ESTRA: AN EASY STREAMING DATA ANALYSIS TOOL

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

ECEHAN SAVAŞ BAŞAK

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

FEBRUARY 2021

Approval of the thesis:

ESTRA: AN EASY STREAMING DATA ANALYSIS TOOL

submitted by **ECEHAN SAVAŞ BAŞAK** in partial fulfillment of the requirements for the degree of **Master of Science in Computer Engineering Department, Middle East Technical University** by,

Prof. Dr. Halil Kalıpçılar
Dean, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. Halit Oğuztüzün
Head of Department, **Computer Engineering** _____

Prof. Dr. Mehmet Volkan Atalay
Supervisor, **Computer Engineering, METU** _____

Examining Committee Members:

Prof. Dr. Ferda Nur Alpaslan
Computer Engineering, METU _____

Prof. Dr. Mehmet Volkan Atalay
Computer Engineering, METU _____

Prof. Dr. Fazlı Can
Computer Engineering, Bilkent University _____

Date:

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Surname: Ecehan Savaş Başak

Signature :

ABSTRACT

ESTRA: AN EASY STREAMING DATA ANALYSIS TOOL

Savaş Başak, Ecehan
M.S., Department of Computer Engineering
Supervisor: Prof. Dr. Mehmet Volkan Atalay

February 2021, 63 pages

Easy Streaming Data Analysis Tool (ESTRA) is designed with the aim of creating an easy-to-use data stream analysis platform that serves the purpose of a quick and efficient tool to explore and prototype machine learning solutions on various datasets. ESTRA is developed as a web-based, scalable, extensible, and open-source data analysis tool with a user-friendly and easy to use user interface. ESTRA comes with a bundle of datasets (Electricity, KDD Cup'99, and Covertype), dataset generators (Sea and Hyperplane), and implementations of various analysis and learning algorithms (D3, Hoeffding Tree, CluStream, DenStream, kNN, k-means, and StreamKM++). Moreover, ESTRA provides an easy way to investigate various properties of the datasets and to observe the results of executed machine learning algorithms. ESTRA's straightforward and clean architecture with open source tools allows it to be extensible. Used libraries and frameworks in ESTRA like React, Python and Scikit-Multiflow are popular open source tools with broad community support and extensions. ESTRA's capabilities of easy prototyping and exploring machine learning solutions are demonstrated by repeating the machine learning experiments performed in various studies.

Keywords: Data Streaming, Data Analysis, Real-Time Data Analysis, Data Stream Analysis, Data Stream Analysis Tool

ÖZ

ESTRA: KOLAY KULLANIMLI AKAN VERİ ANALİZ ARACI

Savaş Başak, Ecehan
Yüksek Lisans, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi: Prof. Dr. Mehmet Volkan Atalay

Şubat 2021 , 63 sayfa

Kolay Kullanımlı Akan Veri Analiz Aracı (ESTRA) çeşitli veri kümelerinde makine öğrenimi çözümlerini araştırmak ve prototip oluşturmak için, hızlı ve verimli bir şekilde hizmet eden kullanımı kolay bir veri akışı analiz aracı olarak tasarlanmıştır. ESTRA, kullanıcı dostu ve kolay kullanılabilir kullanıcı arayüzü ile; web-tabanlı, ölçeklenebilir, genişletilebilir ve açık kaynak kodlu bir veri analiz aracı olarak geliştirilmiştir. ESTRA, çeşitli veri kümeleri (Electricity, KDD Cup'99, ve Covertype), veri üreticileri (Sea ve Hyperplane) ve çeşitli analiz ve öğrenme algoritmalarının (D3, Hoeffding Tree, CluStream, DenStream, kNN, k-means, ve StreamKM++) gerçeklenmesini içerir. Bununla birlikte, ESTRA veri kümelerinin farklı özelliklerini incelemeye ve çalıştırılan makine öğrenmesi algoritmalarının sonuçlarının gözlemlemeye imkan sağlar. ESTRA'nın kullandığı açık kaynak kodlu araçlar ile basit ve anlaşılır mimarisi, ESTRA'nın genişletilebilir olmasını destekler. ESTRA'da kullanılan React, Python ve Scikit-Multiflow gibi kütüphaneler ve yazılım çatıları, geniş topluluklar tarafından geliştirilen ve desteklenen yaygın açık kaynak kodlu araçlardır. ESTRA'nın kolay prototip oluşturma ve makine öğrenimi çözümlerini deneyimleme yetenekleri, çeşitli çalışmalarda gerçekleştirilen makine öğrenimi araştırmalarının tekrarlanma-

sıyla gösterilmiştir.

Anahtar Kelimeler: Akan Veri, Veri Analizi, Gerçek Zamanlı Veri Analizi, Akan Veri Analizi, Akan Veri Analiz Aracı

To my lovely family...

ACKNOWLEDGMENTS

I would first like to thank and express my gratitude to my thesis advisor Prof. Dr. Mehmet Volkan Atalay for his supervision and guidance during the research.

I also would like to thank to rest of my thesis committee, Prof. Dr. Ferda Nur Alpaslan and Prof. Dr. Fazlı Can for evaluating this thesis.

I would like to thank my employer ASELSAN A.Ş. for giving me the opportunity of continuing my education and supporting me.

Besides, I am grateful to Alaettin Zubaroğlu for his contributions.

I want to thank to my supportive friends and all people who have helped me during my thesis study.

Finally, I have no suitable words to express my deepest gratitude to my family for their support in every aspect of my life. I am indebted to my mother Songül Savaş and my father Reha Savaş for their care and everlasting love to me. I am grateful to my sister Bihan Çıttır and my lovely nephew and nieces for enjoying and supporting me. I am thankful to my husband Ahmet Eren Başak with all my heart for his unconditional love, trust, encouragement, guidance, and understanding throughout my life, I could not accomplish my goals without their love.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

LIST OF ABBREVIATIONS

ABBREVIATIONS

AMQP	Advanced Message Queuing Protocol
AUC	Area under the ROC Curve
CPU	Central Processing Unit
D3	Discriminative Drift Detector
ESTRA	Easy Stream Analysis Tool
FN	False Negative
FP	False Positive
HTTP	Hypertext Transfer Protocol
JMS	Java Message Service
kNN	k-Nearest Neighbor
MOA	Massive Online Analysis
PAW	Probabilistic Adaptive Window
RAM	Random Access Memory
SAMOA	Scalable Advanced Massive Online Analysis
SPA	Single Page Application
TN	True Negative
TP	True Positive
VFDT	Very Fast Decision Tree
WEKA	The Waikato Environment for Knowledge Analysis

CHAPTER 1

INTRODUCTION

In recent years, newer technologies and advancements in the mobile devices and sensors increased the connectivity of devices to the internet and enabled more data to be generated via social media, mobile devices, sensors, and internet of things. With this growing size and increasing speed of the data, streaming data analysis has attracted more attention. Extracting valuable information from such a growing amount of data is a challenge by itself and when it's crucial to be able to turn the data into meaningful information in real-time, the ability to process the data as it arrives is an essential ability [6]. There have been many studies on streaming data analysis and various algorithms and tools have been developed for this purpose. Nowadays, it's a challenge to navigate in the streaming data analysis ecosystem to find the correct algorithm for the problem at hand and tune it to obtain optimum results.

One initial approach to real time analysis of the streaming data was to perform traditional methods on the batches of the arriving data [7]. Batch data is defined to be the block of data collected within a certain period, and batch data processing is processing the collected and stored block of data. There are disadvantages of batch data processing. For example, before processing, data should be collected, and the data collection causes a delay. Also, the storage of data is another problem, when data is continually growing [7].

Streaming data is defined as the type of data that is continuously flowing from one or multiple data sources. It is ordered by the arrival time or timestamps and it needs to be processed in a short time after it's produced [8]. Real-time processing is defined as processing incoming data with very low latency [9]. Data stream analysis is processing the data as soon as it's received from various sources with real-time or near

real-time analytic results [7].

In general, streaming data analysis methods are preferred over batch data analysis methods. The main reason is the need to obtain results in real-time or near real-time. Moreover, batch processing may not always be suitable for streaming data analysis [7].

Nowadays, there exist a number of tools, libraries, frameworks and algorithms to work with streaming data. This plenitude enables quick solutions to many of the existing problems, however, this also requires high investment for searching for the right approach for a given problem. Most of the time, a researcher needs to learn, install, configure and tune numerous existing tools and libraries while trying to find a solution for a streaming data analysis problem. In this thesis, in order to address this problem, a web-based, user-friendly, scalable, extensible and open source data stream analysis tool called Easy Stream Analysis Tool (ESTRA) is described to analyze streaming data with very low latency.

1.1 Motivation and Problem Definition

A stream of data may be generated by various sources such as sensors, the internet of things, and mobile devices; therefore data stream analysis is used in many domains such as security, energy, and communication [10, 11]. This variety resulted to have many studies on this area and several tools and frameworks have been developed for general purpose usage as well as for domain-specific problems. Depending on the targeted problem and environment, these tools and platforms have various trade-offs. Some of these tools, such as Massive Online Analysis (MOA) [12], work locally while the others, such as The Waikato Environment for Knowledge Analysis (WEKA) [13], work on a remote server; some of these platforms such as R [14], are extendable and others, such as WEKA, are not. Even though there are various tools with different targets, there is an absence of a tool that aims to be web-based, easy-to use and allows quick prototyping.

1.2 Proposed Methods and Models

In this study, we focused on constructing a web-based, scalable, extensible, open source, and user-friendly stream analysis tool which can be used with minimal prior knowledge. The tool is designed to do the heavy lifting on a server environment not to be limited by users' personal device capabilities and capacities. When we were designing our platform, we paid attention to the Scalability, Load Balancing, Extensibility (integration) and, Accuracy as key points to the analysis of the streaming data.

1.3 Improvements and Novelties

The improvements brought by this study are as follows:

- A survey about the existing data streaming approaches and streaming data analysis tools is presented with their categorizations and comparisons by various dimensions.
- A distributed, scalable, and extensible architecture for a general purpose data stream analysis tool that can serve personal needs as well as a large number of users and can be improved by users to support more use cases with more algorithm implementations and more datasets, is proposed with the roles and scalability characteristics of the components of the architecture explained.
- ESTRA; a web-based, user-friendly, scalable, extensible, and open-source data stream analysis tool is introduced that supports different data stream analysis algorithms, datasets, dataset generators, and visualized analysis results with charts and tables.
- Guidance on the usage of ESTRA with screenshots and detailed explanations on the user experience is provided.
- Example uses cases of ESTRA are presented by repeating the experiments performed in various studies in the literature and by comparing the results obtained by ESTRA with the original results achieved in the studies, thus presenting

ESTRA's capabilities as a quick prototyping platform for data stream analysis problems.

1.4 Outline of the Thesis

The rest of this thesis is organized as follows. In Chapter 2, the related work in data streaming platforms, data stream analysis tools are examined, and brief information about their features are given. In Chapter 3, the tool that we developed, ESTRA, is explained; its architecture, the platforms and technologies used in its development, and the algorithms which are added to the final product are detailed. In Chapter 4, the usage of ESTRA is demonstrated. Chapter 5 is devoted to algorithms, datasets, and the parameters used in this work. In Chapter 6, the obtained results from ESTRA and similar platforms are compared and discussed. Finally, in Chapter 7, our work is summarized, conclusions and limitations are presented, and some ideas for future work are described.

CHAPTER 2

DATA STREAMING AND STREAMING DATA ANALYSIS APPROACHES

Several algorithms, tools, and methods have been developed to analyze streaming data. Figure 2.1 demonstrates the relations between data streaming tools, data processing tools and machine learning tools and libraries along with some of the available tools. First, the data coming from different data sources are collected and streamed via data streaming tools. Then, data processing tools and machine learning tools and libraries process the data, and report the analysis results.

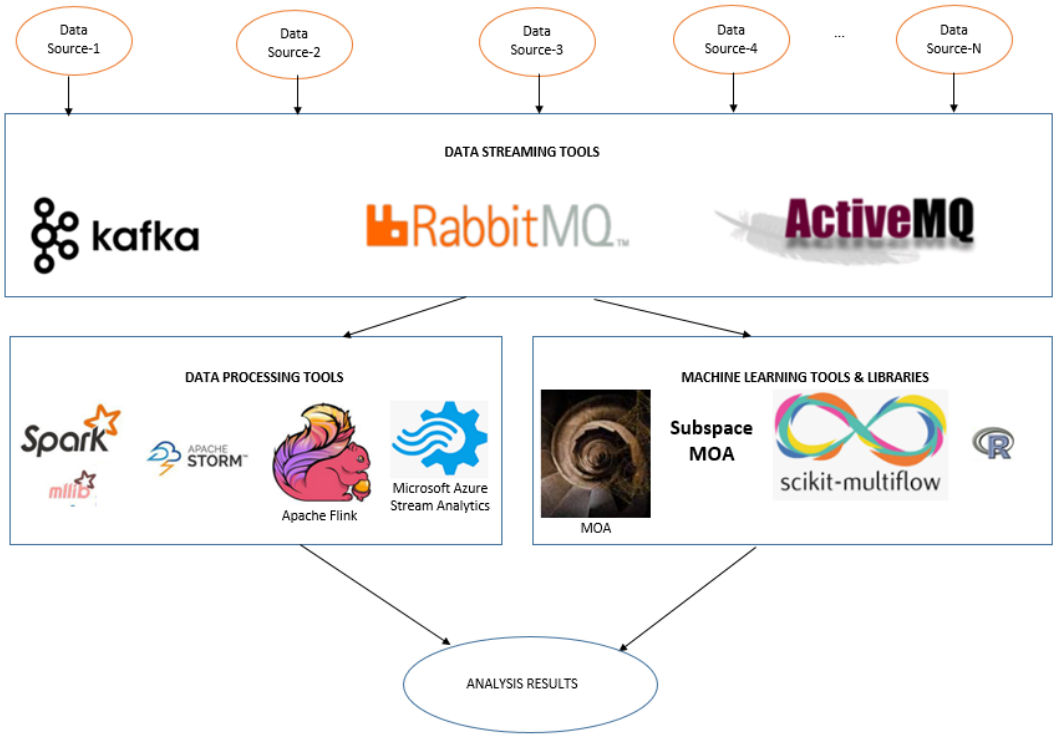


Figure 2.1: Relations between data streaming tools, data processing tools and machine learning tools and libraries.

Data streaming tools, data processing tools and machine learning tools and libraries need to address the following specifications [7].

- Scalability is the ability for a system to adapt to the growth of the load such as data size, number of dimensions, or the frequency of requests. This is one of the major problems when it comes to analyzing the growing data, because it's common that the increase rate of the data is greater than the growth rate of computing resources. For this reason, scalability is the key point the solve this need-source problem [7].
- Processing Speed is a metric to understand for how long does it take to analyze the data.
- Consistency is about obtaining the same or nearly the same analysis results with the same data, same algorithms, and the same parameters.
- Load Balancing is the ability to distribute the work evenly among the workers or computing units to maximize the output and increase efficiency in a distributed environment.
- Integration is generally valid for distributed systems, meaning that components of the system can work on a smaller subset of the problem and overall the outputs of the smaller components can be integrated back together in order to form the overall solution.
- Accuracy represents the closeness of the obtained results with the assumed truth [1]. It can be scoped to different metrics such as data accuracy, prediction accuracy, analysis result accuracy, etc.
- Fault Tolerance represents the ability of the system to continue operation despite the failures without interruption [7].

Understanding the use cases and features of the commonly used data analysis tools is essential to understand the infrastructure and the motivation of designing ESTRA. As the main goal of ESTRA to provide a easy to use way to experiment and learn machine learning especially on streaming data, ESTRA contains a simple data streaming tool

in itself while also utilizing many of the existing tools and libraries used for data analysis.

In this chapter, well known approaches to management, analysis, and learning of streaming data are reviewed. These approaches are investigated under the topics of concepts in these areas, tools for data streaming, tools for processing data streams, and tools and libraries for machine learning on streaming data.

2.1 Concepts

This section explains a few concepts regarding data streaming and streaming data analysis tools including Publish-Subscribe Systems and Micro Bathing Methodology.

Publish-Subscribe is one of the most commonly used messaging approach for distributed systems where multiple parties are involved in the generation and usage of the data. In this method, data is transmitted in the forms of messages with specific topics in the system. There are publishers that generate data messages with topics, and the subscribers that listen to the messages with their topics of interests. Publishers and subscribers are controlled by brokers that ensure the data is delivered from right publishers to right subscribers. Publishers and subscribers are agnostic to each other so that publishers send the messages to brokers and brokers send the message to the subscribers, in other words, brokers do the routing of the data [15]. Publish-Subscribe architecture in simple terms is shown in Figure 2.2.

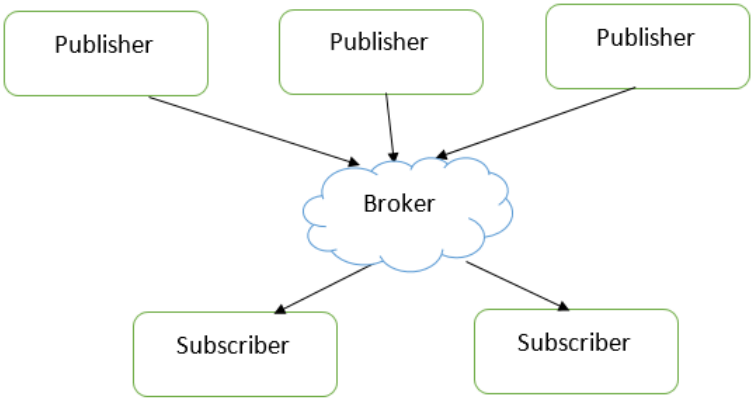


Figure 2.2: Publish-subscribe system architecture.

Micro Batching is another methodology similar to batch processing, where the streaming data is accumulated in batches before being processed. In micro batching, smaller batches are used and processing is done more frequently. Micro batching is a middle ground between streaming data analysis and batch analysis approaches, allowing batch processing methods to be used while making it possible to achieve near real-time latency by tuning the batch size; thus optimizing the trade-off between latency, throughput and accuracy [2]. Figure 2.3 represents a typical micro batching approach.

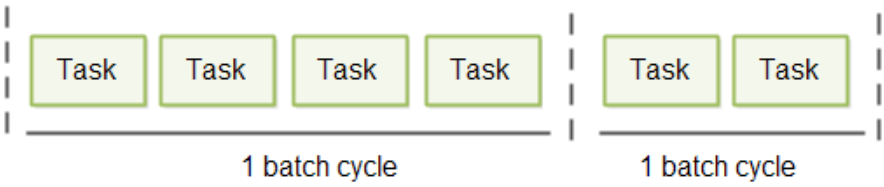


Figure 2.3: Micro batching example where small number of data points (tasks) are grouped and processed as micro batches (adapted from [2]).

2.2 Data Streaming Tools

Data Streaming Tools are necessary when low latency or real-time processing is a requirement to analyze the streaming data [16]. Some of the most commonly used data streaming tools are given below.

Apache Kafka is one of the most well known, open-source distributed messaging systems which was developed at LinkedIn [17, 9]. While Kafka was developed with Scala and Java, it supports programming languages such as C, C++, Java, Python. Apache Kafka manages the messages, distributes data streams on several machines and provides asynchronous communication. It has producers, consumers and brokers which essentially form a publish-subscribe system. Kafka is a highly scalable and fault-tolerant system that supports high throughput message delivery [18]. How long the messages will be stored in a Kafka cluster can be configured so Kafka can be configured with lower retention settings when fault tolerance is not critical or the data can be kept forever to use Kafka as a persistent database system. Kafka supports exactly-once semantics (even if the message is sent multiple times, the consumer takes it only once) without message order conservation [19]. To run Kafka, there is

a need for Apache Zookeeper application that stores the metadata and manages the status of Kafka [19].

RabbitMQ is an open-source publish-subscribe message broker system [20]. RabbitMQ was developed with Erlang language and for the running environment, Erlang should be installed. It uses Advanced Message Queuing Protocol (AMQP) which is an asynchronous messaging middleware that provides accepting connections from different platforms. RabbitMQ is a load balancing and fault-tolerant system. RabbitMQ supports at least once semantics with order conservation, by using queue structures for the message order guarantees [19]. RabbitMQ does not store messages, therefore there is no need for disk space, it can be run in a small environment [20].

ActiveMQ is an open-source general purpose message broker system. It was developed with Java and uses Java Message Service (JMS) which is an asynchronous messaging middleware that only accepts connections from Java platforms. In ActiveMQ, producers push the message to all consumers and they are responsible for the message delivery. To prevent message loss, broker stores messages in a persistent data store [20]. ActiveMQ supports once and only once messaging semantic [21].

In summary, the data streaming tools mentioned above are all open-source software that provide asynchronous communication, low latency, load balancing, and fault-tolerance. They are messaging systems that can be used for managing a stream of data by selectively directing multiple data sources to a single target or broadcasting a single data source to multiple targets. Moreover, these tools can be integrated with data processing tools and machine learning tools and libraries and this integration provides the collected data from multiple source data streams to be directed to the tools for analysis. All of them support Java programming language, and all of them have advantages and disadvantages. For example, in a limited environment such as a Raspberry Pi, RabbitMQ is more suitable than the others; if there is a need to process big streaming data, Apache Kafka is more reasonable, because of the high throughput. The comparison of these tools is given in Table 2.1.

Table 2.1: Comparison of data streaming tools.

Messaging System	Messaging Semantic	Other Features
Apache Kafka	Exactly once (no ordering guarantees)	• Long term message storage • High throughput • Scalable • Suitable for IoT applications • Developed with Scala and Java • Supports diverse languages (C, C++, Java, Python, Scala, Go, etc...) • Apache Zookeeper required
RabbitMQ	At least once (ordering guaranteed)	• Uses AMQP standard • Accepts connection from different platforms (C#, Java Platforms, etc...) • Disk space not required • Scheduled data sending • Suitable for restricted environment • Developed with Erlang • Erlang required
ActiveMQ	Once and only once	• Message storing • Uses JMS Standard • Accepts connection only from Java platforms • Scheduled data sending • Developed with Java

2.3 Streaming Data Processing Tools

In this section, well-known general purpose streaming data processing tools are described. These tools have a variety of uses from simple aggregations to Map-Reduce type of workloads. Moreover, by providing additional libraries and extensions, they also support application of machine learning methods on streaming data. Table 2.1 summarizes the specifications of these tools.

Apache Spark uses the micro batching methodology and splits the input query into multiple sub-queries, which relies on cluster computing [22]. Spark is an efficient tool for analyzing heterogeneous data and with Spark Streaming library, it becomes a scalable and high-throughput framework [11]. Spark has much higher latency because of its design compared to similar platforms such as Storm and Flink [23]. Spark supports exactly-once messaging assurance. Spark's machine learning library called MLlib provides clustering, classification, regression and evaluation methods [24].

Apache Storm is an open-source distributed streaming data processing system. It was developed for analyzing in real time structured data and unstructured data such

as video, audio [11]. Storm is a fault-tolerant, fast, and reliable system with using two daemons that are master node and slave nodes. Master node assigns the process to the slave nodes and when there is a failure in the process that is run by a slave node, master node reassigns the same process to another slave node [25, 11]. It provides once at a time processing which is one of the most suitable methods for low latency requirements. Storm also supports micro batching processing that provides at least once semantic [26]. Storm's library Scalable Advanced Massive Online Analysis (SAMOA) based on Massive Online Analysis (MOA) [12] provides classification, clustering, and regression methods for machine learning [27].

Apache Flink is an open-source distributed streaming data processing tool [28]. It was developed for analyzing streaming data and batch data [11]. Flink is very robust to fluctuations in the data arrival rate in aggregation workloads [22]. Flink was implemented in Java, and it uses a mechanism called check pointing to guarantee to process. Flink supports exactly-once messaging assurance and supports SQL similar to Apache Spark. FlinkML is an Apache Flink's machine learning library that provides data pre-processing, supervised and other learning methods [29].

Microsoft Azure Stream Analytics is a proprietary streaming data processing tool as a cloud service. It is a flexible, reliable, scalable (it allows scale up and scale out), fully managed serverless (users don't need to manage hardware or clusters) system. It assures security by encoding all communication channels. Azure Stream Analytics uses Stream Analytics Query Language for the defining process. It guarantees exactly-once messaging. Azure Stream Analytics provides anomaly detection and sentiment analysis as a machine learning techniques [30].

Streaming Data Processing Tools have all low latency and they are fault-tolerant systems. All of them support micro-batching methodology. Except for Azure Stream Analytics, they support Java programming languages. According to the usage, platform, and needs for the stream data processing, the most suitable one can be selected. As an example; if data is unstructured, then Apache Storm becomes more useful than the others; however, if there is a need for batch analysis, then Apache Flink becomes more advantageous. Azure Stream Analytics is focused on sentiment analysis in the machine learning part. The comparison of these tools is shown in Table 2.2.

Table 2.2: Comparison of data processing tools.

Framework	Messaging Semantic	Other Features
Apache Spark	Exactly once	• Open source • Scalable • SQL support • Supports Java, Python, and Scala • Provides classification, clustering, regression and evaluation methods with MLlib
Apache Storm	At least once	• Open source • Scalable • Suitable for structured and unstructured data • Supports Java • Provides classification, clustering, and regression methods with SAMOA
Apache Flink	Exactly once	• Open source • Event time processing • SQL support • Supports Java, Python, and Scala • Provides supervised and unsupervised learning with FlinkML • Provides data pre-processing with FlinkML
Microsoft Azure Stream Analytics	Exactly once	• Proprietary • Cloud service • Scalable • Uses Stream Analytics Query Language • Extensible • Provides sentiment analysis and anomaly detection

2.4 Machine Learning Tools and Libraries for Data Streams

There are a few streaming data analysis tools and libraries that specialized for machine learning. Some of the essential features included in these tools are being open-source, easy to use, extensible and the capability of supporting multiple machine learning algorithms. Table 2.3 presents a summary of the machine learning tools and libraries discussed in this section.

Massive Online Analysis (MOA) is one of the most popular open source Java frameworks for data stream analysis. It is based on Waikato Environment for Knowledge Analysis (WEKA) [13] which is an open-source software for the field of machine learning that supports both supervised and unsupervised learning [31], and includes algorithms for regression, classification, clustering, and attribute selection [32]. It is a software environment for implementing algorithms and running experiments for

online learning from data streams. It also provides data stream generators [12]. MOA includes a collection of offline and online methods such as classification, regression, clustering, outlier detection, and recommender systems as well as tools for evaluation such as Holdout and Prequential [33]. It recently started to support multi-label classification and graph mining [34]. MOA can be used as both a stream processing tool and an environment to develop a stream processing algorithm, however it is not suitable for large scale applications because of the lack of scalability [12, 35].

Scikit-Multiflow is an open-source Python framework for machine learning from data streams based on Scikit-Learn that is one of the most popular open-source software machine learning libraries focused on batch learning with a wide range of state-of-the-art machine learning algorithms for medium-scale supervised and unsupervised problems [36]. Scikit-Multiflow provides multi-output learning that can produce multiple outputs to assign to multiple target variables. In other words, the dimensions of output can be more than one. It is built upon Scikit-Learn, MOA, and WEKA. It provides multiple learning methods such as Support Vector Machines, Random Forests, k-means; data generators such as Agrawal, Hyperplane, Random-RBF, Random Tree; and evaluators such as prequential and holdout for different stream learning problems, including single-output, multi-output, and multi-label. It complements Scikit-Learn, whose main focus is batch learning [37]. Unfortunately, scikit-multiflow does not have a user interface and it requires external tools, and libraries such as Docker.

R is an open-source software environment which is generally used for statistical computing [38]. It bundles a framework for data stream modeling and associated data mining tasks such as clustering, and classification. The packages in R provide support for modeling and simulating data streams as well as an extensible framework for implementing, interfacing and experimenting with algorithms for various data stream mining tasks [39]. It supports various types of analysis approach clustering, classification, outlier detection, regression and evaluation algorithms. Some of the available algorithms are k-means, StreamKm++, DenStream, CluStream, kNN, Support Vector Machines, Decision Trees, and Local Outlier Factor [14].

Streaming data analysis tools mentioned above are desktop applications and all of

them are open source. However, none of them are web-based applications. While Scikit-Learn and Scikit-Multiflow provide diverse algorithms for Python users, MOA is offering a wide range of algorithms for the Java users. The comparison between these analyze tools shown in Table 2.3.

Table 2.3: Comparison of streaming data analysis tools.

Data Analysis Tool	Language	Supports
MOA	Java	• Supervised learning • Unsupervised learning • Stream generators • Classification algorithms • Regression algorithms • Clustering algorithms • Outlier Detection algorithms • Evaluation methods
Scikit Multiflow	Python	• Clustering algorithms • Classification algorithms • Multi-output learning • Learning methods • Data generator • Evaluation methods
R	R	• Clustering algorithms • Classification algorithms • Outlier detection algorithms • Regression algorithms • Evaluation algorithms

Overall, all the approaches that are mentioned about data streaming, streaming data processing, and machine learning tools offer extensive features and commonly used in the data streaming analysis studies. However, all of these approaches have long lists of steps to getting started and steep learning curves, causing researchers to spend considerable time to prepare before starting to work.

CHAPTER 3

EASY STREAM ANALYSIS TOOL (ESTRA) AND ITS ARCHITECTURE

As described in the previous chapter, many studies have been done on data stream analysis and learning, thus there are plenty of tools and frameworks are available. However, the learning curves of the tools are steep, installation process is long and many of them need to be integrated together. Therefore, getting into streaming data analysis becomes more challenging for inexperienced researchers and more time and effort is required to perform experiments on streaming data. Easy Stream Analysis Tool (ESTRA) is designed and implemented to provide an easy-to-use, web-based and beginner-friendly platform with support for various datasets, dataset generators and streaming data analysis algorithms to help researchers to spend minimum time on preparing their working environment and to start quickly performing their experiments.

In this chapter, the architecture of ESTRA and the employed technologies are described. ESTRA is designed with the aim of creating an easy-to-use data stream analysis platform that serves the purpose of a quick and efficient tool to explore and prototype machine learning solutions on various datasets. Therefore, ESTRA is developed as a web-based, scalable, extensible, and open-source data analysis tool with a user-friendly and easy to use user interface. On top of that, ESTRA comes with a bundle of datasets, dataset generators, and implementations of various analysis and learning algorithms. Moreover, ESTRA provides an easy way to investigate various properties of the datasets and to observe the results of executed machine learning algorithms.

3.1 Architecture

ESTRA consists of four main components that are shown in Figure 3.1.

- User Interface (Web Browser);
- Web Server;
- Database;
- Background Worker.

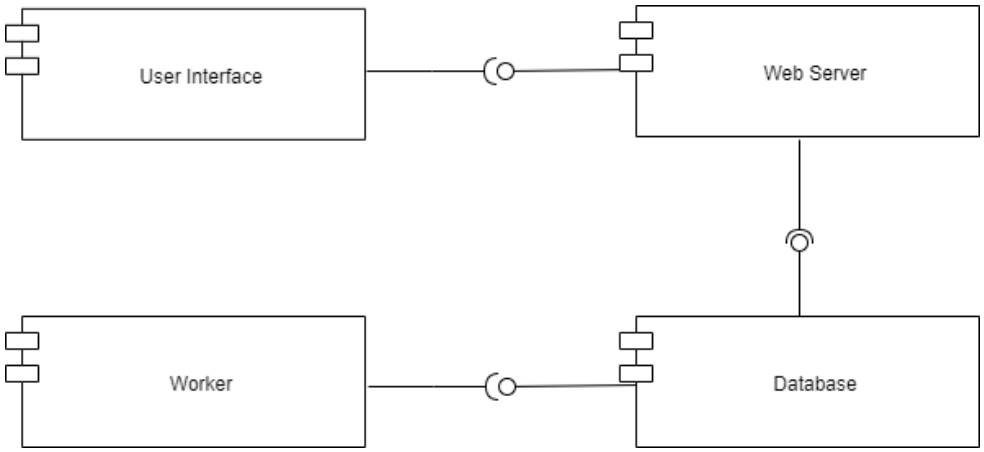


Figure 3.1: Component model that shows the relation between components.

User interface is the component that the user interacts with in order to access the datasets and algorithms implemented in ESTRA. On a high level, the user interface of ESTRA is developed as the client of a client-server architecture. As in a typical web-based client-server application, HTTP requests are used as the main protocol between the user interface and the backend components. This allows ESTRA to benefit from many available tools and libraries, as HTTP is a common protocol with broad support among almost all programming languages [40].

Essentially, user interface works as a single-page application (SPA) [41] developed with ReactJS, a popular user interface library. As in a typical SPA workflow, the client sends HTTP requests to a web server via FETCH API and updates the user interface partially as it receives a response from the web server. In ESTRA, the client application periodically polls the web server to fetch the latest state of the processes

and if there is any change in the state or details of the executed machine learning jobs (processes), then the client-side updates the relevant areas. Similarly, any new machine learning execution submission by the user in the user interface creates a request that is transmitted to the web server. For the user interface design, ESTRA leverages "material-ui" library that provides out-of-shelf user interface components, themes, icons, and accessibility features that are compatible with React [42]. For visualization of the machine learning algorithm results, a chart rendering library named "reCharts" is used that allows ESTRA to include readable visual graphs [43].

Web server component is a lightweight web application written with Django which is a web application framework for Python. Web server component is a kind of bridge between the client and the workers of ESTRA and actually contains minimal business logic. Web server provides endpoints to the user interface component for either taking new machine learning job requests or return the state of the machine learning jobs submitted by the user, including the states and results. Moreover, the web server component contains the database models, describing the table and index structure used in the database.

Database component is responsible for storing the data generated by the system which are mainly the set of job requests by the users, their state, and results. The database component acts as the main source of truth and the only non-volatile place for the system. In other words, the Database component contains the state of the ESTRA system. The database component is an instance of the PostgreSQL [44] database that holds the requests of the clients, the request status, and the analysis results in a simple PostgreSQL table. PostgreSQL is an open-source database management system that has wide adoption in the world and it is used by many small and large companies. It has many extensions for various use cases in addition to being easy to use and scalable. The only table in the database is named "jobs" and it contains the details of the requests such as the selected dataset, algorithm, and its parameters, as well as the results obtained as the corresponding machine learning process, is executed. The database model of a process is presented in Figure 3.2.

The details of the fields that are used in database model (Figure 3.2) are given below.

- Dataset Name represents the selected dataset that can be a built-in dataset such

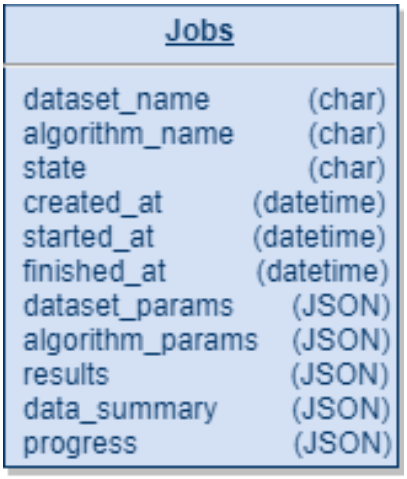


Figure 3.2: Database model of a process.

as Electricity or a generated dataset such as Sea generated dataset.

- Dataset Parameters is for the parameters of the selected dataset generator. For generated datasets, the dataset parameters correspond to the specified generator's parameters. For SEA, this is Noise Percentage and for Hyperplane generator these are Feature Count, Drifted Feature Count, Magnitude Change, Noise Percentage and Sigma Percentage. More details on these datasets and their parameters are provided in Chapter 5.
- Algorithm Name is used for the selected algorithm which can be any algorithm described in Chapter 5.
- Algorithm Parameters represents the parameters of the selected algorithm. These parameters are described in Chapter 5.
- State is used for the state of the process. The valid values are "queued", "in progress", "failed" and "finished".
- Created At is the time that is used for when the user wants the run a process.
- Started At is the time that is used for when the process started.
- Finished At is the time that is used for when the process finished.
- Data Summary is used for the summary of the selected dataset. It includes; column names and mean, average, minimum and maximum values of each feature

(column), and sample counts.

- Progress is used for the online results of the analysis.

- Results is used for the all results of the analysis.

The logic of the application is defined in the *Worker* component implemented in Python. The implementations and usage of the algorithms are included in the Worker component. The workflow of this component is shown in Figure 3.3. Initially, the worker checks the jobs in the database, and if there is any job in the queue that is ready to be processed, then the worker takes the job, and the state of that process changes to "in progress" as shown in Figure 3.4. The worker starts running the selected algorithm with the dataset specified by the user. When the process is successfully completed, the worker then updates the database with the results and looks for another job.

3.2 Workflow

The communication flow between the components of ESTRA is shown in Figure 3.5. Fundamentally, the user initiates a machine learning activity (a process) by selecting the dataset, algorithm, and their parameters via the client application in their web browser. The browser sends the details of the process selected by the user to the web server component to be stored in the database and to be processed when a background worker is available. When the background worker becomes available, it runs the process by looking at the stored process details and writes the result to the database. During this process, the user interface component periodically asks the web server component for the updates, which in turn returns the current state of the processes in the database. Finally, when the worker component finished the process, the web server is able to provide the final results of the finished process to the user interface component and the user will be able to observe visually the results.

3.3 Deployment, Scalability, and Extensibility

ESTRA provides a flexible deployment structure depending on the use case. For personal use, ESTRA can be run on a regular personal computer. For a large scale use, every component can be deployed into their own servers and they can even be deployed as a load-balanced multi-instance fashion. ESTRA's deployment model is presented in Figure 3.6.

As in a typical web-based client-server SPA application, the user interface component scales as the actual code is run in a web browser on the user's computer. This means that the user interface component naturally scales as the user count grows and it is limited only by the computing power of the users' environment.

The web server component is a typical lightweight and stateless web application back-end; therefore it's really easy to scale it with the commonly used load balancing methods. The way to scale up the web server component is essentially obtained by deploying the web application to multiple server instances, placing a load-balancer in front of the server instances, and configuring the user interface to communicate with the load balancer instead of sending requests directly to the web servers.

Databases are generally one of the most complex components in a software system as they store the state of a system and act as the source of truth. ESTRA's database itself is rather simple and lightweight with only one data table with no complex relations and index structures. That makes the database component durable to scaling demands and withstand more than 10.000 of requests per second without adding any extra scaling measures [45]. However, if there is further request to scale up ESTRA's database, it is possible to make use of the popular PostgreSQL scaling methods such as partitioning, indexing and sharding within the web server or via PostgreSQL extensions.

Worker component actually contains the main logic along with the dataset generation and learning algorithms as well as the datasets. Thus, it is the first component that will be required to scale up. A worker component runs one machine learning job at a time, meaning that the number of available worker processes determines how many machine learning requests are processed concurrently. The worker components peri-

odically poll the database for any machine learning jobs that are ready to be processed and once a worker finds a job ready to be processed, the job is marked as "being processed" by the worker. This allows multiple workers to be deployed on the same system independently and without any performance impact on each other.

Scaled deployment of ESTRA component is provided in Figure 3.7.

ESTRA's straightforward and clean architecture with open source tools allows it to be highly extensible. The libraries and frameworks employed in ESTRA such as React, Python and Scikit-Multiflow are popular open source tools with broad community support and having several extensions. Combining this with ESTRA's open source distribution makes it easy for developers and researchers to modify and extend ESTRA.

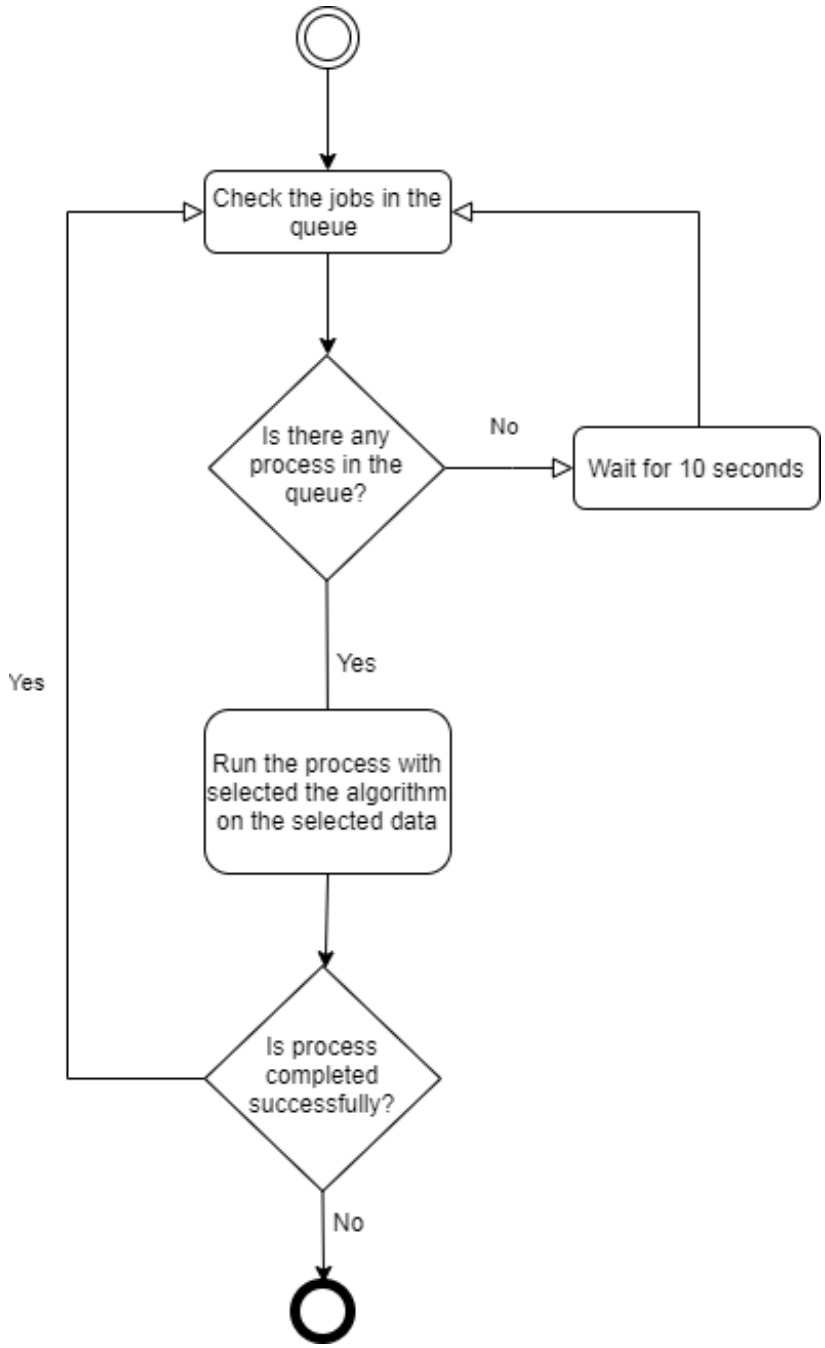


Figure 3.3: Activity diagram of a worker.

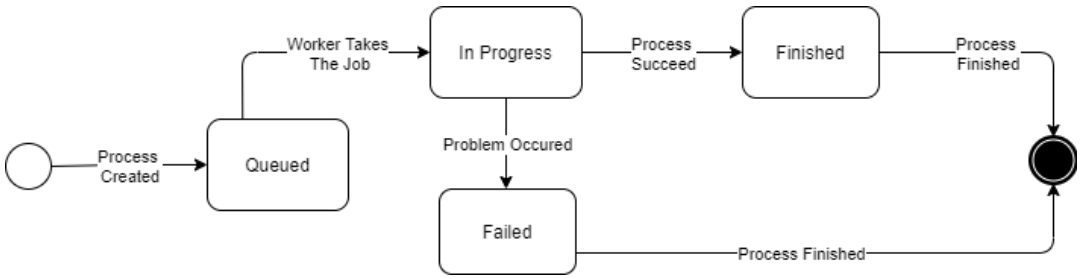


Figure 3.4: State machine diagram of the state of a process.

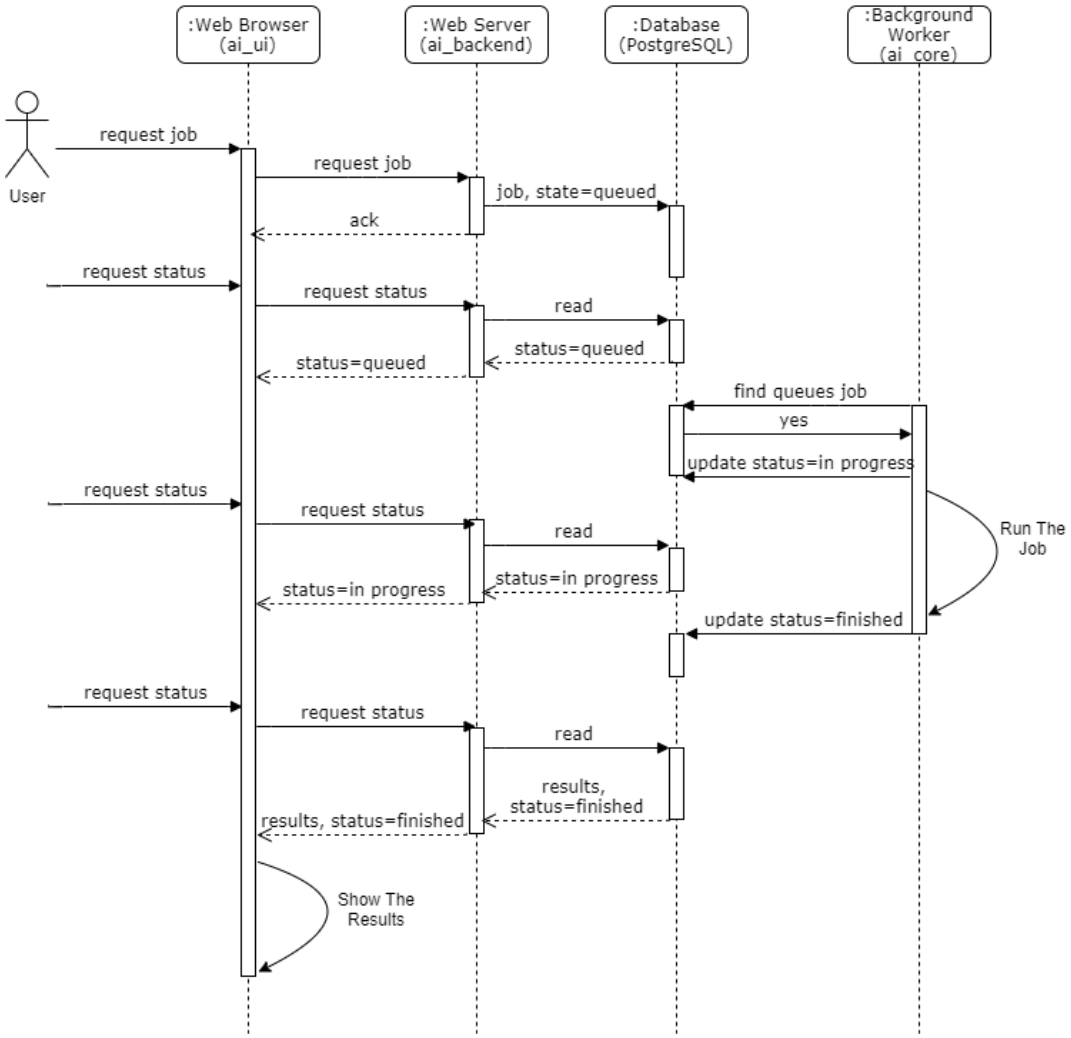


Figure 3.5: A sequence diagram that demonstrates the interaction between the user and the components of ESTRA.

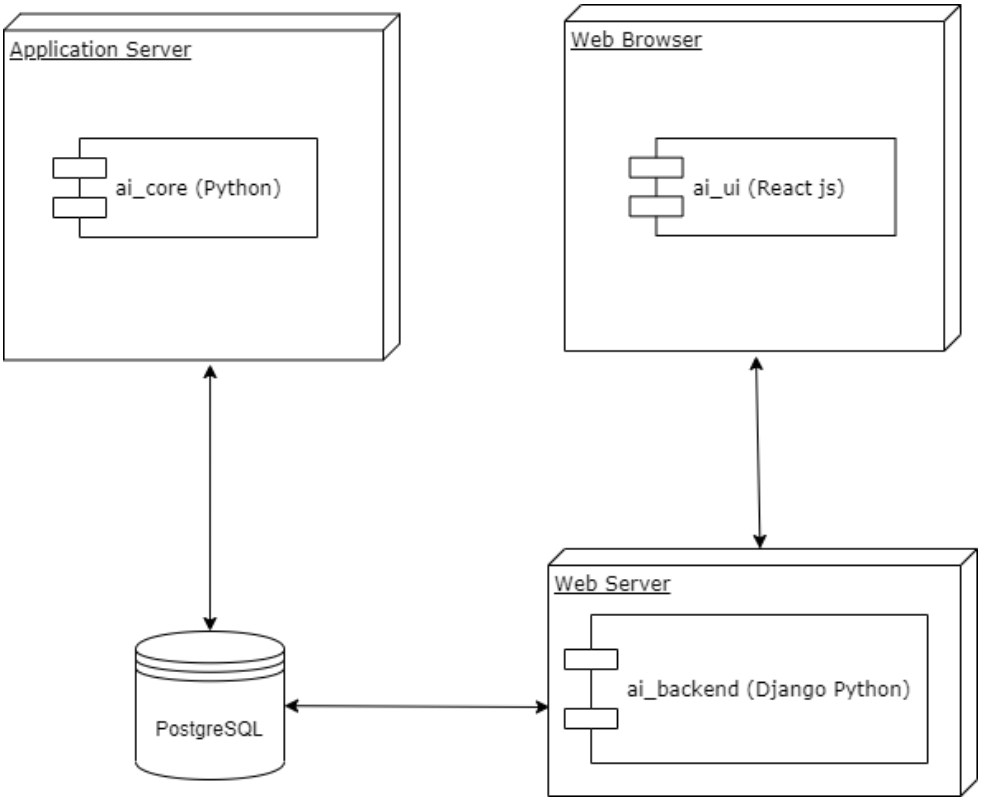


Figure 3.6: Deployment model of ESTRA.

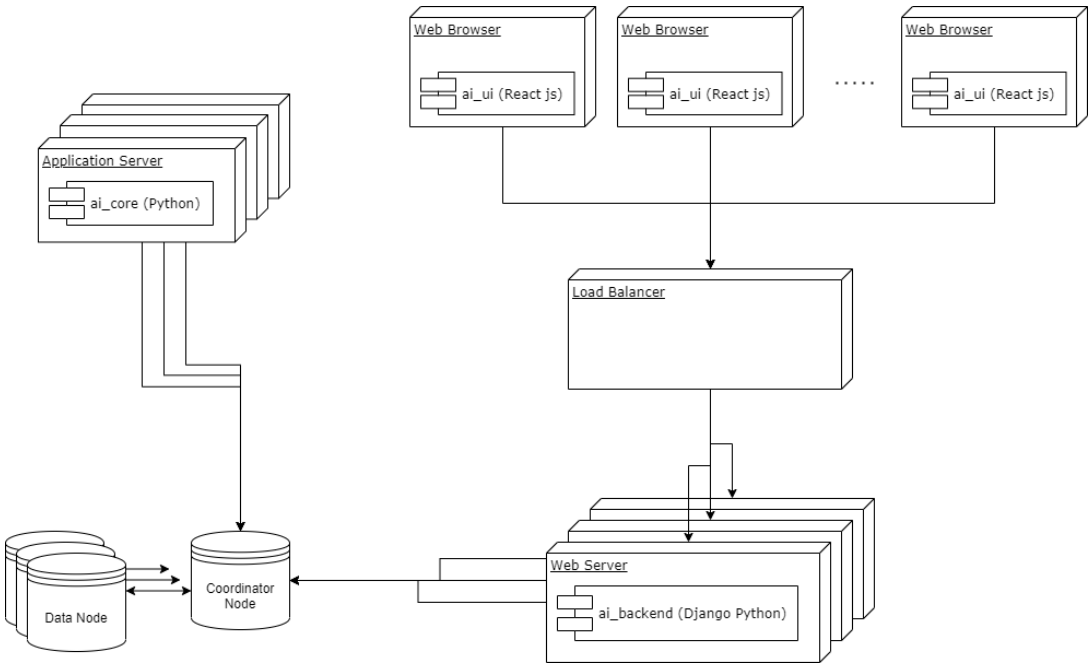


Figure 3.7: Scalability of the deployment model of ESTRA

CHAPTER 4

USAGE OF ESTRA

This chapter explains ESTRA's user interface and provides guidance on how to use ESTRA for streaming data analysis. ESTRA's main screen contains list of processes that has been executed so far and the "START NEW PROCESS" button for initiating a new streaming data analysis process (see Figure 4.1). The statuses of the processes in this list are indicated with the state icon in each row. Pressing the "START NEW PROCESS" button opens the new process dialog. This dialog contains three steps: dataset selection, algorithm selection, and review and run.

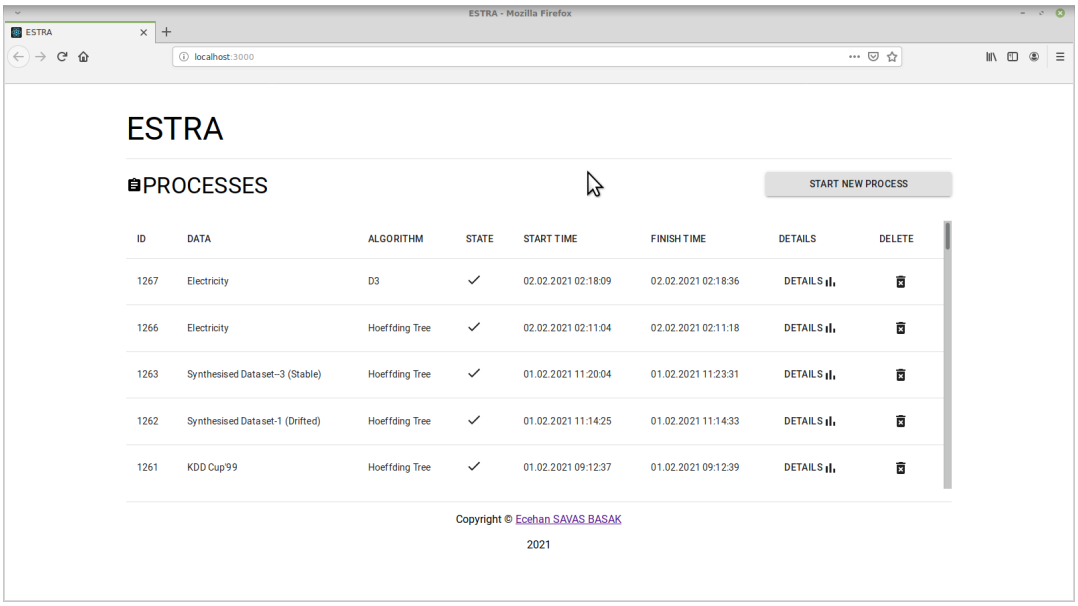


Figure 4.1: Main view of ESTRA

4.1 Dataset Selection

In Dataset Selection step of the new process dialog, users are able to choose datasets from two different categories: predefined datasets and dataset generators. In predefined datasets category, users can select one of the built-in datasets to be streamed in the analysis process as shown in Figure 4.2. In the dataset generators category, users can select one of the included data generators to generate data on-the-fly as well as modifying the parameters available for each data stream generator as seen in Figure 4.3.

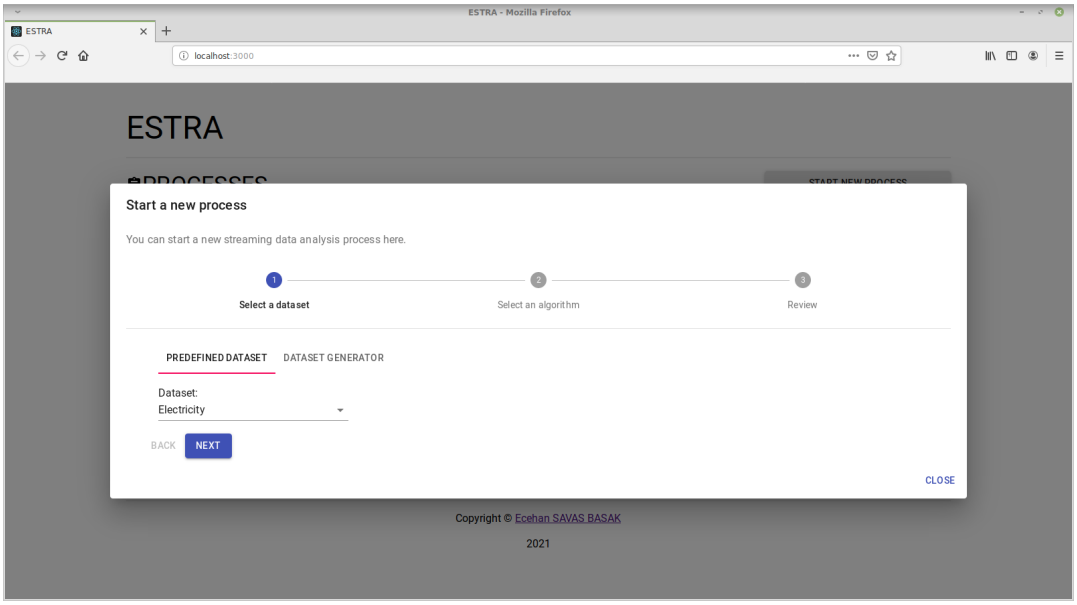


Figure 4.2: Built-in dataset selection.

4.2 Algorithm Selection

The second step of the new process dialog is the algorithm selection where users can select one of the integrated algorithms in ESTRA as well as modify the parameters of the selected algorithm as shown in Figure 4.4.

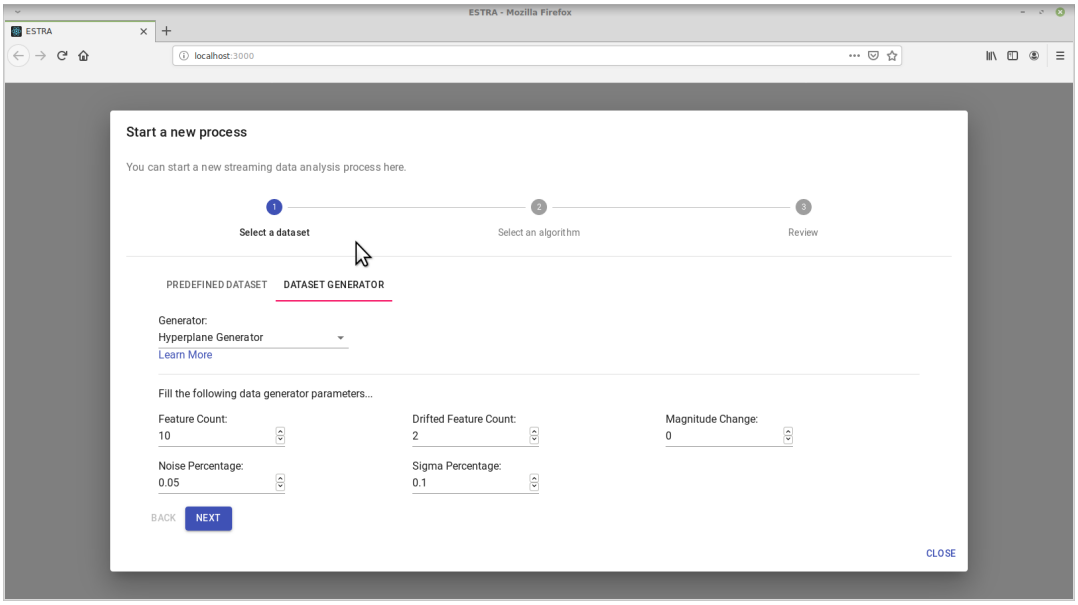


Figure 4.3: Dataset generator selection.

4.3 Review and Run

After selecting the dataset and the algorithm, the user is brought to the review step where the previously selected algorithm, dataset, and parameters are shown to the user for a final check as well as if there's any invalid inputs in the parameters (see Figure 4.5). Users can start the process after reviewing their selections.

4.4 Process List

When the user starts the new process, created process is inserted into the queue of waiting jobs as shown in Figure 4.6. At this point, the state of the process is labeled as queued and represented with a clock icon. When a worker is available, it picks the process waiting in the queue and starts the execution. The process that is being executed is indicated with a synchronization icon in the process list as in Figure 4.7. After the process is finished, the process list is updated with a tick icon representing the finished state and the finish time of the process as shown in Figure 4.8.

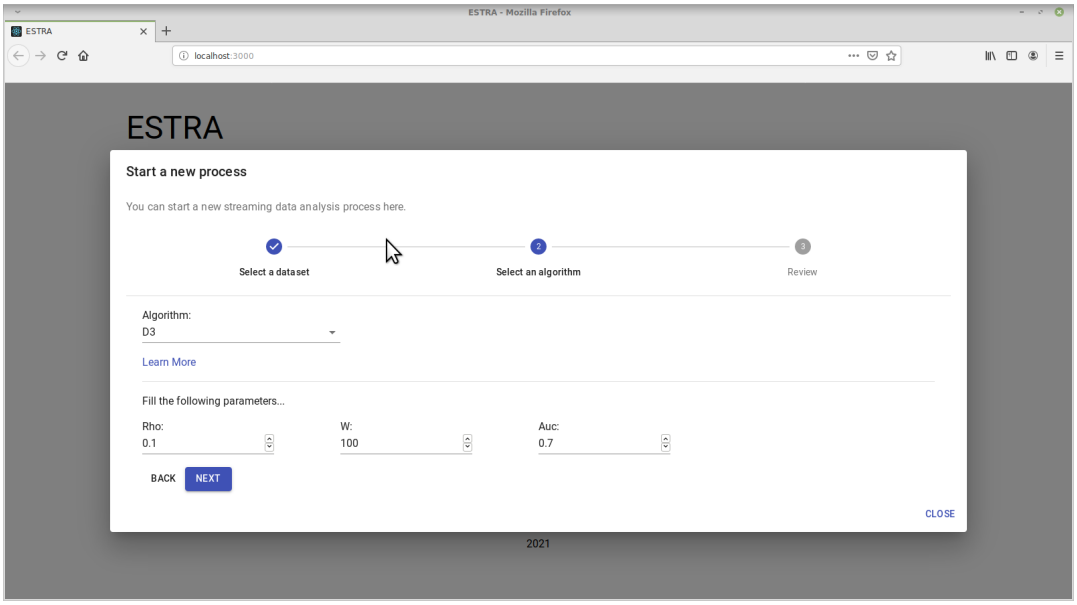


Figure 4.4: Algorithm selection

4.5 Analysis Results

The analysis result of a dataset can be seen any time after the process is started in the details dialog by clicking the "DETAILS" button. In the details dialog, the information about the executed process is displayed under three categories. The first category "DATA SUMMARY" category is a summary of the data that provides insights about the features of the used dataset are available as shown in Figure 4.9. The second category of "GRAPHS" contains visual representations of the analysis results in the form of line charts as demonstrated in Figure 4.10. The charts are generated according to the relevant metrics for each algorithm described in Table 5.2. Finally, in the third "RAW RESULTS" category, the obtained results from the analysis in json format are presented, which can bee seen in Figure 4.11.

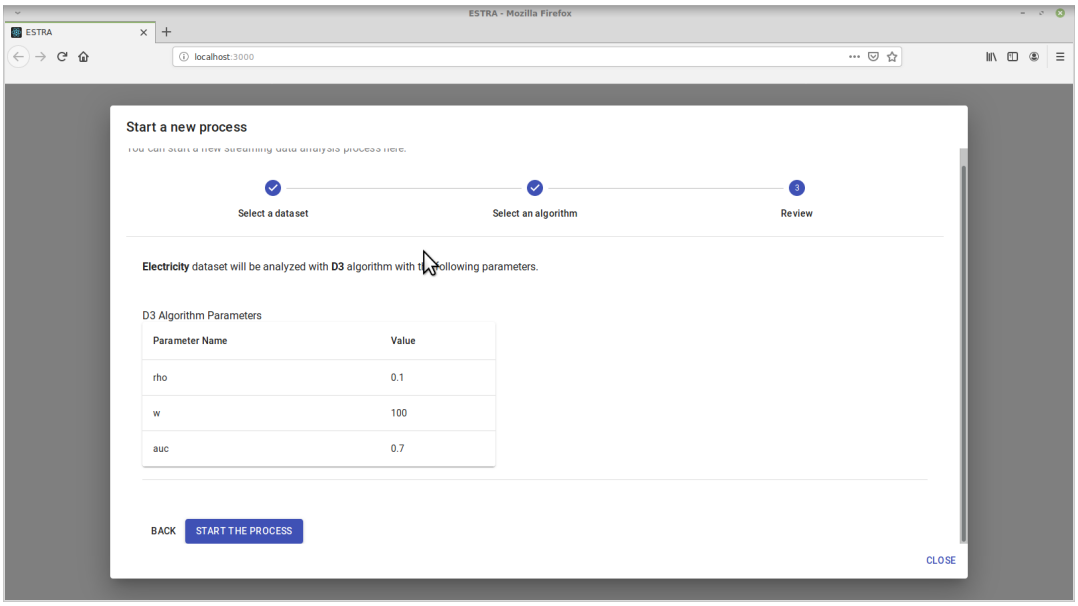


Figure 4.5: Review of selected dataset, algorithm and their parameters.

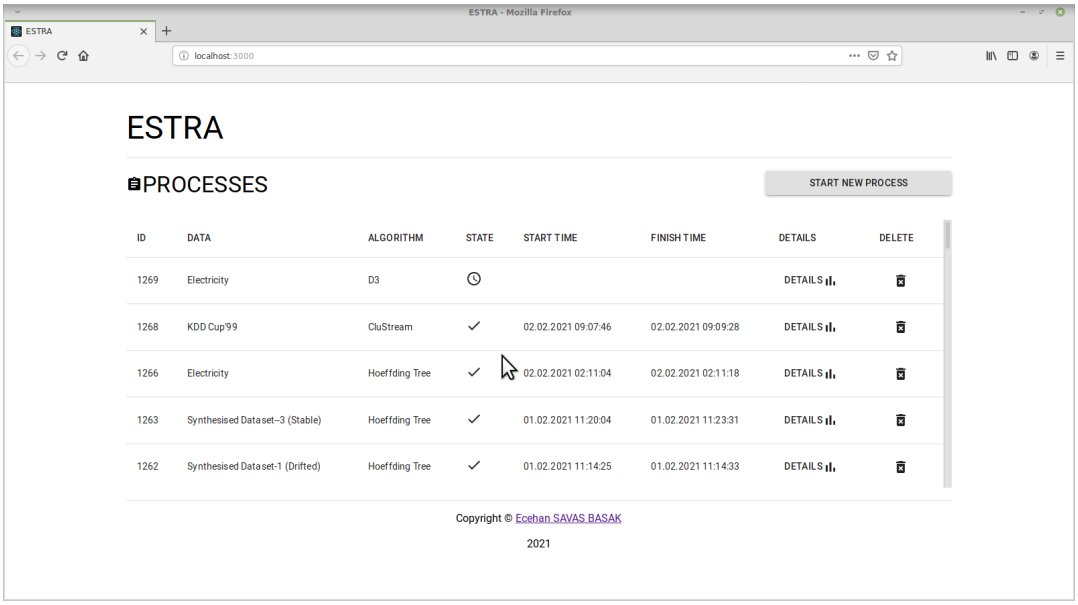


Figure 4.6: Process added in the queue for execution.

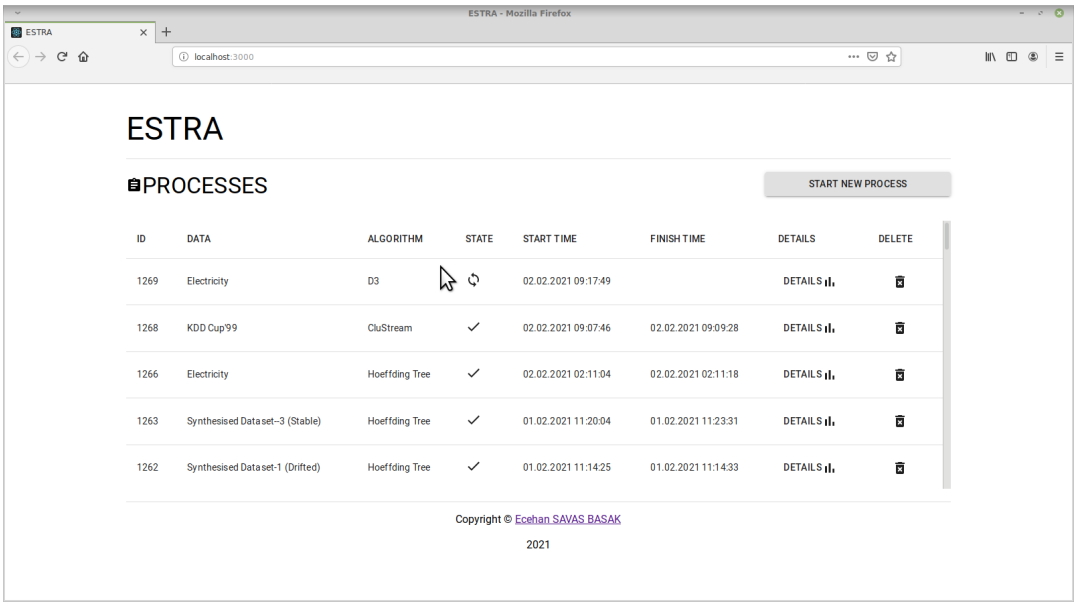


Figure 4.7: Process is in progress.

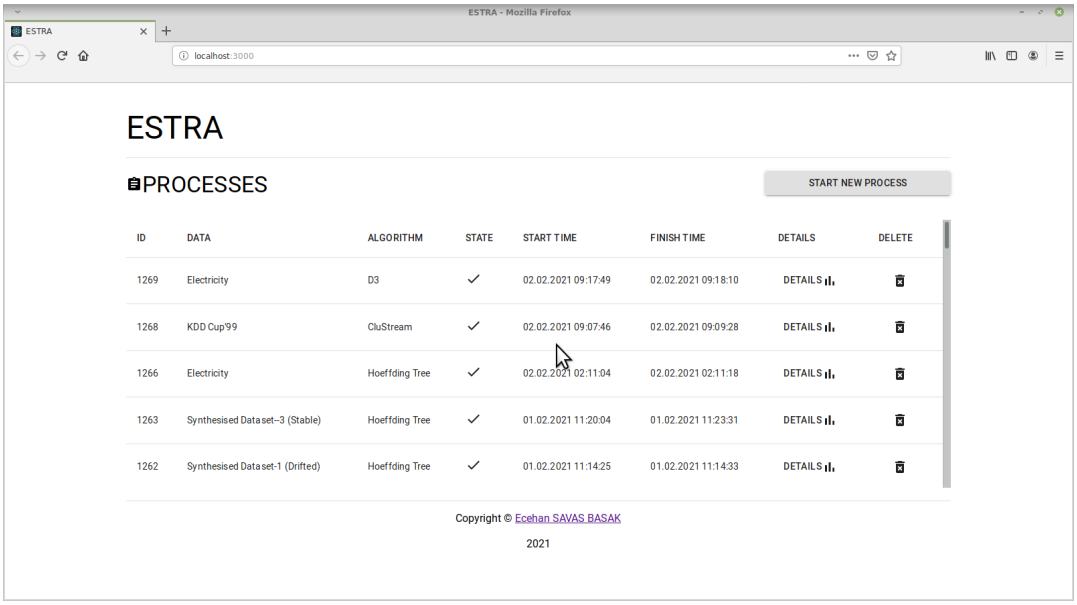
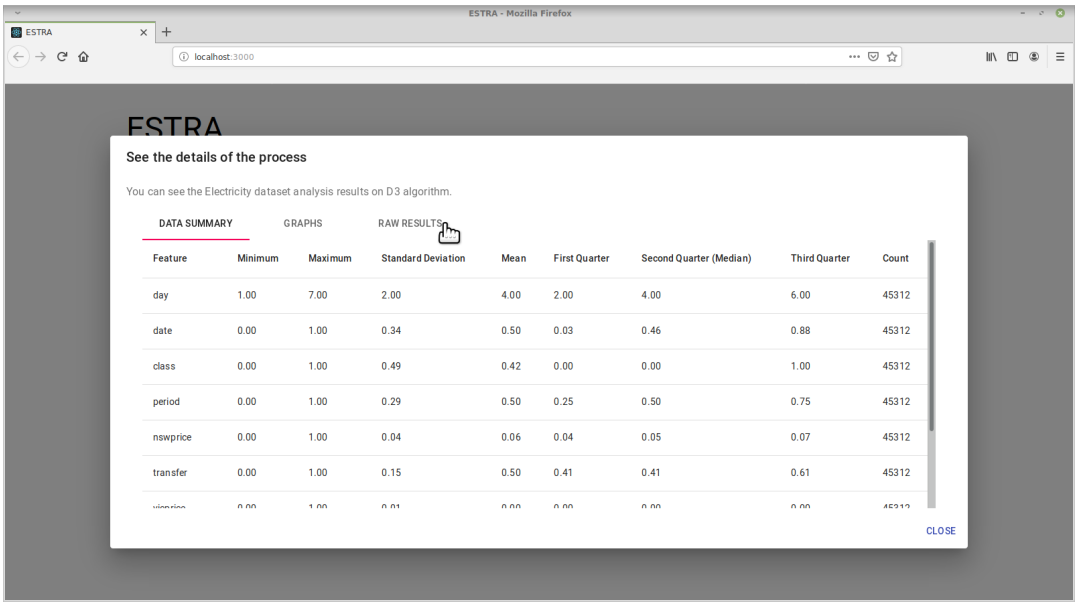


Figure 4.8: Process is finished.



Feature	Minimum	Maximum	Standard Deviation	Mean	First Quarter	Second Quarter (Median)	Third Quarter	Count
day	1.00	7.00	2.00	4.00	2.00	4.00	6.00	45312
date	0.00	1.00	0.34	0.50	0.03	0.46	0.88	45312
class	0.00	1.00	0.49	0.42	0.00	0.00	1.00	45312
period	0.00	1.00	0.29	0.50	0.25	0.50	0.75	45312
nswprice	0.00	1.00	0.04	0.06	0.04	0.05	0.07	45312
transfer	0.00	1.00	0.15	0.50	0.41	0.41	0.61	45312

Figure 4.9: Summary of the dataset.

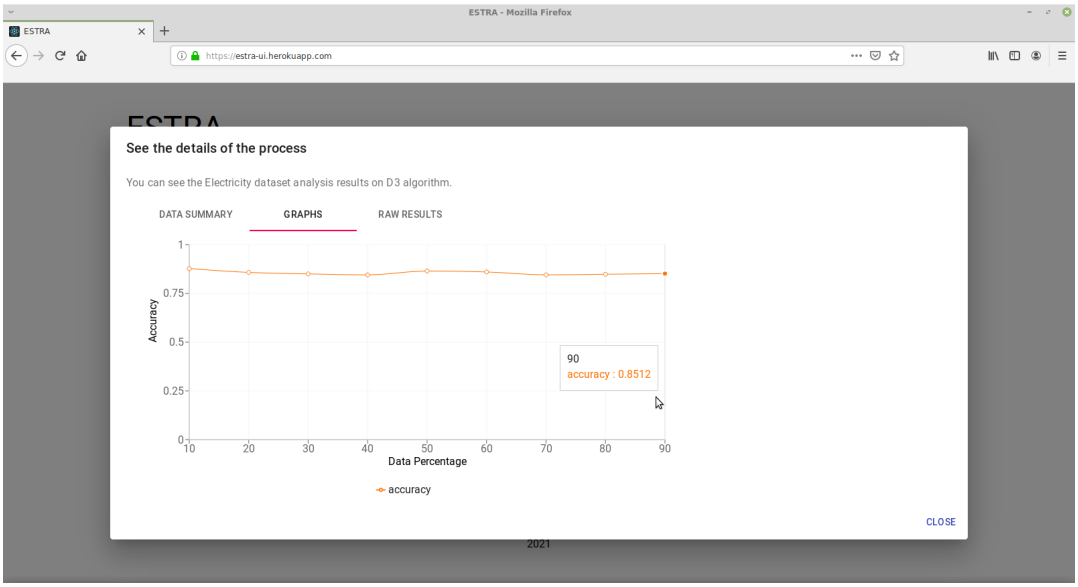


Figure 4.10: Graphs of the analysis results.

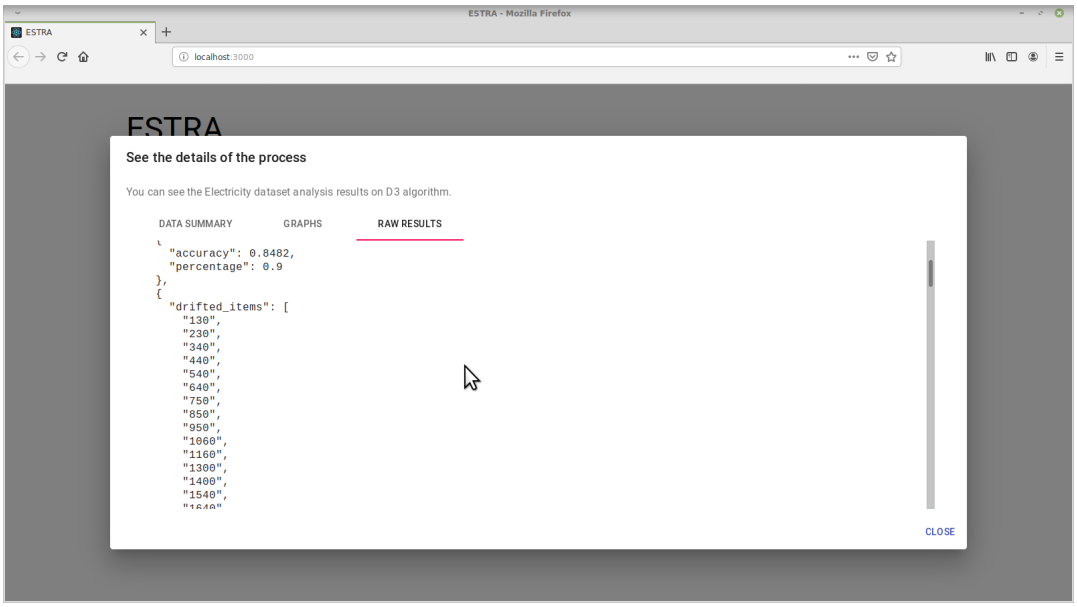


Figure 4.11: Raw results obtained from analysis.

CHAPTER 5

LEARNING ASSESSMENTS, ALGORITHMS, AND DATASETS

This chapter describes the learning assessments used in ESTRA as well as available algorithms and datasets. ESTRA utilizes well known languages and libraries such as Massive Online Analysis (MOA), Scikit-Multiflow and R for the integration of various machine learning algorithms, dataset generators and data streaming methods.

5.1 Learning Assessments

The results of the machine learning algorithms can be assessed through various metrics. In this study, purity, accuracy, and adjusted rand index are employed for evaluation purposes.

Purity is a measure that indicates the homogeneity of a clustering. As more data points that belong to the same class land in the same cluster, the purity value (cluster quality) increases. Given a clustering of (N) data points with a set of classes (D) into a set of clusters (K), the purity is calculated as the sum of the number of instances in each cluster that belong to the most commonly found class in that cluster, divided by the total number of instances [46] as described in Equation 5.3 that is adapted from [47].

$$\textit{Purity} = \frac{1}{N} \sum_{k \in K} max_{d \in D} |d \cap k| \tag{5.1}$$

Accuracy in a classification problem is the metric that represents how close the obtained results are to the true values [1]. In general, the accuracy is calculated as the ratio of true predictions to all predictions made. For example, in a binary classifica-

tion problem where two classes are described as positive or negative and a classification is called true if the predicted result is the same with the real result as shown in Figure 5.1, accuracy is calculated as the total number of true positive classifications (TP) and true negative classifications (TN), divided by the all classifications ($TP, TN, FP, andFN$) made as can be seen in Equation 5.2 that is adapted from [1]).

Table 5.1: The confusion matrix (adapted from [1]).

	Actual Condition Positive	Actual Condition Negative
Predicted Condition Positive	True Positive (TP)	False Positive (FP)
Predicted Condition Negative	False Negative(FN)	True Negative (TN)

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{5.2}$$

Rand Index is a measurement that calculates the similarities of two clusters (see Equation 5.3) that is adapted from [48]. Rand Index is calculated as the ratio of agreements between two clusters, where in a dataset of n instances, agreements are defined as the total number of instance pairs ($\binom{n}{2}$) that are grouped in the same cluster in both clusters (a) and the number of instance pairs that are grouped in different clusters in both clusters (b) while disagreements are the number of instance pairs that are grouped in different clusters in one clustering while grouped in the same clusters in the other clustering (c and d) [48]. The definition of Rand Index is highly similar to the definition of Accuracy. The difference is that, accuracy is used for classification problems where class labels are known while Rand Index is applied on the results of clustering. Due to the lack of class labels, Rand Index does not penalize the assigning wrong labels to the clusters as long as the points in the same class are grouped in the same cluster.

$$RI = \frac{a + b}{a + b + c + d} = \frac{a + b}{\binom{n}{2}} \tag{5.3}$$

Adjusted Rand Index is a variant of Rand Index where the agreements between different clusters in the compared clustering results are taken into account as expected similarity and the number of overlaps between different clusters are compared with

the expected similarity. Instead of correcting the result by the number of agreements with the permutation of the possible distributions as in Rand Index, Adjusted Rand Index uses the expected values to normalize the result. Since the calculated similarity is compared with the expected similarity, Adjusted Rand Index can take negative values. In order to calculate Adjusted Rand Index, first a rxs contingency matrix needs to be generated where r represents the number of clusters in the first clustering, s represents the number of clusters in second clustering and values in the matrix n_{ij} is the number of data points common between the ith cluster of the first clustering and jth cluster of the second clustering as shown in Figure 5.1. Then the values in the contingency matrix is calculated to evaluate the found index, expected index and maximum index as shown in Equation 5.4 that is adapted from [3].

$X \backslash Y$	Y_1	Y_2	$\cdots$	Y_s	sums
X_1	n_{11}	n_{12}	$\cdots$	n_{1s}	a_1
X_2	n_{21}	n_{22}	$\cdots$	n_{2s}	a_2
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
X_r	n_{r1}	n_{r2}	$\cdots$	n_{rs}	a_r
sums	b_1	b_2	$\cdots$	b_s	

Figure 5.1: Contingency matrix for ARI calculation of two clusterings (adapted from [3]).

$$ARI = \frac{\sum_{ij} \binom{n_{ij}}{2} - \left[\sum_i \binom{a_i}{2} \sum_j \binom{b_j}{2}\right] / \binom{n}{2}}{\frac{1}{2}\left[\sum_i \binom{a_i}{2} + \sum_j \binom{b_j}{2}\right] - \left[\sum_i \binom{a_i}{2} \sum_j \binom{b_j}{2}\right] / \binom{n}{2}} \tag{5.4}$$

Employed algorithms and the corresponding evaluation metrics are provided in Table 5.2. It should be noted that the performance criteria between the classification and clustering algorithms are different. For classification algorithms where the real label values are known, it's possible to compare the real labels with the predicted labels that are the output of the algorithm and the used metric is "Accuracy". However, for clustering algorithms, accuracy is not an applicable metric, since accuracy compares the predicted labels with the real labels, which are not available in clustering algorithms. Therefore, "Purity" and "Adjusted Rand Index" metrics are utilised.

Table 5.2: Applicable metrics for each algorithm.

Algorithm \ Metric	Accuracy	Purity	Adjusted Rand Index
k-means	No	Yes	No
kNN	Yes	No	No
D3	Yes	No	No
CluStream	No	Yes	Yes
DenStream	No	Yes	Yes
StreamKM++	No	Yes	Yes
Hoeffding Tree	Yes	No	No

5.2 Algorithms

There are various algorithms for data classification and clustering in the literature. Some of them are especially suitable for data stream analysis. In this section, clustering, classification, and drift detection algorithms that are available in ESTRA are explained. k-means, kNN, D3, DenStream, CluStream, StreamKm++, and Hoeffding Tree algorithms have been integrated in ESTRA. The motivation for choosing these algorithms is that they are well known for data stream analysis and they have already been employed in stream analysis studies. The details of these algorithms are provided in this section.

The details of the algorithms available in ESTRA are given below. Working principles and parameters that are configurable via ESTRA of each algorithm are explained.

k-Means is one of the most popular clustering algorithms that splits the dataset into k different clusters [49]. First, k random points within the datasets are chosen as initial cluster centroids. Then, each data point is assigned to the closest centroid, forming the clusters. After all data points are assigned to closest centroids, the cluster centroids are adjusted as the center of assigned data points. Until the centroids converge toward stable points or a predefined maximum number of iterations have been achieved, steps of assignment of data points and adjustment of cluster centroids are repeated [50]. To

decrease the chance of falling into local optima, the algorithm can be repeated many times with different initial centroids.

For k-means algorithm's implementation in ESTRA, Scikit-Multiflow libraries are used, and several parameters are configurable in order to find the most suitable clustering results.

- Cluster Count represents the predefined number of clusters that the data points will be grouped into, which is also the number centroids being used and also known as the k value.
- Maximum Iteration Number determines the upper bound for the number of repetitions of the cycle of assigning data points to the proper centroids and adjusting centroid positions according to the assigned data points.
- Number of Different Centroid Seeds determines the number of re-clusterings that will be executed with different initial centroid seeds. In other words, the k-means algorithm will be executed Number of Different Centroid Seeds times all over again and the final result will be the best of the Number of Different Centroid Seeds executions.

kNN (k-Nearest Neighbor Classifier) is a supervised classification algorithm that classifies the data by calculating similarities among them. In kNN classification, a data point is classified by checking the closest k neighbors according to the selected distance metric such as Euclidean distance and Manhattan distance and selecting the label with highest frequency among the nearest k neighbors. In other words, the majority class in the found k nearest neighbors is used to label the new data point. In order to increase the speed of finding the k nearest neighbors of a given data point, a pre-processing of the data points into a KDTree is performed where the data space is divided into sub-sections in a hierarchical fashion and stored as leaves of the KDTree until each sub-section contains a limited number of data points. In applications of kNN on streaming data analysis, usually a finite window of samples is included in the process, meaning that a limited number of most recent samples are kept while older data points are discarded.

In ESTRA, Scikit-Multiflow library is used for kNN algorithm's implementation and several parameters can be tuned by the user [51].

- Number of Neighbors indicates the k value in kNN that is the number of neighbors to be considered when determining the label of a new data point.
- Maximum Window Size indicates the number of last observed samples that are kept in the data stream and used for classification.
- Leaf Size indicates the maximum number of instances each leaf of the KDTree can hold.

Discriminative Drift Detector (D3) is an unsupervised classification method developed for detecting concept drift with a discriminative classifier. D3 uses a fixed-size sliding window that works as a queue to analyze the data so that new data points received from a data stream are put at one end of the window and if the windows if full, the older data points are discarded from the other end, preserving the time order of the data samples inside the window. The size of the sliding window is set to $\omega + \omega \times \rho$ where ω is the parameter that determines how many data points are considered as old in the window and ρ is the parameter that determines the ratio of new data samples to old data samples $\rho < 1$. According to this definition, the data samples in the window are partitioned into two: the oldest ω samples in the window are labeled as old, and the newest $\omega \times \rho$ data samples are labeled as new. As the data samples are received from the data stream, the algorithm uses a simple discriminative classifier such as Hoeffding Tree that is run over the window to obtain AUC score [52]. If the calculated AUC score is bigger than the predetermined threshold value (τ), then the algorithm determines that there is a drift in the window between the old data points and new data points and discards all the old data points in the window, while if the AUC score does not meet the threshold, then it's evaluated as there's no drift and the oldest $\omega \times \rho$ points are discarded. After all or a portion of the oldest data points in the window are removed depending on the outcome of discriminative classifier, new data points are obtained from the stream and included in the window, effectively sliding the window over the streaming data which is followed by the classifier used inside the window being updated accordingly [4]. Working principle of D3 is shown in Figure 5.2 where

the stream of data is analyzed in a window (the rectangle at the top) and data is classified as old (blue) and new (green). In the case of drift, where the old data and new data are clearly separable (1), the old data is completely discarded from the window and in the case of no drift (2) the old data and new data are not separable therefore the window slides as usual.

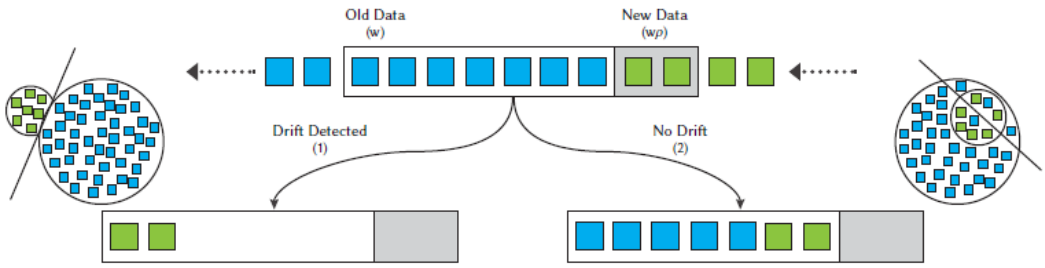


Figure 5.2: D3 Algorithm drift detection flow (adapted from [4]).

The implementation of Gözüaçık et al. [4] is adapted in ESTRA, and various parameters can be tuned by the users.

- ρ is the percentage of the new data with respect to old data.
- ω uses as a size of the old data.
- τ is the threshold value for the AUC score and it is between 0 and 1.

CluStream is a clustering algorithm developed for large drifting data streams, with the aim of being able to utilize older data points in a stream more than once in the evaluation of newer data points to get more accurate detection of drifts. CluStream works in two phases: online micro clustering and offline macro clustering. In the online micro clustering phase, synopsis of the data is collected, summarized and stored in predefined constant number (m) of micro clusters. Then, these micro clusters are used in the offline macro clustering phase which analyzes the summary of data and generates larger clusters. The online phase satisfies the one-pass data constraint and in the offline phase, the stored data can be used more than once for better clustering results. Thus, offline phase is not restricted by the one-pass constraint. In the offline macro clustering phase, CluStream uses the modified k-means algorithm for grouping the micro clusters within a specified time interval (horizon) obtained in the

previous phase into macro clusters. In this modified k-means method, the micro clusters obtained in the online phase are treated as pseudo-points at the centroids of the corresponding micro clusters with distance and seed adjustment calculations are performed according to the these centroids and initial centroid seeds are sampled in a probabilistic manner where the probability of each pseudo-point is proportional to the number of actual data samples stored in the corresponding micro cluster [53, 54]. Figure 5.3 shows the working principle of the two phase clustering approach that CluStream [53] algorithm is based on. Briefly, in the online phase, micro-clusters are created from data streams, and then in the offline phase, the micro-clusters are turned into macro-cluster with a clustering algorithm.

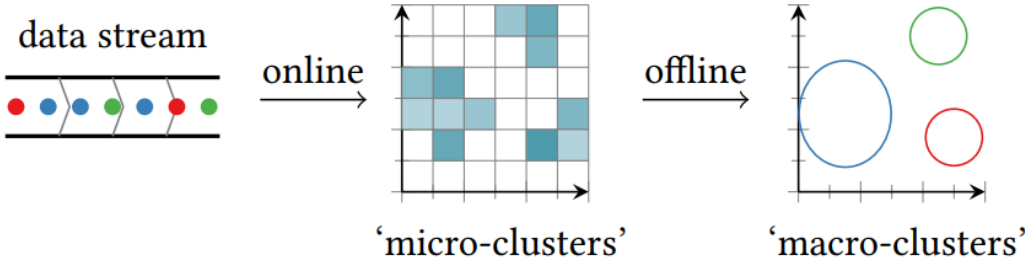


Figure 5.3: Two phase stream clustering approach (adapted from [5]).

Implementation of CluStream by Zubaroğlu et.al [55] is adapted in ESTRA, and various parameters can be tuned by the user.

- Time horizon (horizon) is used for determining the amount of history which is used in the offline macro clustering phase. The micro clusters obtained in this period will be used in the macro clustering.
- k holds the number of macro clusters to be created.
- m is used for the maximum number of micro clusters.

DenStream is a density-based clustering algorithm developed for evolving data streams with noise. Similar to CluStream, DenStream also works in two phases: the online phase of micro clustering and the offline phase of generating the clusters. In the online phase; potential micro clusters (p-micro-clusters) and outlier micro clusters(o-micro-clusters) are created. In this phase, as data samples arrive, the algorithm attempts to

merge potential micro clusters. If this is not possible, then the algorithm merges the arriving data with the closest outlier micro clusters. If the density of a outlier micro cluster is increased sufficiently so that it is larger than a given threshold, then it becomes a potential micro cluster. However, if the density of an outlier micro cluster stays under the threshold, then it is deleted with a pruning strategy after a certain time in order to prevent wasting memory. In the offline phase; the DBSCAN algorithm is applied to the potential micro clusters, in order to find the final clustering results and as a result, the core micro clusters (c-micro-clusters) are created. DenStream uses a damped window model in which each data point has a weight representing their recency so that as the data gets older, their weights are decreased. DenStream searches for arbitrary shaped clusters in the data unlike many clustering algorithms. This is performed by marking two clusters as reachable, if their centers are closer than a given value $(2 * \epsilon)$ and connecting the micro clusters that are reachable into a larger cluster. This approach provides more realistic results because of the nature of the streaming real data [10, 54].

Implementation of Zubaroğlu et.al [55] for DenStream is adapted in ESTRA, and various parameters can be tuned by the user.

- k denotes the number of core micro clusters.
- (ϵ) defines the radius of density area of a micro cluster.
- (β) is used for deciding whether an outlier micro cluster should be promoted to a potential micro cluster.

StreamKM++ is a clustering algorithm based on k-means++ [56]. First, a subset of the whole dataset is selected by using a probabilistic approach depending on the squared distance of a data point to the already selected points, which empirically allows a smaller subset of data to accurately represent the whole dataset. In order to improve the dataset selection speed, the StreamKM++ algorithm uses a special binary tree variation called coreset tree which divides data into two sub-clusters until the number of clusters reaches the predefined cluster count. The coreset clusters' centers are found with k-means++ which is a modified version of k-means that picks the initial centroid seeds in a probabilistic fashion. In k-means++, after the first centroid

seed is randomly selected from the data points, the consecutive centroid seeds are selected one by one from the data points with probability of each data point proportional to its distance to the nearest previously selected centroid seed. This approach increases the chances of initial centroid seeds to be in different clusters at the beginning. After applying k-means++ to find the centroid seeds, to find the final clusters, the Euclidean k-means method is applied to the coreset. StreamKM++ has some disadvantages: it is designed for neither evolving data streams nor datasets that contain outliers. Similar to most of the other clustering algorithms, StreamKM++ needs the number of cluster given before execution.

MOA [12] libraries are used for StreamKM++ algorithm implementation in ESTRA. A few parameters can be tuned by the user.

- Cluster Count represents the number of clusters that generated with the algorithm.
- Size Coreset holds the size of the coreset that is the subset of the data to represent the whole dataset and used during the clustering phase.

Hoeffding Tree is a stream classifier. It is an incremental and very fast decision tree (VFDT) algorithm assumes that data distribution does not change over time. This algorithm is created to overcome the classic decision tree algorithms' limitations such as memory and time. In Hoeffding Tree algorithm, similar to classic decision tree structure interim nodes contain a test criterion according to one attribute, effectively splitting the data space, and leaves corresponds to a class. As a stream classifier, Hoeffding tree is incrementally constructed, starting with a empty root and the tree is grown as more data samples are received. The distinctive feature of The Hoeffding Tree algorithm is to be able to guarantee performance bounds by limiting the number of samples and memory used by the tree and this is achieved by leaves storing statistics about the data samples being observed by the leaf. In other words, as data samples are received, the samples are sorted in the tree to assign a label but also the stored statistics of the corresponding leaf node are updated. Each leaf node keeps a counter for each attribute value and the associated label and as data points are sorted through the tree, a split evaluation function is calculated with the updated statistics. This split criterion represents the prepotency of an attribute being high enough by

using an adaptive Naive Bayes classification. When the split criterion in a leaf is met, then the leaf is converted into an interim node with the prepotent attribute and two new leaves are generated, inheriting the statistics from their ancestor. When the two attributes have similar gains, causing split criterion not being met, then the leaf is split if the Hoeffding's bound is below a given threshold. With this approach, Hoeffding Tree guarantees use of finite amount of memory with infinite amount of streamed data [57].

In ESTRA, Scikit-Multiflow [37] library is used for implementing the Hoeffding Tree Algorithm, and various parameters can be tuned by the users for optimizing the Hoeffding Tree [57] results.

- Grace Period indicates the number of data points a leaf should receive before the split decision.
- Naive Bayes Threshold indicates the number of data points a leaf should receive before allowing Naive Bayes is used.
- Tie Threshold holds the threshold value for splitting.

5.3 Datasets

In this section, the two types of datasets available in ESTRA are detailed which are built-in datasets that are fixed in size and number of feature; and data stream generators that are configurable and generate the data at the time of job execution.

5.3.1 Built-in Datasets

This section provides details on the built-in datasets available in ESTRA which are taken from public domain and various other studies. In contrast to the generated datasets mentioned in the previous section, the datasets explained here are stored as files in the system and synthetically streamed during the learning jobs initiated by the users. Three of the six available built-in datasets, KDD Cup'99, ELECTRICITY and COVERTYPE, contain data about real-world situations while the other three datasets

are artificially constructed to be used for drift detection exercises. All of these datasets are labeled and the sum of the datasets are shown in Table 5.3.

KDD CUP'99 [58] is used to build network intrusion detection. This is an unlabeled dataset and it is to be used for unsupervised learning algorithms such as k-means. It had been used in The Third International Knowledge Discovery and Data Mining Tools Competition. It contains 42 features and 494021 samples. Each data point in the dataset labeled as Normal or a type of Attack (22 types of attack exists such as Smurf, Teardrop, Buffer overflow, Neptune, Spy, etc.) [59]. The KDD CUP'99 dataset in ESTRA contains the continuous 34 features instead of all 42 features as in many studies utilizes this subset such as [10].

ELECTRICITY [58] is a labelled dataset to identify whether the electricity prices that changes by demand will decrease or increase. The data is sampled from the Australian New South Wales Electricity Market and contains the electricity price and demand of the source state as well as the neighboring Victoria state, timestamp of the data, day of the week, the time period in the day and the scheduled electricity transfer between the states. Each data point represents a 30 minute interval over a 2.5 year period and the label indicated whether the price increases or decreases compared to the average price of the past 24 hours window [60].

COVERTYPE [58] is a labelled dataset to predict forest cover type in the Roosevelt National Forest in Colorado, USA. It consist of 54 features about the cartographic variables of the forest area and soil characteristics, and 581012 samples in 30x30 meter resolution [58].

SYNTHESISED DATASET-1 is a labeled artificial dataset created for drift detection by Zubaroğlu et.al [61]. This dataset is generated with MOA's Random RBF Generator [33] method with 10 classes, 50 dimensions, 50000 samples, 0.02 average radius of cluster, and drift speed set to 100 as parameter values.

SYNTHESISED DATASET-2 is a labeled artificial dataset created for drift detection by Zubaroğlu et.al [61]. This dataset is generated with MOA's Random RBF Generator [33] method with 10 classes, 10 dimensions, 50000 samples, 0.02 average radius of cluster, and drift speed set to 100 as parameter values.

SYNTHESISED DATASET-3 is a labeled artificial dataset created for drift detection by Zubaroğlu et.al [61]. This dataset is generated with MOA's Random RBF Generator [33] method with 10 classes, 50 dimensions, 50000 samples, and 0.02 average radius of cluster as parameter values. This dataset does not contain drift unlike previous synthesised datasets.

Table 5.3: Properties of the datasets in ESTRA.

Dataset Name	Total Dimension / Used Dimension	# of Classes	Sample Size	Real / Synthesised
KDD Cup'99	42 / 34	23	494.021	Real
Electricity	8	2	45.312	Real
Covertype	54	7	581.012	Real
Synthesised Dataset-1	50	10	50.000	Synthesised
Synthesised Dataset-2	10	10	50.000	Synthesised
Synthesised Dataset-3	50	10	50.000	Synthesised

5.3.2 Stream Generators

This section contains information about data stream generators included in ESTRA that are Sea Data Generator and Hyperplane Generator. For these generators Scikit-Multiflow libraries [62] are used.

Sea Data Generator generates streamed data which contains abrupt concept drift for classifications [62]. It generates 50000 instances, 3 features, each sample may take a value between 0 and 10, and every 12500 instances there occur an abrupt drift. [63].

In the Sea Generator method that's used in ESTRA, there's one parameter that can be set by the user.

- Noise Percentage indicates the probability of the noise of the generated sample.

Hyperplane Generator generates datasets that fit into the hyperplane definition where every data point x satisfies $\sum_{i=1}^{d} \omega_i x_i = \omega_0$ where x_i represents a dimension of x and $\omega_0 ... \omega_i$ are constants defining the hyperplane. In the Hyperplane Generator, it's possi-

ble to introduce noise or drift to all or a subset of features. The drift is implemented as the incremental changes to the weights of the features being drifted with probabilistic reversals in the drift direction. It generates 200000 instances with given number of features, noise ratio and drift characteristics [62].

In the Hyperplane Generator method that's used in ESTRA, the following parameters can be set by the user.

- Feature Count holds the number of features that will be generated.
- Noise Percentage indicates the probability of the noise on the generated sample.
- Drifted Feature Count holds the number of features that will be drifted.
- Magnitude Change indicates the drift speed in the data.
- Sigma Percentage indicates the probability of the direction change in the data drift in each data point.

The available datasets, algorithms, and which dataset-algorithm pairs are supported for data analysis in ESTRA are summarized in Table 5.4.

Table 5.4: Usage of algorithms, dataset on the ESTRA

Algorithm \ Dataset	KDD Cup'99	Electricity	Covertype	Synthesised-1 (Drifted)	Synthesised-2 (Drifted)	Synthesised-1 (Stable)	Sea (Generator)	Hyperplane (Generator)
K-means	Yes	Yes	Yes	Yes	Yes	Yes	No	No
kNN	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
D3	Yes	Yes	Yes	Yes	Yes	Yes	No	No
CluStream	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
DenStream	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
StreamKM++	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Hoeffding Tree	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

CHAPTER 6

RESULTS AND DISCUSSION

In this chapter, several use cases under the framework of ESTRA are demonstrated by means of replicating the setups of various studies already reported in the literature. The results obtained with ESTRA are compared with those achieved in the respective studies. The purpose of this chapter is to verify that the algorithm implementations in ESTRA produce similar results with the studies performed in the streaming data analysis area and also to demonstrate ESTRA's value as a quick and easy prototyping tool which we believe, improves iteration and experimentation speed in streaming data analysis research.

ESTRA results shown in this chapter are obtained on a personal computer manufactured in 2011 with third generation Intel i7 mobile processor and 8 GB of RAM, running Linux Mint operating system with Linux kernel version 4.15.

6.1 Detecting Concept Drift With Discriminative Drift Detector (D3) Algorithm

Gözüaçık et al. [4] presented Discriminative Drift Detector (D3) that is an unsupervised concept drift detection algorithm for streaming data as detailed in Section 5.2. For experimental evaluation of their method, Gözüaçık et al. used 8 different datasets with drift, including Covertype and Electricity and combinations of values for different parameters of D3 algorithm: ρ = 100, 250, 500, 1000, 2500, ω = 0.1, 0.2, 0.3, 0.4, 0.5, τ = 0.60, 0.65, 0.70, 0.75, 0.80, 0.85, 0.90. As the results, they found the optimum values for these parameters as $\rho = 100$, $\omega = 0.1$, and $\tau = 0.70$.

When the same experiments are performed in ESTRA exactly with the same parame-

ter values of the study by Gözüaçık et al., the results were similar for Electricity and Covertype datasets [58]. Gözüaçık et al. obtained the average accuracy of 86.69%, and 87.17% on Electricity dataset and Covertype dataset, respectively with D3 algorithm. The corresponding accuracy values achieved with ESTRA are 84.57% and 86.26%, for the Electricity dataset and the Covertype dataset, respectively. The obtained values are summarized in Table 6.1. It must be noted that the obtained average accuracy values change with every execution of the algorithm with the same set up, because of the randomized nature of Hoeffding Tree's evaluation.

Table 6.1: Comparison of D3 algorithm's accuracy with that of ESTRA.

Dataset	Original Accuracy (%)	ESTRA's Accuracy (%)
Electricity	86.69	84.57
Covertype	87.17	86.26

On top of the repetitions of the experiments done by Gözüaçık et al. in their study, D3 in ESTRA has also been tested with additional datasets available in ESTRA: KDD Cup'99 and the three Synthesised Datasets. In these experiments, D3 achieved 99.93% accuracy on KDD Cup'99, 92.56% accuracy on Synthesised Dataset-1, 94.54% accuracy on Synthesised Dataset-2 and 89.54% accuracy in Synthesised Dataset-3. The results are show in Table 6.2.

Table 6.2: Accuracy values of D3 algorithm in ESTRA on other datasets.

Dataset	ESTRA's Accuracy (%)
KDD Cup'99	99.93
Synthesised Dataset-1	92.56
Synthesised Dataset-2	94.54
Synthesised Dataset-3	89.54

6.2 Generated Data Stream Classification With k-NN Algorithm

Bifet et al. run kNN algorithm on datasets generated by Hyperplane Generator and SEA Data Generator [62] measuring the accuracy and processing times. On Hyperplane Generator, they were able to obtain 83.33% accuracy in 223 seconds with kNN while ESTRA achieves 82.03% accuracy within 126 seconds. On the Sea Data Generator, Bifet et al. obtained 86.80% accuracy within 110 seconds with kNN while ESTRA achieves 98.90% accuracy within 125 seconds, as summarized in Table 6.3.

Results obtained in Bifet et al. and with ESTRA are close to each other in terms of accuracy while ESTRA runs faster. The accuracy and time differences can be due to differences in the used libraries and platforms. While Bifet et al. uses MOA for the dataset generators, kNN implementation; ESTRA uses Scikit-Multiflow for the generators and kNN classifier.

Table 6.3: Knn algorithm's accuracy and time comparison with ESTRA.

Stream Generator	kNN Original Accuracy (%) (Time s)	ESTRA's Accuracy (%) (Time s)
Hyperplane Generator	83.33 (223)	82.03 (126)
Sea Data GENERATOR	86.80 (110)	95.90 (125)

6.3 Data Classification with Hoeffding Tree Algorithm

While Bifet et al. [63] obtained the results of kNN, they also compared these results with Hoeffding Tree algorithm.

Bifet et al. run Hoeffding Tree algorithm on real datasets such as Electricity and Covertype [58] as well as synthetic datasets generated with Sea Data Generator and Hyperplane Generator [62]. In their experiments, Bifet et al. achieved 79.20% accuracy with 1 second runtime on Electricity dataset and 80.31% accuracy with 20 seconds of runtime on Covertype dataset. For the same experiment setups, ESTRA achieves 87.79% accuracy within 1 second on Electricity dataset and 61.66% accuracy in 7 seconds on Covertype dataset. On the synthetic datasets, Bifet et al.'s results were 78.77% accuracy in 10 seconds for Hyperplane Generator and 86.42% accuracy

in 5 seconds for SEA Data Generator while ESTRA achieves 87.96% accuracy within 8 seconds for Hyperplane Generator and 93.99% accuracy in 2 seconds for SEA Data Generator as can be seen in Table 6.4.

Similar to the results obtained with kNN, the accuracy and time differences here can also be due to differences in the used libraries and platforms as Bifet et al. uses MOA for the dataset generators and Hoeffding Tree implementation while ESTRA Scikit-Multiflow for them. It must also be noted that the working environments are not the same; therefore, comparing ESTRA in terms of performance may not be accurate.

Table 6.4: Accuracy and time comparison for Hoeffding Tree algorithm with ESTRA.

Dataset	Original Accuracy (%) (Time s)	ESTRA's Accuracy (%) (Time s) With Streaming
Electricity	79.20 (1)	87.79 (1)
Covertype	80.31 (20)	61.66 (7)
Hyperplane Generator	78.77 (10)	87.96 (8)
Sea Data Generator	86.42 (5)	93.99 (2)

6.4 Density Based Clustering with DenStream Algorithm

Cao et al. [10] introduced density based clustering algorithm DenStream that discovers clusters on evolving data streams with noise as detailed in Section 5.2. Cao et al. implemented DenStream algorithm with Microsoft Visual C++ and evaluated the algorithm with KDD Cup'99 dataset by using various metrics such as purity, memory usage, and execution time.

Cao et al. calculated the purity of the resulting clustering according to different outlier threshold values, however they didn't specify the exact values they got as the result of their experiment but presented the results in a line chart, therefore the numbers taken from their study are approximate numbers. They were able to obtain about 97% purity when the outlier threshold is 0.2 and similarly, ESTRA achieves 95.98% purity for the same outlier threshold value. Cao et al. obtained the purity approximately 96% when outlier threshold is 0.4 while ESTRA achieves 97.28%. Their results were

96%, 95% and 80% when corresponding outlier threshold values were 0.6, 0.8, and 1 respectively while the corresponding purity values achieved with ESTRA are 97.39% for all of the three outlier threshold values. The results are summarized in Table 6.5.

ESTRA's implementation gave more stable results than the original results for this example with the higher outlier threshold values. The reason of the observed differences between these results can be explained with implementation differences and possible improvements made since the original experiments in the last 10+ years.

Table 6.5: Purity over outlier threshold (β) in DenStream algorithm with ESTRA.

Outlier Threshold (β)	Original Purity (%)	ESTRA's Purity (%)
0.2	$\cong 97$	95.98
0.4	$\cong 96$	97.28
0.6	$\cong 96$	97.39
0.8	$\cong 95$	97.39
1	$\cong 80$	97.39

CHAPTER 7

CONCLUSIONS, LIMITATIONS AND FUTURE WORK

In this thesis, ESTRA, an easy-to-use, web-based, scalable, extensible, and open-source tool that provides streaming data analysis by using machine learning techniques is proposed. First, the well-known tools that are used for data streaming and stream analysis are described along with their advantages and disadvantages. Their challenges such as their dependencies, setup requirements, learning curves, and environmental limitations are given and in order to address these challenges, ESTRA is proposed as a web-based easy-to-use solution. ESTRA's scalable, extensible, and web-based architecture is then described and the usage of the ESTRA is demonstrated step by step. The diverse datasets, streaming data generators, algorithms, and quality metrics used in ESTRA are described in detail. Finally, some of the use cases reported in the literature are reproduced using ESTRA and the results obtained with ESTRA are compared with the original results in terms of various metrics such as accuracy, purity, and performance.

As the result of repeated experiments with ESTRA, it's demonstrated that ESTRA can achieve comparable accuracy results with the original studies. In Gözüaçık et al.'s studies for D3 on Electricity and Covertype datasets, they achieved 86.69% and 87.16% accuracy while D3 on ESTRA achieved similar accuracy numbers with 84.57% and 86.26% respectively. When D3 on ESTRA is run on some other datasets, achieved accuracy values were 99.93% on KDD Cup'99, 92.56% on Synthesised Dataset-1, 94.54% on Synthesised Dataset-2 and 89.54% on Synthesised Dataset-3. In Bifet et al.'s experiments on kNN algorithm on generated dataset with Hyperplane and Sea Data generators, they achieved 83.33% and 86.90% while with kNN on ESTRA the results achieved with are 82.03% and 95.90% respectively. In their experiments on Hoeffd-

ing Tree algorithm with Electricity, Covertype datasets as well as generated datasets by Hyperplane and Sea Data generators, they achieved 79.20%, 80.31%, 78.77%, 86.42% accuracy values respectively while Hoeffding Tree on ESTRA achieves 87.79%, 61.66%, 87.79% and 93.99% accuracy on the same datasets. Finally, in Cao et al.'s experiments on DenStream algorithm on KDD Cup'99 dataset with different outlier threshold values of 0.2, 0.4, 0.6, 0.8 and 1; they achieved purity values of about 97, 96, 96, 95 and 90 while the purity results obtained with ESTRA were more stable as 95.98%, 97.28%, 97.39%, 97.39% and 97.39%.

There are six datasets available in ESTRA; three of them are real datasets while the rest are artificially generated datasets. There are also two data stream generators, SEA and Hyperplane generators. As a future work, more datasets can be added as built-in datasets, and streaming data generators can be implemented for ESTRA to increase the diversity of the available datasets. Similarly, more algorithm implementations can be added to ESTRA. ESTRA currently comes with seven machine learning algorithms where six of them are specially used in streaming data analysis. Some candidates for new algorithms to be added in ESTRA can be ClusTree, D-Stream or HDDStream [47]. Moreover, adding more metrics such as sum of squared errors, silhouette index or rand index can be added easily as the ESTRA is an open-source software.

Apart from integrating more datasets with ESTRA, it would also be beneficial to provide ability to upload custom datasets of the users. By this way, the users can work with any dataset that is relevant to their studies and ESTRA can be used in a broader area of research. This feature would also require making proper security, access control and rate limiting features to be included as data upload is a compute, network and storage intensive operation thus can be used for denial of service attacks.

As ESTRA is open source, developers and researchers can integrate other algorithm implementations with ESTRA. However, currently, there's no clear guide on how to do that and the integration should happen deep inside the code. Most of the modern extensible software provide extension application programming interfaces (API), providing a clear and straightforward way to be support third party additions to the code without requiring extension developers to make changes in the core parts of the

software. As a future work, a similar approach for supporting additional algorithm implementations can be pursued for ESTRA as well.

One of the most important areas of improvement in ESTRA is to provide solutions for security and privacy issues. Currently, ESTRA does not incorporate any features for establishing privacy of the users or rate limiting. This makes ESTRA vulnerable to be released as a public software system as anyone can see and delete others' work and also perform denial of service attacks. In order to address these problems, a user management system can be implemented. Depending on the deployment environment, the user management system can be a common email and password type user registration system as well as an embedded user management service such as Microsoft Active Directory.

In order to take a step forward and go outside of being an experimental suite and to be able to also satisfy business workloads, the ability to integrate ESTRA with real data streams can be added. This feature would allow ESTRA to consume live data streams and process them in real time, thus making it an end-to-end data stream analysis platform. Furthermore, also the ability to export information about the analysis results to outside sources can be also included, making ESTRA a candidate for any data stream analysis pipeline for any business needs.

ESTRA has been developed with the focus on ease of use and simplicity. However, usability, accessibility and user interface consistency aspects can still be improved with the touch of user interface and user experience experts.

Finally, even though ESTRA is open source and can be downloaded and run by anyone easily, it's not ready to be made publicly available as a service yet, mainly due to the lack of access control features. After security and access control aspects are added, ESTRA can be provided as a service to the community and be developed as a community-driven product.

REFERENCES

[1] S. Visa, B. Ramsay, A. L. Ralescu, and E. Van Der Knaap, "Confusion matrix-based feature selection.," *MAICS*, vol. 710, pp. 120–127, 2011.

[2] J. Jenkov, "Micro batching url:http://tutorials.jenkov.com/java-performance/micro-batching.html," Apr 2016. [Online; accessed 2021-09-01].

[3] A. J. Gates and Y.-Y. Ahn, "The impact of random models on clustering similarity," *arXiv preprint arXiv:1701.06508*, 2017.

[4] Ö. Gözüaçık, A. Büyükçakır, H. Bonab, and F. Can, "Unsupervised concept drift detection with a discriminative classifier," in *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pp. 2365–2368, 2019.

[5] M. Carnein, D. Assenmacher, and H. Trautmann, "An empirical comparison of stream clustering algorithms," in *Proceedings of the Computing Frontiers Conference*, CF'17, (New York, NY, USA), p. 361–366, Association for Computing Machinery, 2017.

[6] Q. Zhang, Z. Chen, and L. T. Yang, "A nodes scheduling model based on markov chain prediction for big streaming data analysis," *International Journal of Communication Systems*, vol. 28, no. 9, pp. 1610–1619, 2015.

[7] T. Kolajo, O. Daramola, and A. Adebiyi, "Big data stream analysis: a systematic literature review," *Journal of Big Data*, vol. 6, no. 1, p. 47, 2019.

[8] L. Golab and M. T. Özsu, "Issues in data stream management," *ACM Sigmod Record*, vol. 32, no. 2, pp. 5–14, 2003.

[9] C. P. Chen and C.-Y. Zhang, "Data-intensive applications, challenges, techniques and technologies: A survey on big data," *Information sciences*, vol. 275, pp. 314–347, 2014.

[10] F. Cao, M. Estert, W. Qian, and A. Zhou, "Density-based clustering over an evolving data stream with noise," in *Proceedings of the 2006 SIAM international conference on data mining*, pp. 328–339, SIAM, 2006.

[11] W. Inoubli, S. Aridhi, H. Mezni, M. Maddouri, and E. Nguifo, "A comparative study on streaming frameworks for big data," in *VLDB 2018-44th International Conference on Very Large Data Bases: Workshop LADaS-Latin American Data Science*, pp. 1–8, 2018.

[12] G. Holmes, R. Kirkby, and B. Pfahringer, "Moa: massive online analysis url:http://sourceforge.net/projects/moa-datastream," 2007. [Online; accessed 2020-09-07].

[13] S. R. Garner *et al.*, "Weka: The waikato environment for knowledge analysis," in *Proceedings of the New Zealand computer science research students conference*, vol. 1995, pp. 57–64, 1995.

[14] Y. Zhao, *R and data mining: Examples and case studies*. Academic Press, 2012.

[15] "Publish/subscribe url:https://www.ibm.com/support/knowledgecenter/," Sep 2020. [Online; accessed 2020-10-13].

[16] E. B. I. B. J. L. Myers and J. L. Myers, "Introduction to streaming data platforms url:https://www.infoworld.com/article/3095868/introduction-to-streaming-data-platforms.html," Jul 2016. [Online; accessed 2020-09-07].

[17] P. Le Noac'h, A. Costan, and L. Bougé, "A performance evaluation of apache kafka in support of big data streaming applications," in *2017 IEEE International Conference on Big Data (Big Data)*, pp. 4803–4806, 2017.

[18] A. Kafka, "A high-throughput distributed messaging system," *URL: kafka. apache. org as of*, vol. 5, no. 1, 2014.

[19] P. Dobbelaere and K. S. Esmaili, "Kafka versus rabbitmq: A comparative study of two industry reference publish/subscribe implementations: Industry paper," in *Proceedings of the 11th ACM international conference on distributed and event-based systems*, pp. 227–238, 2017.

[20] V. M. Ionescu, "The analysis of the performance of rabbitmq and activemq," in *2015 14th RoEduNet International Conference-Networking in Education and Research (RoEduNet NER)*, pp. 132–137, IEEE, 2015.

[21] B. Snyder, D. Bosanac, and R. Davies, "Introduction to apache activemq," *Active MQ in action*, pp. 6–16, 2017.

[22] J. Karimov, T. Rabl, A. Katsifodimos, R. Samarev, H. Heiskanen, and V. Markl, "Benchmarking distributed stream data processing systems," in *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, pp. 1507–1518, IEEE, 2018.

[23] S. Chintapalli, D. Dagit, B. Evans, R. Farivar, T. Graves, M. Holderbaugh, Z. Liu, K. Nusbaum, K. Patil, B. J. Peng, and P. Poulosky, "Benchmarking streaming computation engines: Storm, flink and spark streaming," in *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pp. 1789–1792, 2016.

[24] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen, *et al.*, "Mllib: Machine learning in apache spark," *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 1235–1241, 2016.

[25] A. S. Apache, "Apache storm url:https://storm.apache.org/documentation," 2015. [Online; accessed 2020-10-10].

[26] J. S. v. d. Veen, B. v. d. Waaij, E. Lazovik, W. Wijbrandi, and R. J. Meijer, "Dynamically scaling apache storm for the analysis of streaming data," in *2015 IEEE First International Conference on Big Data Computing Service and Applications*, pp. 154–161, 2015.

[27] G. D. F. Morales and A. Bifet, "Samoa: scalable advanced massive online analysis.," *J. Mach. Learn. Res.*, vol. 16, no. 1, pp. 149–153, 2015.

[28] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, and K. Tzoumas, "Apache flink: Stream and batch processing in a single engine," *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, vol. 36, no. 4, 2015.

[29] "Flinkml - machine learning for flink url:https://ci.apache.org/projects/flink/flink-docs-release-1.12/." [Online; accessed 2021-01-01].

[30] "Introduction to azure stream analytics url:https://docs.microsoft.com/en-gb/azure/stream-analytics/stream-analytics-introduction," 2020. [Online; accessed 2020-06-13].

[31] G. Holmes, A. Donkin, and I. H. Witten, "Weka: A machine learning workbench," in *Proceedings of ANZIIS'94-Australian New Zealnd Intelligent Information Systems Conference*, pp. 357–361, IEEE, 1994.

[32] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The weka data mining software: an update," *ACM SIGKDD explorations newsletter*, vol. 11, no. 1, pp. 10–18, 2009.

[33] A. Bifet, G. Holmes, B. Pfahringer, P. Kranen, H. Kremer, T. Jansen, and T. Seidl, "Moa: Massive online analysis, a framework for stream classification and clustering," in *Proceedings of the First Workshop on Applications of Pattern Analysis*, pp. 44–50, PMLR, 2010.

[34] P. Kranen, H. Kremer, T. Jansen, T. Seidl, A. Bifet, G. Holmes, B. Pfahringer, and J. Read, "Stream data mining using the moa framework," in *International Conference on Database Systems for Advanced Applications*, pp. 309–313, Springer, 2012.

[35] R. Tönjes, P. Barnaghi, M. Ali, A. Mileo, M. Hauswirth, F. Ganz, S. Ganea, B. Kjærgaard, D. Kuemper, S. Nechifor, *et al.*, "Real time iot stream processing and large-scale data analytics for smart city applications," in *poster session, European Conference on Networks and Communications*, sn, 2014.

[36] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, *et al.*, "Scikit-learn: Machine learning in python," *the Journal of machine Learning research*, vol. 12, pp. 2825–2830, 2011.

[37] J. Montiel, J. Read, A. Bifet, and T. Abdessalem, "Scikit-multiflow: A multi-output streaming framework," *The Journal of Machine Learning Research*, vol. 19, no. 1, pp. 2915–2914, 2018.

[38] "The r project for statistical computing url:https://www.r-project.org/." [Online; accessed 2020-09-01].

[39] "R documentation and manuals url:https://www.rdocumentation.org/," 2020. [Online; accessed 2020-09-13].

[40] H. S. Oluwatosin, "Client-server model," *IOSR J Comput Eng (IOSR-JCE)*, vol. 16, no. 1, p. 67, 2014.

[41] M. A. Jadhav, B. R. Sawant, and A. Deshmukh, "Single page application using angularjs," *International Journal of Computer Science and Information Technologies*, vol. 6, no. 3, pp. 2876–2879, 2015.

[42] "Ui: A popular react ui framework url:https://material-ui.com/." [Online; accessed 2020-12-13].

[43] "Recharts url:https://recharts.org/en-us/." [Online; accessed 2020-12-07].

[44] P. G. D. Group, "Postgresql: The world's most advanced open source relational database url:https://www.postgresql.org/," Jan 2021. [Online; accessed 2021-01-21].

[45] S. Sharma, "Fermi estimates on postgres performance url:https://www.citusdata.com/blog/2017/09/29/what-performance-can-you-expect-from-postgres/," Sep 2017. [Online; accessed 2021-01-03].

[46] C. D. Manning, P. Raghavan, and H. Schutze, "Introduction to information retrieval," *Ch*, vol. 20, pp. 405–416, 2008.

[47] S. Mansalis, E. Ntoutsi, N. Pelekis, and Y. Theodoridis, "An evaluation of data stream clustering algorithms," *Statistical Analysis and Data Mining: The ASA Data Science Journal*, vol. 11, no. 4, pp. 167–187, 2018.

[48] M. J. Warrens, "On the equivalence of cohen's kappa and the hubert-arabie adjusted rand index," *Journal of classification*, vol. 25, no. 2, pp. 177–183, 2008.

[49] A. K. Jain, "Data clustering: 50 years beyond k-means," *Pattern recognition letters*, vol. 31, no. 8, pp. 651–666, 2010.

[50] D. Puschmann, P. Barnaghi, and R. Tafazolli, "Adaptive clustering for dynamic iot data streams," *IEEE Internet of Things Journal*, vol. 4, no. 1, pp. 64–74, 2016.

[51] O. Kramer, "Scikit-learn," in *Machine learning for evolution strategies*, pp. 45–53, Springer, 2016.

[52] T. Fawcett, "An introduction to roc analysis," *Pattern recognition letters*, vol. 27, no. 8, pp. 861–874, 2006.

[53] C. C. Aggarwal, S. Y. Philip, J. Han, and J. Wang, "A framework for clustering evolving data streams," in *Proceedings 2003 VLDB conference*, pp. 81–92, Elsevier, 2003.

[54] A. Bifet, R. Gavaldà, G. Holmes, and B. Pfahringer, *Machine learning for data streams: with practical examples in MOA*. MIT Press, 2018.

[55] A. Zubaroğlu, "Unpublished code." Personal Communication.

[56] D. Arthur and S. Vassilvitskii, "k-means++: The advantages of careful seeding," tech. rep., Stanford, 2006.

[57] P. Domingos and G. Hulten, "Mining high-speed data streams," in *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 71–80, 2000.

[58] D. Dua and C. Graff, "Uci machine learning repository url:http://archive.ics.uci.edu/ml," 2017. [Online; accessed 2020-12-20].

[59] K. Cup, "Kdd cup99 http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html," *The UCI KDD Archive*, 1999.

[60] J. Vanschoren, "Electricity dataset url:https://www.openml.org/d/151," Apr 2014. [Online; accessed 2020-09-28].

[61] A. Zubaroğlu, "Dataset files url:https://gitlab.com/alaettinzubaroglu/phd-studies/," Jan 2020. [Online; accessed 2021-01-10].

[62] "Api reference url:https://scikit-multiflow.readthedocs.io/en/stable/api/api.html." [Online; accessed 2020-09-01].

[63] A. Bifet, B. Pfahringer, J. Read, and G. Holmes, "Efficient data stream classification via probabilistic adaptive windows," in *Proceedings of the 28th annual ACM symposium on applied computing*, pp. 801–806, 2013.