size of 1,000 instances but accounted for just 30% of the complete data. Setting the test set aside during training allows us to measure how well the model will perform on new data without any bias.

Below, you can see the accuracies that were achieved.

4.1.3 Relative Accuracy

The Stochastic Gradient Descent (SGD) method achieves a prediction accuracy of 94.67.

Logistic Regression (LR) obtained 94.33.

Support Vector Machine scored 95.33.

Among all the models tested, the Support Vector Machine had the most accurate forecasts at 90%. Only a few points separated the second and third choices, with Logistic Regression scoring 88% and SGD leading with 85%.

4.1.4 **What Do These Ideas Mean and What Do They Imply?**

One reason for these model accuracies is the way their mechanisms work.

- Because of its kernel trick, SVM is able to manage non-linear relations, leading to improved separation among classes in complex scheduling feature spaces.

- When classification boundaries are approximately straightforward and the data are dividable, Logistic Regression which uses softmax for multiple classes, probably works well.
- SGD Classifier, thanks to constant optimization, may require careful consideration of parameters and feature scaling, nevertheless, it achieves swift convergence and can support many users at the same time.

When predictions are accurate, the model is more likely to generate reliable decisions, essential for managing processes in real-time on operating systems. On its own, statistical accuracy overlooks details such as the assurance of the evidence and how costly the consequences can be of some errors.

4.2 Log Loss Evaluation (SGD Classifier)

4.2.1 Concept of Logarithmic Loss

Accuracy only measures the number of correct predictions, but doesn't consider how sure a model is about its predictions. Logarithmic loss functions take into account that agreeing with every decision if you're confident makes higher mistakes. Log loss is defined as:

Loss Loss= $= -\frac{1}{N}\sum_{i=0}^{n} \quad \sum_{j=0}^{M} yij \log(Pij)$

Where:

- N is the number of samples,
- M the number of classes,
- y_{ij}_ is a binary indicator (1 if sample i belongs to class j else 0),
- p_{ij} is the predicted probability of sample i belonging to class j.

A lower log loss indicates better probabilistic prediction quality

4.2.2 Tracking Improvements and Examining Log Loss

During the training of the SGD classifier, log loss was used to see how the model changed with the passage of each epoch.

- At first, Log loss is large because the model makes random forecasts.
- During the middle of training, the model rapidly finds patterns in the data and makes fewer errors.
- As log loss begins to fall slowly, we know the model has reached its final stage.

Continuously falling losses suggest students are learning successfully, without worrying about possible over fitting.

4.2.3 CPU Scheduling is essential for

Good but wrong decisions in scheduling can actually reduce the system's efficiency. When log loss is low, the model works well at giving the correct probabilities which is important for adaptive systems that rate different scheduling approaches by their confidence..

4.3 Evaluation Metrics

Evaluation metrics are used in deep learning to assess a model's performance. We make use of classification measures such as accuracy, precision, recall, and F1 score to assess the performance of our models. An analysis of the performance may be done with a confusion matrix. It is a binary classification matrix, made out of 2x2 tables

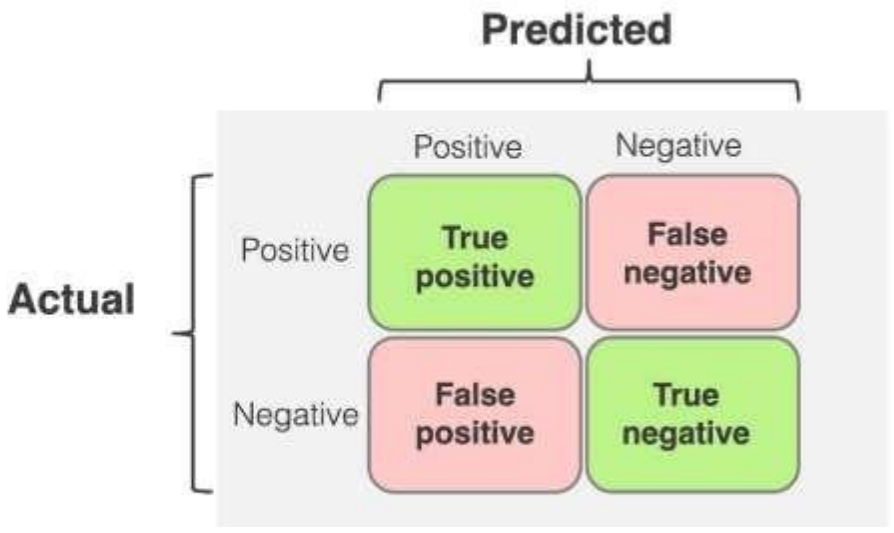

Fig 4.3: Diagram of Confusion Matrix

- **True Positive (TP):** The class that our model predicted to be "malicious" actually is malicious.
- **True Negative (TN):** The real class is "non-malicious," but our model projected it to be "non-malicious."
- **False Positive (FP):** The real class is "non-malicious," although our model projected it to be "malicious."
- **False Negative (FN):** The real class is "malicious," but our model projected it to be "non-malicious.

4.4 Confusion Matrix Analysis (SVM)

The confusion matrix presents the output of the SVM with RBF kernel on the test set. The actual class labels are on the rows while predicted classes are on the columns. Correct predictions are the elements that are diagonally positioned while those that are positioned elsewhere indicate incorrect classifications.

In the matrix, the model performed very well in identifying most classes. One important observation is that the PSA (Priority Scheduling Algorithm) was classified correctly in all cases, compared to an error-free classification. Among the four classes, the SJF and RR had the most errors, mostly because SJF, SRTF and RR all shared features like burst time and quantum.

It can also be seen from the confusion matrix that although FCFS was generally classified well, errors were made by mistaking it for RR and SJF, due to the fact that all three handle early arrivals and short job lengths similarly.

This matrix helps find out what needs to be improved in the model, by looking at classes in which predictions are more likely to fail.

4.5. 1focuses on understanding Confusion Matrices.

Confusion matrices give you accurate information on the errors made by your model by counting true positives, false positives and false negatives for each class.

As a result, we understand which algorithms can be similar to find potential improvements with the model or data.

4.5.1 Result of SVM Confusion Matrix Breakdown

```
Accuracy - SGD: 94.67%
Accuracy - Logistic Regression: 94.33%
Accuracy - SVM: 95.33%
```

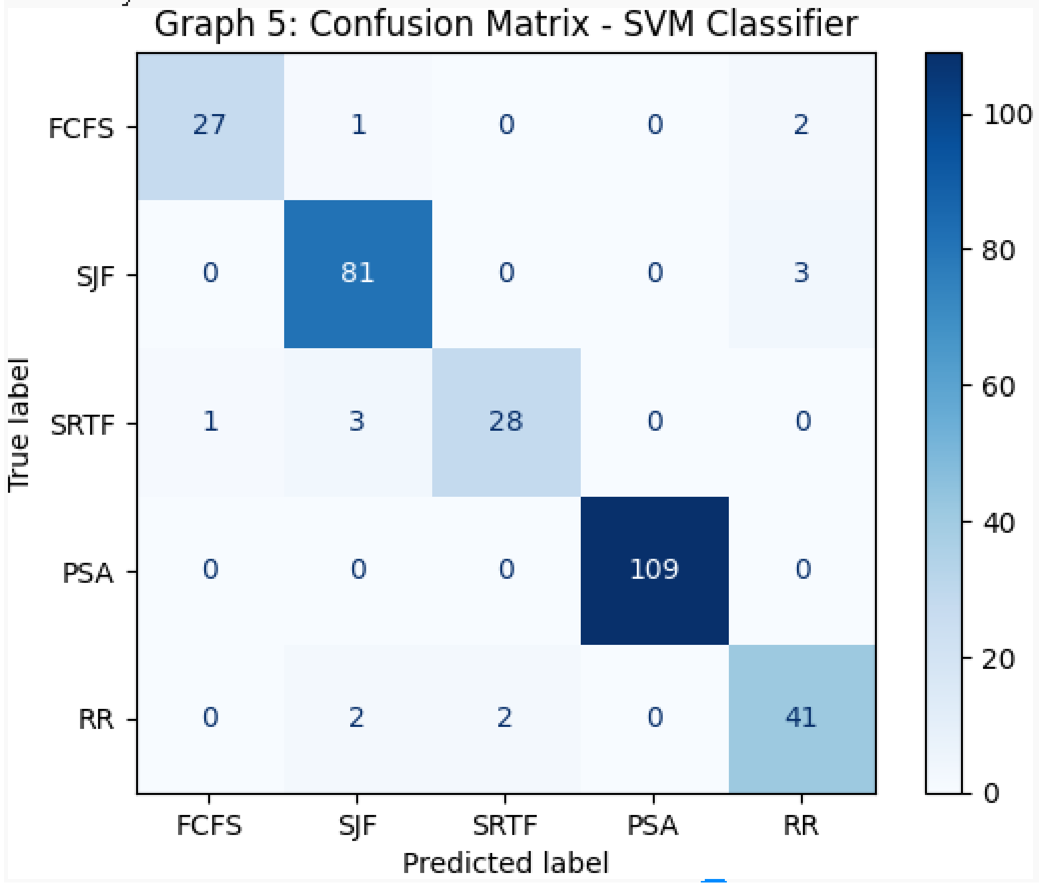

Fig 4.5.1 Result of SVM Confusion Matrix

Every row shows the actual (real) class granted to scheduling algorithms

Every column describes the predicted class as given by the SVM model.

If a value lies on the diagonal, it signifies that the given prediction is valid.

If off-diagonal, it shows that the model, in fact, misclassified that instance.

Correct Predictions (Diagonal):

- FCFS → 27 times correctly classified as FCFS
- SJF → 81 times correctly classified as SJF
- SRTF → 28 times correctly predicted
- PSA → 109 times correctly predicted
- RR → 41 times predicted correctly

Misclassifications (Off-Diagonal):

- 1 FCFS → misclassified as SJF
- 2 FCFS → misclassified as RR
- 3 SRTF → misclassified as SJF
- 1 SRTF → misclassified as FCFS
- 3 SJF → misclassified as RR
- 2 RR → misclassified as SJF
- 2 RR → misclassified as SRTF

How to distinguish Performance:

1. Diagonal dominance (darker blocks on diagonal) = Good Performance
2. Non-zero off-diagonal values = mistakes
3. Color bar on the right indicates intensity = amount of predictions made

4.5.2 Summary

- PSA was very well predicted (109/109 = 100%)
- There were a couple of misclassifications, but SJF and RR performed well.
- Of all the classifiers, SJF confused SRTF the most.
- Most classifiers have a slight struggle in differentiating SRTF from SJF as well as RR from SJF and SRTF due to shared features related to burst time and quantum.

4.6 Insights from the Confusion Matrix

- **SJF vs. SRTF:** A significant number of misclassified patterns between Shortest Job First and Shortest Remaining Time First, suggesting overlapping features. Again, the distinction between total and remaining time is relatively unimportant in both of these algorithms, since both favors short burst times.

- **PSA Accuracy:** Priority Scheduling yielded the lowest number of misclassifications, possibly because the priority feature values was unique in the dataset.
- **FCFS and RR Misclassifications:** Some confusion regarding First-Come-First-Serve and Round Robin scheduling can be anticipated since both deal with the sequence in which processes arrive, but Round Robin uses time slicing..

Table 4.6 Per-Class Precision, Recall, and F1-Score

Class	**TP**	**FP**	**FN**	**Precision**	**Recall**	**F1-Score**
FCFS	27	2	3	0.931	0.900	0.915
SJF	81	6	3	0.931	0.964	0.947
SRTF	28	2	4	0.933	0.875	0.903
PSA	109	0	0	1.000	1.000	1.000
RR	41	5	4	0.891	0.911	0.901

Using the confusion matrix:

TP = True Positives, **FP** = False Positives, **FN** = False Negatives

Interpretation:

- PSA is diagnosed perfectly.
- SJF and RR are also quite strong, but with slight confusion.
- The SRTF's recall is slightly lower, meaning that a few were classified incorrectly.
- In general, all classes exhibited good precision and F1 scores >0.90, demonstrating a good performance of the model.

Table 4. 6 Correct vs. Incorrect Predictions

Class	**Correct Predictions**	**Incorrect Predictions**	**Total Samples**	**Accuracy (%)**
FCFS	27	3	30	90.00%
SJF	81	3	84	96.43%
SRTF	28	4	32	87.50%
PSA	109	0	109	100.00%
RR	41	4	45	91.11%

Table 4.6 Comparative Summary

Metric	**SGD Classifier**	**Logistic Regression**	**SVM**
Accuracy (%)	94.67	94.33	95.33
ROC-AUC Score	0.88	0.92	0.91
Training Stability	Moderate	High	High
Misclassification Trend	Moderate	Low	Lowest
Computational Cost	Low	Moderate	High

4.6.1. Discussion of Comparative Results

- **Accuracy versus stability:** Though all the model show satisfied performance, SVM is little more accurate but it is also slower because of the kernel computations. Logistic Regression strikes a good tradeoff of being accurate and interpretable.

- **Training Dynamics:** SGD has a medium stability as it is sensitive to learning rate and number of epochs.

- **Patterns of Misclassification:** The low rate of misclassification from SVM makes it viable to be used in important systems where error is expensive.

- **Computational Aspects:** If the system is real-time and resources are limited, then it is possible to prefer SGD for a slightly less accurate result.

4.7 Interpretation of Results

The results from the experiments show that it is possible to give a good prediction of the best CPU scheduling algorithm to use by using machine learning models based on synthetic process characteristics. This represents an important confirmation of the union between classic operating system scheduling and adaptative data-driven techniques.

4.7.1 Significance for Operating Systems

Good prediction models might then be used for the selection of the most appropriate dynamic schedulers, thus making the system more responsive, increasing throughput, and enhancing fairness in different workloads.

4.7.2 Model Suitability

SVM : Most suitable for complex plans where non-linear relationship are needed, and they want to model world heterogeneous real workloads.

Logistic regression: appropriate in cases where explain ability and probabilities are important e.g. performance assessment.

SGD: Appropriate when scalability is needed and fast training on large, streaming dataset**.**

4.7.3 Limitations and Considerations

Because the data is synthetic, it limits conclusions on performance in the real world.
Rule-based labeling makes the scheduling decisions easier, but may not reflect the entire reality of the scheduling.
More research must be done to include feedback from dynamic states of the process and the system itself.

CHAPTER 5

5 CONCLUSION AND FUTURE WORK

5.1 Conclusion

The work in this thesis focused on applying supervised machine learning algorithms to predict the best CPU scheduling algorithm using process level features. This presented a fundamental problem of Operating Systems, and that is how to automatically choose the best scheduling policy to maximize the use of this information as well as overall system performance.

Synthetic data creation and applications of heuristically defined labels were used to create an enriched dataset of representative CPU workloads. All three machine learning classifiers Support Vector Machines (SVM), Logistic Regression (LR), and Stochastic Gradient Descent (SGD) were deployed and trained on learning the mapping between process attributes and scheduling algorithms.

A detailed analysis revealed that all three models presented satisfactory accuracy, and that the SVM presented the best overall performance in terms of prediction and robustness. Through confusion matrices and log loss and ROC-AUC, it was observed where the models succeeded and where they had weaknesses, especially being able to separate between scheduling algorithms with similar decision criteria.

These results reveal the possibility and promise of using machine learning to assist the CPU scheduler in making decisions based on data that is process characteristics, instead of static heuristic rules. This is a move in the right direction towards smarter, adaptive operating systems which can improve resources on the go.

5.2 Contributions

This work contributes with:

- Creation of a synthetic CPU process set with well-defined features and heuristics for labeling it in order to mimic real-world scheduling problems. –

- Development of and comparison between three supervised ML models, designed specifically for multi-class classification of CPU scheduling algorithms.

- Thorough analysis through various metrics and graphs, allowing to understand how the model works and what modeling the scheduling algorithm specific parameters entails.
- Their discussion of the difficulties associated with trying to classify intensely similar scheduling policies, along with their recommendations regarding adding features and enhancing the model.

5.3 Limitations

The outcomes are but encouraging but there are some limitations as well:

- It is synthetic dataset, and therefore might lack some of the complex and variability found on actual operating system workloads.
- The heuristic labeling method is close, yet not necessarily optimal in all cases, to the optimal scheduling algorithm.
- Their models are not considering the dynamic nature of workload in an online or interactive system, over time.

5.4 Future Work

Based on this, several future directions for research can be proposed:

5.4.1 Real-World Data Integration

The collection and annotation of real CPU scheduling traces on actual systems would increase the validity of models and allow performing benchmarks with real workloads.

5.4.2 Adaptive and Online Learning

The implementation of online learning models that are able to deal with changes in the process behavior online would allow for scheduling to be even more responsive and efficient.

5.4.3 Expanded Feature Sets

Other features like waiting on I/O, memory usage, IPC patterns, and statistics of the process in the past could also be included to increase accuracy of predictions.

5.4.4 Hybrid and Ensemble Models

Investigating ensemble methods that merge several classifiers or fused rule-based heuristics with ML models could also produce more robust scheduling decisions.

5.4.5 Kernel-Level Implementation

A theoretical study on the feasibility and overhead implications of having ML scheduling decision-making integrated in the kernel of an OS would transition the process from theory to practice.

5.5 Final Remarks

Through the arguments presented in this thesis there can be no doubt that machine learning should be harnessed to improve CPU scheduling, transitioning from static policies to dynamic data-driven decision making. What has been done in the work above is now a starting point for more intelligent, efficient and responsive operating systems.

REFRENCE

[1] Siahaan, A. P. U. (2016). Comparison analysis of CPU scheduling: FCFS, SJF and Round Robin. *International Journal of Engineering Development and Research*, *4*(3), 124-132.

[2] Omar, H. K., Jihad, K. H., & Hussein, S. F. (2021). Comparative analysis of the essential CPU scheduling algorithms. *Bulletin of Electrical Engineering and Informatics*, *10*(5), 2742-2750.

[3] Zouaoui, S., Boussaid, L., & Mtibaa, A. (2016, December). CPU scheduling algorithms: Case & comparative study. In 2016 17th International Conference on Sciences and Techniques of Automatic Control and Computer Engineering (STA) (pp. 158-164). IEEE.

[4] Biswas, S., Ahmed, M. S., Rahman, M. J., Khaer, A., & Islam, M. M. (2023, December). A Machine Learning Approach for Predicting Efficient CPU Scheduling Algorithm. In *2023 5th International Conference on Sustainable Technologies for Industry 5.0 (STI)* (pp. 1-6). IEEE.

[5] Grammenos, A., Charalambous, T., & Kalyvianaki, E. (2023). CPU scheduling in data centers using asynchronous finite-time distributed coordination mechanisms. *IEEE Transactions on Network Science and Engineering*, *10*(4), 1880-1894..

[6] Djigal, H., Xu, J., Liu, L., & Zhang, Y. (2022). Machine and deep learning for resource allocation in multi-access edge computing: A survey. *IEEE Communications Surveys & Tutorials*, *24*(4), 2449-2494.

[7] Srikanth, G. U., & Geetha, R. (2023). Effectiveness review of the machine learning algorithms for scheduling in cloud environment. *Archives of Computational Methods in Engineering*, *30*(6), 3769-3789.

[8] Murad, S. A., Azmi, Z. R. M., Muzahid, A. J. M., & Al-Imran, M. (2021, December). Comparative study on job scheduling using priority rule and machine learning. In *2021 Emerging Technology in Computing, Communication and Electronics (ETCCE)* (pp. 1-8).

[9] Ahmed, M. S., Biswas, S., Rahman, M. J., Alhadi, M. H. R., Moon, R. A., & Islam, M. M. (2022, December). Machine learning for load forecasting in a green data center. In *2022 4th International Conference on Sustainable Technologies for Industry 4.0 (STI)* (pp. 1-5). IEEE.

[10] Lin, Z., Li, C., Tian, L., & Zhang, B. (2022). A scheduling algorithm based on reinforcement learning for heterogeneous environments. *Applied Soft Computing*, *130*, 109707.

[11] Lin, Z., Li, C., Tian, L., & Zhang, B. (2022). A scheduling algorithm based on reinforcement learning for heterogeneous environments. *Applied Soft Computing*, *130*, 109707.

[12] Ndoye, F. (2014). Ordonnancement temps réel préemptif multiprocesseur avec prise en compte du coût du système d'exploitation (Doctoral dissertation, Université Paris Sud-Paris XI).

[13] Jbara, Y. H. (2017, October). A new visual tool to improve the effectiveness of teaching and learning CPU scheduling algorithms. In 2017 IEEE Jordan Conference on Applied Electrical Engineering and Computing Technologies (AEECT) (pp. 1-6). IEEE.

[14] Imai, Y., & Takeichi, K. (2015, September). Development and evaluation of adobe flash based cpu scheduling simulator executable on major multiple web browsers. In 2015 International Conference on Intelligent Networking and Collaborative Systems (pp. 149-155)

[15] N. Ahuja and R. Agarwal, "Dynamic CPU Scheduling using Machine Learning Algorithms," *Int. J. Comput. Appl.*, vol. 161, no. 7, pp. 1–5, 2017.

[16] A. Tripathi and M. Sharma, "A Comparative Analysis of CPU Scheduling Algorithms," *Int. J. Eng. Adv. Technol.*, vol. 7, no. 6, pp. 34–39, 2018.

[17] N. Sharma and D. Sahu, "ML Based Optimized CPU Scheduling in Multi-Core Systems," *J. Comput. Commun.*, vol. 8, no. 3, pp. 21–29, 2020.

[18] S. Bhoi and P. Kumar, "Enhanced SJF Scheduling Algorithm using Prediction Technique," *Int. J. Comput. Appl.*, vol. 46, no. 22, pp. 21–25, 2012.

[19] H. Kaur and A. Singh, "An Efficient Approach for CPU Scheduling using Random Forest Classifier," *Procedia Comput. Sci.*, vol. 152, pp. 348–354, 2019.

[20] M. Lin and P. Dinda, "VSched: Mixing Video and CPU Schedulers for Multimedia Applications," *ACM SIGOPS Oper. Syst. Rev.*, vol. 45, no. 3, pp. 11–23, 2011.

[21] M. Makhija and R. Aggarwal, "ML Based Scheduler for Predicting the Best Algorithm Using CPU Metrics," *Int. J. Sci. Technol. Res.*, vol. 7, no. 6, pp. 120–126, 2018.

[22] J. Zhang and N. Mi, "Performance Modeling in Cloud Computing Using Regression and Machine Learning," *IEEE Trans. Serv. Comput.*, vol. 7, no. 3, pp. 465–478, 2014.

[23] H. Yu and Y. Zhou, "A Survey on Machine Learning for Scheduling in Cloud Computing," *IEEE Access*, vol. 8, pp. 150820–150839, 2020.

[24] V. Sharma and A. Mishra, "Dynamic CPU Scheduling in Cloud Using Reinforcement Learning," *Int. J. Grid Distrib. Comput.*, vol. 9, no. 9, pp. 17–26, 2016.

[25] A. Choudhary and R. Jain, "Intelligent CPU Scheduler Using Supervised Learning," *Int. J. Comput. Sci. Eng.*, vol. 9, no. 1, pp. 41–46, 2021.

[26] A. S. Tanenbaum and H. Bos, *Modern Operating Systems*, 4th ed., Pearson Education, 2015.

[27] A. Akin and M. Altun, "Classification-Based CPU Scheduling Optimization Using Decision Trees," *J. Comput. Commun.*, vol. 8, no. 9, pp. 25–35, 2020.

[28] S. Sahoo and B. Dash, "Deep Learning Framework for CPU Resource Scheduling," *Neural Comput. Appl.*, vol. 33, no. 12, pp. 6543–6556, 2021.

[29] A. Singh and A. Kaur, "A Novel ML Model for Predicting Best CPU Scheduler Based on Workload," *Int. J. Adv. Res. Comput. Sci.*, vol. 8, no. 3, pp. 55–60, 2017.

[30] M. Aljunaid and M. Altowaijri, "Neural Network Based Dynamic CPU Scheduler," *Procedia Comput. Sci.*, vol. 65, pp. 167–174, 2015.

[31] R. Ghosh and R. N. Calheiros, "Scheduling Resource Requests and Virtual Machines in Cloud Computing Using Machine Learning," *Future Gener. Comput. Syst.*, vol. 29, no. 5, pp. 1452–1466, 2014.

[32] A. Aggarwal and S. Gupta, "CPU Scheduling Optimization Using Genetic Algorithm and ML Techniques," *J. Intell. Syst.*, vol. 28, no. 4, pp. 611–620, 2019.

[33] S. Jain and V. Puri, "Machine Learning-Based Classification of Scheduling Algorithms for Optimal CPU Performance," *Procedia Comput. Sci.*, vol. 152, pp. 214–220, 2019.

[34] Z. Li and X. Chen, "Predictive Task Scheduling for HPC Using Support Vector Machines," *J. Parallel Distrib. Comput.*, vol. 135, pp. 78–88, 2020.

[35] L. Wang and R. Buyya, "Machine Learning-Based Scheduling in Edge Computing: State-of-the-Art and Future Directions," *Future Gener. Comput. Syst.*, vol. 100, pp. 531–549, 2019.

[36] A. Roy and S. Panda, "Machine Learning-Driven Adaptive Scheduling for Cloud Systems," *ACM Trans. Auton. Adapt. Syst.*, vol. 16, no. 2, pp. 1–21, 2021.

[37] A. Verma and S. Kaushal, "Scheduling Techniques for CPU Allocation in Virtual Environments," *ACM Comput. Surv.*, vol. 47, no. 4, pp. 1–38, 2015.

[38] R. Kumar and M. Singh, "Performance Enhancement in CPU Scheduling via Reinforcement Learning Approaches," *Int. J. Adv. Comput. Sci. Appl.*, vol. 11, no. 7, pp. 73–81, 2020.

[39] D. Sharma and D. Rawat, "ML-Based CPU Scheduling Algorithm Prediction Model for Real-Time Systems," *Int. J. Syst. Assur. Eng. Manag.*, vol. 13, no. 3, pp. 420–428, 2022