

Institute of Parallel and Distributed Systems

Masters Thesis

An Analytics Framework for the IoT Platform MBP

Abhishek Kumar

Course of Study: Computer Science

Examiner: PD Dr. rer. nat. habil. Holger Schwarz

Supervisor: Dr. rer. nat. Pascal Hirmer

Commenced: October 28, 2019

Completed: April 28, 2020

Abstract

The emergence of IoT has introduced a huge amount of applications that generate massive amounts of data at a high rate. This data stream needs intelligent data processing and analysis. The evolution of Smart cities and Smart industries has resulted into an ocean of data from millions of sensors and devices. Surveillance systems, telecommunication systems, smart devices, and smart cars are some examples of such systems. However, this data itself does not provide any information unless it is analyzed. This results into a need of analytics tools and frameworks which can efficiently analyze this data and provide with useful information. Analytics is all about inspection, transformation and modelling of data to achieve information that further suggests and assists in decision making. In a world of IoT, analytics has a crucial role to play to improve life and better manage the infrastructure in a secure, sustainable and cost effective manner. The smart sensor network serves as the base for IoT. In this context, one of the major tasks is to develop advanced analytics frameworks for the interpretation of data provided by the sensors. MBP is a platform for managing IoT environments. Sensors and devices can be registered to this platform and the status of sensors can be viewed and modified from the platform. This platform is used to collect data from the sensors and devices connected to the platform. There are two types of mining that can be performed on raw data, one technique analyzes the data on the fly as it is received (Data Stream Mining) and the other can be performed on demand on the data collected for a longer period of time (Batch Processing). Both types of analysis has its own advantages. Lambda architecture is a data analytics architecture which allows us to perform both stream analysis and batch processing on the same data. This architecture defines some practical and well versed principles of handling big data. The pattern allows us to deal with both real time and historical data, but the analysis is performed separately and does not affect each other. In this thesis, we will create an analytics framework for the MBP IoT platform based on the lambda architecture.

Contents

1	Introduction	9
1.1	Importance of data mining in IoT:	10
2	Fundamentals	13
2.1	Batch processing of data:	13
2.2	Data Stream mining:	14
2.3	Lambda Architecture:	14
2.4	MBP Platform:	16
3	Related Work	19
3.1	Lambda architecture in Amazon Web Services (AWS):	20
3.2	Microsoft Azure IoT analytics:	20
3.3	Analytics in Industrial Internet of Things (IIoT):	22
3.4	Energy Management in Smart Homes using IoT Analytics:	23
4	Technologies Used	25
4.1	Apache Spark (Batch Processing engine)	25
4.2	Scikit-Multiflow (Stream Processing engine)	26
4.3	Python Flask (enabling REST APIs):	28
4.4	MQTT (Message Protocol):	28
5	System Architecture	31
5.1	Basic Blocks of the System:	31
5.2	Technologies and Communication with MBP:	32
5.3	Sequence Diagram:	33
6	Implementation	35
6.1	Implementation of functionalities on MBP user interface:	35
6.2	Implementation of functionalities on the analytics server:	43
6.3	Algorithms available on the analytics framework:	43
6.4	Storing Analytics Models:	49
7	Validation and Discussion	51
7.1	Modeling energy usage in Smart Cities:	51
7.2	Predictive maintenance in Smart Industries:	53
7.3	Descriptive analysis of components in Smart Industries:	54

8	Conclusion	57
9	Future Work	61

List of Figures

2.1	Lambda Architecture	15
2.2	MBP Dashboard showing list of connected devices	16
2.3	MBP Dashboard showing historical data values from sensors	17
3.1	Increase in number of Connected Devices [Ali15]	19
3.2	Implementation of Lambda Architecture in AWS [Raj18]	20
3.3	Microsoft Azure Stream Analytics framework [Kle17]	21
3.4	Analytics in Industrial IoT [LGS17]	23
3.5	Sequence diagram for collection and processing of data [AZR+17]	24
4.1	Performance of Spark compared to other tools [ZXW+16]	26
4.2	Architecture of MQTT [HTS08]	29
5.1	Basic blocks of the analytics framework	31
5.2	Technologies used in MBP and the analytics framework	32
5.3	Sequence diagram showing sequence of commands execution	33
6.1	New Analytics menu item on MBP user interface	35
6.2	Form to create analytics model	36
6.3	Sample input to create a batch processing model	37
6.4	Sample input to create a stream mining model	38
6.5	Access an existing model	39
6.6	View Statistics of selected model	39
6.7	Form to sent value to make prediction	40
6.8	Predicted result based on input values	41
6.9	Delete button to delete the model	42
6.10	Confirmation box to delete the selected model	42
6.11	Minimization and Cost function	44
7.1	Predicted Electricity consumption value	52
7.2	Predicted Status of the device	54
7.3	Finding frequent patterns between different parameters	55
8.1	Total number of connected devices with share of IoT devices [Lue20]	58

1 Introduction

IoT is at the cornerstone of the digital transformation journey for most of the fields [TAG+17]. It enables users to collect useful data from the connected devices. However, connecting devices and ingesting and storing data is just the first step. Being connected is not enough, beyond that, the general objects in IoT should have the capabilities to learn, think and understand the devices, which surround them. The whole point of collecting this data is to extract actionable insights, which can trigger some sort of action that will result in business value.

In the next few decades, many traditional cities will be turned to smart cities [AB18], in order to make them greener, smarter, and more efficient. One key challenge for developing smart cities is integrating different application domains. This transformation is supported by the fast spread of Internet of Things (IoT) and big data analysis. ICT (Information and Communication Technology) is becoming increasingly pervasive to urban environments, providing the necessary basis for sustainability and resilience of smart future cities. Sensors are by no means a new phenomenon, the first sensor was invented way back in the 19th century. The revolutionary aspect of IoT lies in its recent adoption on an unprecedented scale, fuelled by economic factors, such as dramatic drop in prices of sensors, network bandwidth, and processing. Moreover, IoT allows data to be captured and ingested autonomously, avoiding the human data entry bottleneck. The number of connected devices is increasing exponentially. Just to give an idea, the number of connected IoT devices in 2015 was around 15 billion, currently in 2019, it is 27 billion and is expected to grow to about 75 billion till 2025 [Ali15]. IoT data will arguably become the biggest Big Data.

The question that arises is, how to make effective use of this vast ocean of data? The future IoT will be highly populated by a large number of heterogeneous networked devices, which will generate massive amounts of data. This pool of data would be useless if we do not extract insights out of it. This requires developing a systematic analytics framework for IoT. A key challenge in smart city applications is aggregation and processing of streaming information from various domains. A large amount of valuable data remains unused or limited to specific application domains due to a large number of specific technologies and formats, and effective adaptive stream processing of data is still a hard task. The real value of data is gained by new knowledge acquired by performing data analytics using various data mining, machine learning, or statistical methods [DWW13].

Extracting insights from IoT data is essentially a big data analytics problem. It is about analyzing lots of data, coming in fast from a lot of different sources and possibly in different formats. This stream of data from IoT devices has its own set of challenges:

- Data comes from “things”, therefore, data collection has its own challenges, like establishing reliable, performant, and secure connection with the device, overcoming latency issues etc.
- IoT data is almost real time, streamed time series data coming in at different frequencies.
- IoT data often requires real time decisions on data, which often requires some processing to be done on or near to the source to avoid delay of sending data and waiting for a response.
- In practical IoT applications, the obtained massive sensor data can be of mixed characteristics, which is challenging to process.

1.1 Importance of data mining in IoT:

Data mining is the process of gaining insights from data [Han98]. The nature of IoT applications demands real time responses. For example, in the traffic domain, one might want to plan a travel route according to the current traffic conditions, and in smart homes one might want to receive timely alerts about unusual patterns of electricity consumption [TBA+14]. Therefore, real time analysis is of high importance. This brings the requirement of data stream mining where data is analyzed on the fly as it is collected from the devices. There are a set of challenges, which are posed by the high-velocity data streams in the IoT. However, the importance of collecting and analyzing historical IoT data is less immediately apparent. In addition, because of the size of data, this analysis is costly. Furthermore, analysis of long-term data (batch processing) has its own importance. Batch processing is essential in order to reach intelligent decisions, it helps in understanding the context of real time data. For example, one can get the current traffic condition from real time analysis, but does the traffic condition represent normal condition, or an extreme traffic congestion? Does a sudden increase in home energy consumption result from heating in cold weather, or a faulty device? Answers like these can be obtained by the analysis of historical sensor data.

Now, we know the importance of both data stream analysis and batch processing of IoT data. In this thesis, we will use the Lambda architecture for development of an analytics framework for the MBP IoT platform. MBP is a platform for managing IoT devices. Sensors and devices can be registered to this platform and the status of sensors can be viewed and modified from the platform. Lambda architecture is a data processing design pattern to handle massive amounts of data and integrate batch processing and stream processing of data into a single framework, which makes the combined data available for downstream analysis or viewing on a service layer. The MBP platform and Lambda architecture will be discussed in detail in the upcoming sections. Following are the goals of the thesis:

- Design an analytics framework for the MBP IoT platform. This design will be based on the lambda architecture. Hence, provisioning for both continuous stream mining and on demand batch processing of data should be done. Live data from devices should be fed directly for stream processing and should also be stored for batch processing.
- Develop the framework by using open-source tools, such as Apache Spark and Flink for data analysis and integrating it to the MBP platform. The detailed description of these tools and implementation will be covered in later sections.
- Demonstrate the feasibility and use of the framework by presenting real world use cases of analytics in IoT and perform analysis on datasets to show the results.

2 Fundamentals

The goal of this thesis is to create an analytics framework for the IoT platform MBP. The framework will be based on the Lambda architecture for data analytics. The advantage of using a Lambda Architecture is enabling both real time analysis (data stream processing) and on demand analysis of long-term data (batch processing) based on the same data source. We will be discussing these topics in detail in this section.

2.1 Batch processing of data:

Batch processing (also known as data mining) is a useful decision support technique, which can be used to find trend and regularities in data. It aims at discovery of useful patterns and trends in large sets of data [Han06]. We are observing evolution of data mining systems from specialized tools to multi-purpose data mining systems offering a wide range of integration of data management systems. From a user's point of view, data mining can be viewed as advanced querying of data. The basic motivating stimulus behind data mining is that the large databases contain information, which is of value to the database owner, but the information is concealed within the mass of uninteresting data and has to be discovered. The user seeks surprising, unexpected or valuable information and data mining aims to extract this information. Perhaps the main economic driver to the development of data mining tools and techniques has come from the commercial world: the promise of money to be made from data analytics. Furthermore, commercial databases are growing rapidly in size as well as number. Data is produced in IoT at an unimaginable speed. Data mining is a rapidly evolving technology and an increasing number of datasets are now collected with the specific objective of trawling through them and seeking interesting and unusual information. Batch processing is where the processing happens over a block of data stored over a period of time. For example, processing all the transactions that have been performed by a firm in a week. Batch processing usually takes a large amount of time and lots of computing resources. Batch processing works well in situations where one does not need real time analytics results and when it is more important to process data in large volume to get more detailed insights than it is to get fast analytics results. If real time processing of data is required, e.g. in the IoT, data stream processing is more suitable than batch processing.

2.2 Data Stream mining:

Data stream processing is the answer analytics results should be computed in real time. Data stream mining is the process of extracting interesting patterns and trends from a sequence of elements that arrive continuously in a rapid speed. Stream processing allows us to process data in real time as they arrive and quickly detect conditions within a small time period from the point of receiving data. Stream processing allows to feed data into analytics framework as soon as they are generated and get instant analytics results. Stream processing is useful in tasks like anomaly detection, alert filtering and fraud detection [Bif10]. In the IoT, a huge number of devices generate massive streams of data, which need intelligent data processing and online analysis. The storage, querying and mining of such datasets are highly computationally challenging tasks. Mining data streams is concerned with extracting knowledge structures represented in models and patterns in a non-stop stream of data. The research in data stream mining has gained a high attraction due to importance of its application in increasing generation of streaming information (IoT). Recently, the data generation rates have become faster than ever. We are surrounded by millions of devices, which generate data at a high frequency. This rapid generation of continuous streams of data has challenged storage, computation and communication capabilities in computing systems. Motivation to study data streams comes from a variety of areas: sensor networks may have a lot of nodes, each taking readings at a high rate. This data must be processed and analyzed.

2.3 Lambda Architecture:

Lambda Architecture [KMM+15] is a data processing design pattern to handle massive quantities of data and integrate batch processing and real time stream analysis within a single framework. It is a very generic pattern that tries to cater common requirements raised by most big data applications. It defines a set of patterns and guidelines for big data applications. It deals with both historical (batch) and real time data. It is technologically agnostic and generic in nature and it does not depend on any specific technologies. When processing large amounts of data, there is usually a delay between the point when data is collected and its availability in reports and dashboards. Often the delay results from the need to validate or at least identify granular data. However, in some cases, being able to react immediately to new data is more important than being sure of data's validity. The Lambda architecture is an approach that mixes both batch and stream processing and makes the combined data available on the serving layer. It is divided into three layers: the batch layer, serving layer and speed layer.

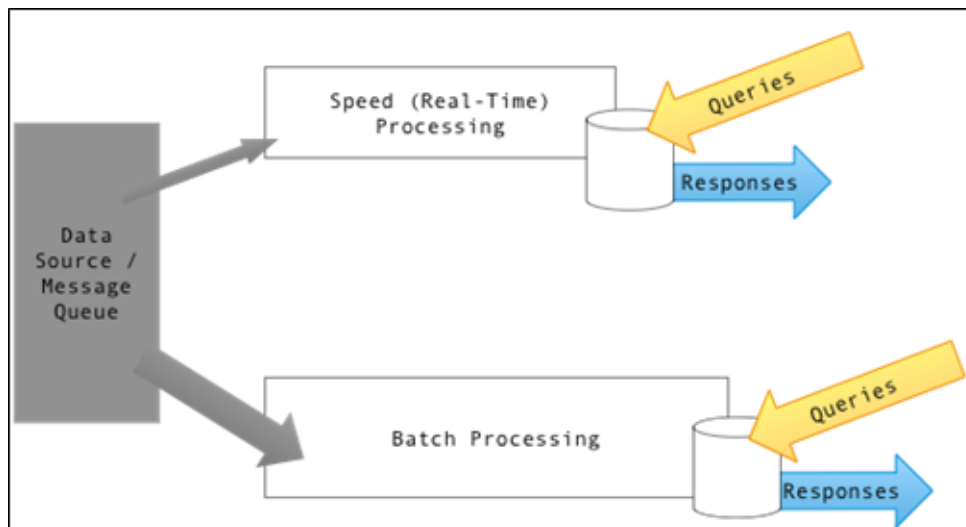


Figure 2.1: Lambda Architecture

- **Data Ingestion:** The data ingestion step comprises data ingestion in both speed and batch layer. For the batch layer, historical data can be ingested on demand at any desired interval. For the speed layer, the fast moving data must be captured as it is produced and streamed for analysis. The data is time tagged or time ordered. Some examples of high velocity data include log collection, website clickstream and IoT device event data.
- **Batch layer:** The batch layer is made for batch processing of the data. The batch layer aims at perfect accuracy as it processes all the available data when processing data. This layer is responsible for batch processing of stored data.
- **Speed layer:** The speed layer, also known as the real time layer is made for stream processing of data. It caters the real time analysis requirements. This layer performs on the fly analysis of data as it arrives. Since the batch layer deals with a large amount of data, processing of the batch layer usually takes time and the business cannot wait for this lag in processing of batch layer. So, for achieving near real-time data analysis, data is incremented to the low latency speed layer.
- **Serving layer:** The core task of the serving layer is to present the results created by both the batch and stream layer. Apart from this, there is a good amount of orchestration work which needs to be done by this layer. A query to the serving layer can be made to retrieve results from speed or the batch layer.

2.4 MBP Platform:

MBP is a platform which enables automatic provisioning and configuration of devices in the Internet of Things (IoT). In the emerging world of IoT, the integration of devices, sensors, and actuators has become more and more important. IoT benefits from data coming from a huge number of connected devices, which helps in creation of so-called smart environments.

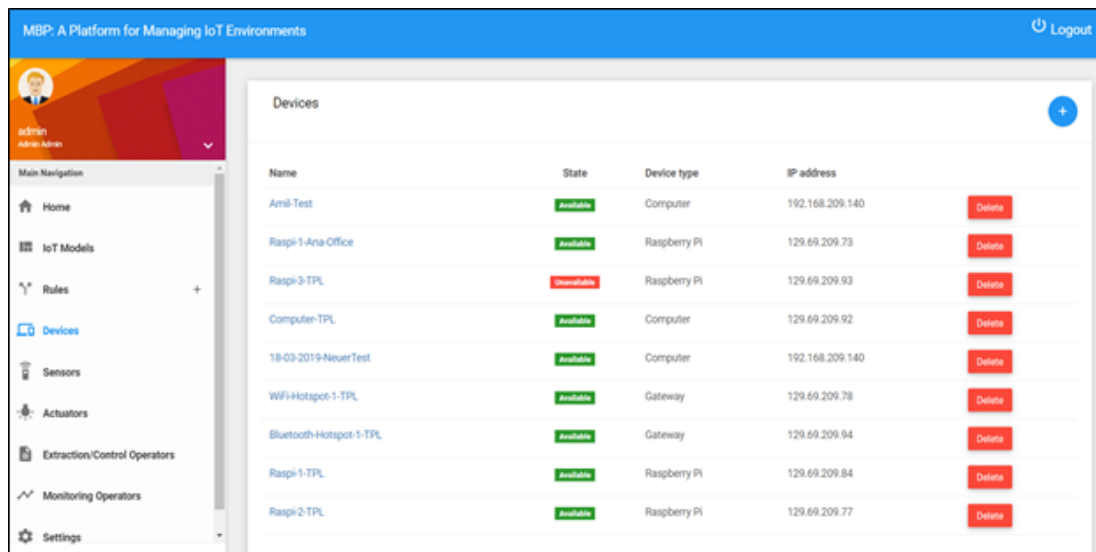


Figure 2.2: MBP Dashboard showing list of connected devices

MBP enables easy modeling of sensors, actuators, devices and their attributes. It also monitors devices in regards to changes or faults. Through the integration of sensor data, high level information can be derived that can lead to huge benefits. The integration of physical actuators controlling the real world IoT devices not only enables the self-organization of devices, but also allows IoT applications to control the smart environment themselves. Appropriate device adapters have to be manually created and deployed for each sensor to connect sensors and actuators to IoT middleware. Deploying these adapters manually is error prone and can take a lot of time. Hence, only automated registration, configuration and binding of devices enables efficient monitoring [HBS+16]. The MBP platform helps in this by reducing the amount of manual intervention required in setting up an IoT environment with sensors, actuators and devices.



Figure 2.3: MBP Dashboard showing historical data values from sensors

The MBP platform collects data from the devices and sensors it manages. Live data from sensors can be fetched by subscribing to the sensors managed by MBP platform. This platform also provides REST APIs for fetching historic data.

3 Related Work

Previously, research on Internet of Things (IoT) mainly focused on enabling general objects to see, hear and listen to surrounding objects and make them connected to share observations. However, later we understood that only being connected is not enough. Beyond that, these objects should have the capabilities to learn, think and understand each other. These connected objects generate massive amounts of big data in an explosive fashion. This raw data by itself does not have much value, but once analyzed, it can give information, which is otherwise not trivial to find. The future of IoT will be highly populated by a large number of heterogeneous networked embedded devices. To give an approximated idea, there would be 75 billion devices till year 2025, and most of these devices would be IoT devices. The share of IoT devices is growing exponentially in the recent years. The image below gives an idea about it:

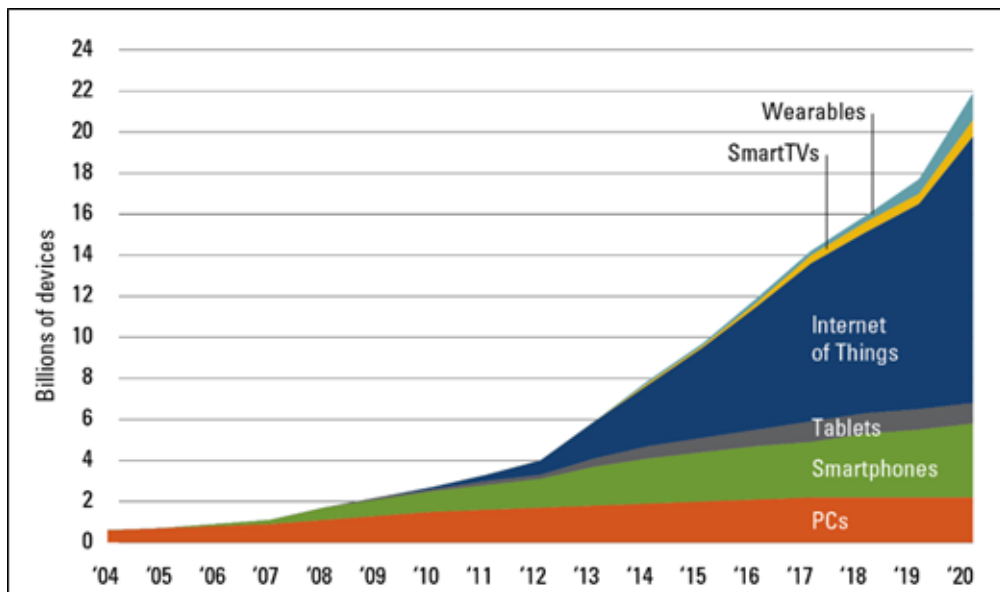


Figure 3.1: Increase in number of Connected Devices [Ali15]

The data collected from these devices may not have much value unless we analyze, interpret, understand and properly exploit it. In recent years, there have been a lot of work in the field of data analytics in the IoT. We will be discussing some of them in this section.

3.1 Lambda architecture in Amazon Web Services (AWS):

When processing large amounts of semi-structured data, there is a delay between the point of data collection and its availability in reports and dashboards. Often, the delay results from the need to validate or at least identify granular data. There are many AWS services available to analyze large volumes of data. For ingesting and processing stream or real-time data, AWS services, like Amazon Kinesis Data Streams, Amazon Kinesis Data Firehose, Spark Streaming and Spark SQL on top of an Amazon EMR cluster are widely used. Amazon Simple Storage Service (Amazon S3) forms the backbone of such architectures providing the persistent object storage layer for the AWS compute service. The image below shows how an AWS Lambda architecture implementation looks:

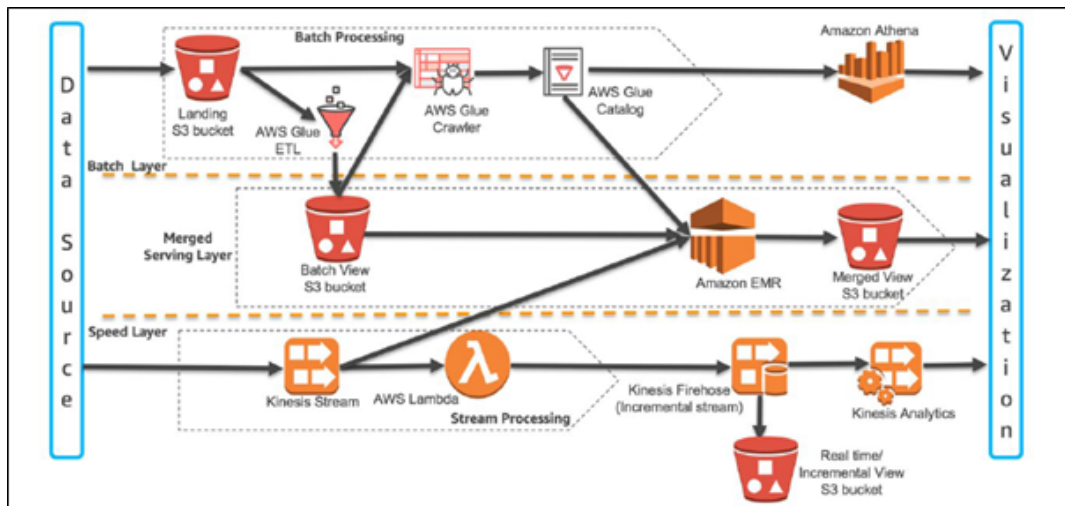


Figure 3.2: Implementation of Lambda Architecture in AWS [Raj18]

This Lambda architecture provides the building blocks of a unified architectural pattern that unifies stream and batch processing within a single code base. In long term, this architecture will reduce the maintenance overhead. It will also reduce the risk for errors resulting from duplicate code bases.

3.2 Microsoft Azure IoT analytics:

A large number of devices are part of a larger IoT solution in which devices send their data to the cloud for storage and analysis. At a higher level, IoT solutions can be broken down into core essentials of device connectivity and data processing and analysis. Device connectivity means simply devices that generate and collect data, which is sent to a cloud gateway. Think about how devices are being used today. Sensors have been connected to

cows, cars and spaceships. Your phone, step counter and smartwatch are all data generating devices. So, when dealing with device connectivity, the challenge is to figure out the best and most effective method and approach for providing not only secure, but also reliable connections. IoT devices are not like other typical client devices. Most of the IoT devices are not sitting at a place, like a desktop PC. These devices are out there in the world, continuously moving and, hence, have challenges, like slow or unreliable connections or limited power resources.

To overcome the innate challenges of IoT solutions, Microsoft introduced Azure IoT hub, a fully managed communications service that provides highly secure, scalable and dependable messaging between IoT devices and solutions. It supports both hardware and software scenarios, such as wide collection of devices, environments and scenarios which fits perfectly in the IoT ecosystem. It also supports millions of simultaneous connected devices. This solves the problem of connecting, fetching and storage of data. The next step is data processing and analysis. At this point, the data from devices is sitting in Azure IoT hub waiting to be picked up by another service for processing. IoT hub is only a temporary holding place for incoming data. The next step is to pick up that data and do something with it, whether it is immediate analysis of data or storing it for long term. Here, the Azure Stream Analytics (ASA) comes into picture. It is a real-time event processing engine that provides analytical capabilities on streaming data from devices, applications and sensors. It is a fully managed, highly scalable Azure service focused on making access to deep data insights powerful, inexpensive and easy to use.

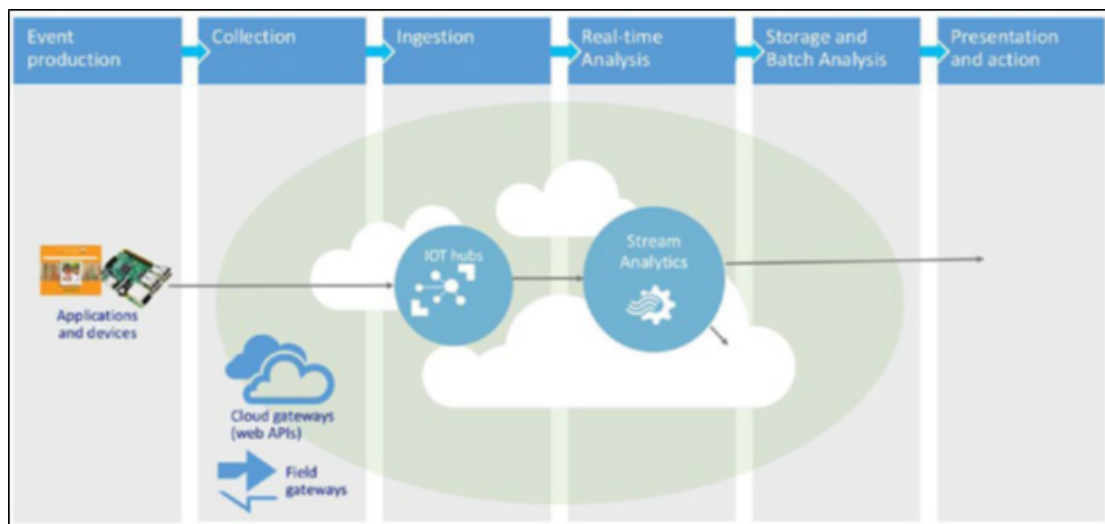


Figure 3.3: Microsoft Azure Stream Analytics framework [Kle17]

Azure Stream Analytics is a really powerful and efficient tool. In the following, we name some of its benefits and capabilities:

- With thousands of devices sending vast amounts of data, ASA is designed and built to handle millions of events per second, up to 1GB/second. The partitioning capabilities of event hubs makes this possible.

- Data loss prevention and business continuity, two key facets of Stream Analytics, are provided via built-in recovery features and the ability to maintain state. Hence, Stream analytics is able to achieve events and reapply processing.
- ASA allows connections to a lot of different sources and destinations.
- Not only creating inputs and outputs is extremely simple, but ASA also makes data transformation simple and supports a variant of SQL languages called Stream Analytics Query Language.
- As with other Azure cloud services, Stream analytics is designed to provide a high-value, real-time analysis solution for a low cost.

3.3 Analytics in Industrial Internet of Things (IIoT):

Ever since the Industrial Revolution, increasing the efficiency in the field of manufacturing has been a constant endeavor. The manufacturing industry has subsequently experienced the power of digitalization and power electronics. However, much of the data collected was only used for direct feedback control. Recently, the manufacturing industry has embarked upon yet another transformation, sparked by connectivity and advanced analytics. This is also known as advanced manufacturing in North America and Industry 4.0 in Europe. The data collected brings transparency about the machines' operations, the raw materials utilized, the facility logistics and even human operations. This transparency is brought about by the application of data analytics and machine learning to discover distinct patterns and characteristics in data. The overarching goal of using analytics in manufacturing is to improve productivity by reducing costs without compromising quality. This in turn makes the whole manufacturing more efficient. Advances in the field of big data analysis and machine learning offers a wide range of tools that can potentially be applied in manufacturing analytics. Analytics helps in improving quality, reducing warranty cost, improving overall yield and performing predictive maintenance.

3.4 Energy Management in Smart Homes using IoT Analytics:

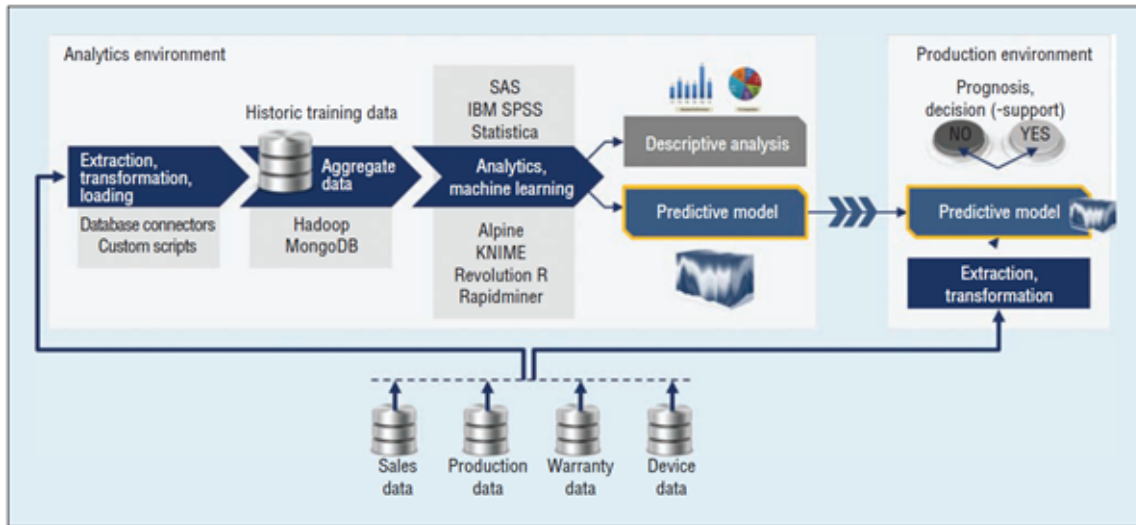


Figure 3.4: Analytics in Industrial IoT [LGS17]

Once data is collected from different devices and stored, a framework for data analysis is needed which is shown in the Figure above. The data first goes through an ETL (Extraction, Transformation and Loading) process and is then loaded into a distributed file system, such as HDFS (Hadoop distributed file system) or a NoSQL database. After this, machine learning and analytics tools perform predictive modeling or descriptive analysis. The big data software stack can be a mixture of open source, commercial and proprietary tools.

3.4 Energy Management in Smart Homes using IoT Analytics:

Increasing costs and demand of electricity has led many organizations to find smart ways to monitor, control and save energy. A smart EMS (Energy management system) can contribute towards cutting the costs while still meeting demands. The emerging IoT environment and big data analytics techniques can be used to better manage energy consumption in residential, commercial and industrial sectors. In a smart home, each connected device transmits data to a centralized server. This big data can be utilized using analytics techniques to get insights and help into achieving more efficient energy consumption. Such a system has following system requirements:

- Energy consumption data from devices should be gathered and sent to centralized server.
- The stored data should be used by the analytics engine to process it and generate reports, charts, etc.
- Clients should be able to view the generated graphs through a cross platform application.

3 Related Work

- Depending on user privileges, the application should render different services to each user, such as viewing reports, status of devices, remote control of devices, etc.

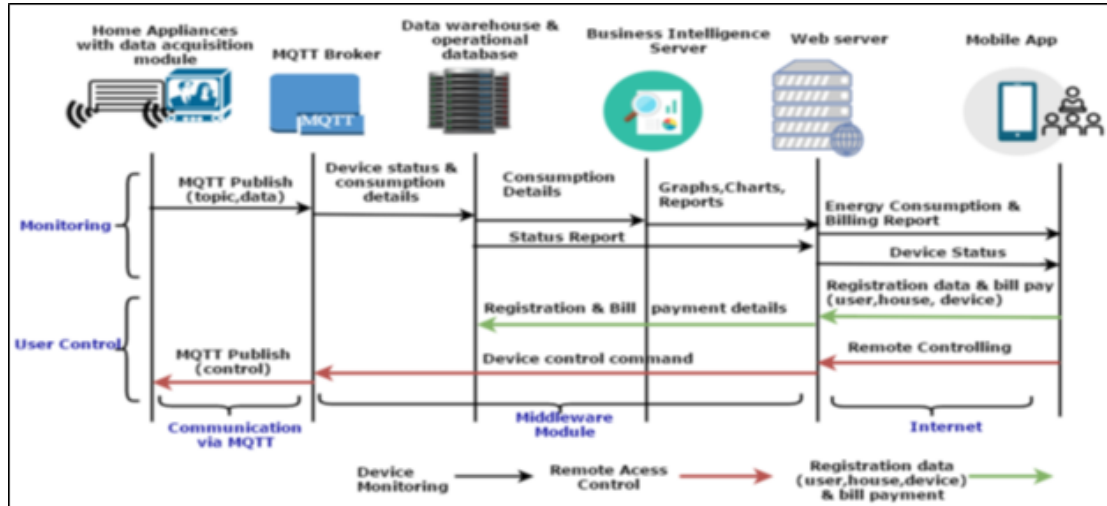


Figure 3.5: Sequence diagram for collection and processing of data [AZR+17]

This kind of a system is set to open new avenues for smart energy management on IoT platforms. The system uses data analytics and scalable storage for building a smart Energy management system to aid different stakeholders with their respective privileges. This kind of system also empowers users to remotely control and monitor the devices.

All these above mentioned systems/architectures are a great inspiration and give a deep insight of data analytics in IoT. They all have their own pros and cons. The goal of this thesis is to take useful points from these systems and build an analytics framework for the IoT platform MBP. MBP is the management platform and, hence, all the devices and sensors are managed by this platform. Data collected from the devices and sensors and stored by the MBP, and can be easily retrieved. Historic data can be retrieved on-demand using the REST APIs and live sensor data can be retrieved by subscribing to the relevant sensor topic for messages. Also, most of the analytics systems focus on either data stream mining or batch processing of data. In this thesis, the goal is to perform both stream mining and batch processing on data from same devices. In this way, we can use the benefits of both stream and batch processing. This is a big advantage of using the Lambda Architecture. Many interesting use cases of data analytics in IoT can also be derived from the above mentioned scenarios. We will be implementing some of the use cases and present the results later in this thesis.

4 Technologies Used

The analytics framework we intend to create for the IoT platform MBP, will be a combination of technologies that help us in performing various small tasks in the framework. As we all know that a big, well-functioning machine is a combination of various small parts which make it function in an expected way, in the same way the analytics framework would be a combination of various small parts combined to enable the intended functionalities. If broken down, the major tasks of the framework would be to collect data from IoT framework, perform batch and stream processing on it and store the results. All these should be enabled by a GUI, so all these functionalities would also be accessible by making REST API calls from the frontend. We took into consideration some open-source tools and picked up few of them based on their advantages and disadvantages. We discuss them further in this section.

4.1 Apache Spark (Batch Processing engine)

The growth of data volumes in industry and research poses tremendous opportunities as well as tremendous computational challenges. In the Apache open source stack, systems like Impala and Storm are also specialized. Even the relational database is moving away from “one-size-fits-all” systems. Apache Spark [ZXW+16] has a programming model similar to Map-Reduce but extends it with data sharing abstraction called “Resilient Distributed Datasets” or RDDs. Using this simple extension, Spark can capture a wide range of processing workloads that previously needed separate processing engines. Spark’s generality has several important benefits. Applications are easier to develop because they use a unified API and also, it is more efficient to combine processing tasks. Spark can run diverse functions over same data, often in memory. A powerful analogy for the value of unification would be to compare smartphones to separate portable devices that existed before them (cameras, cellphones, GPS gadgets, etc.). As parallel data processing becomes common, the composability of processing functions will be one of the most important concerns for usability and performance. The key programming abstraction in Spark are RDDs, which are fault-tolerant collections of objects partitioned across a cluster that can be manipulated in parallel. Spark exposes RDDs through functional programming APIs in Python, Java, Scala and R. Users can simply pass local functions to run on the cluster.

4.1.1 Capabilities and performance of Spark:

Spark's most common applications are for batch processing of large datasets, including ETL (Extract-Transform-Load) processes and offline training of machine learning models. The largest published use case of Spark is an 8000-node cluster at the Chinese social media 'Tencent' that ingests 1PB of data per day. Spark provides a machine learning library (MLib), which implements more than 50 algorithms for distributed model training. For example, it includes common distributed algorithms for decision trees (PLANET), K-means clustering and Alternating Least Square matrix factorization. Also, implementing the optimizations on RDDs, Spark can often match the performance of specialized processing engines. The figure below shows the performance of Spark versus other processing engines on three different tasks: a SQL query, a streaming word count and Alternating least Square factorization. While the results vary for varying workloads, performance of Spark is generally comparable to specialized engines. Even on highly competitive benchmark tests, Spark achieves state of the art performance.

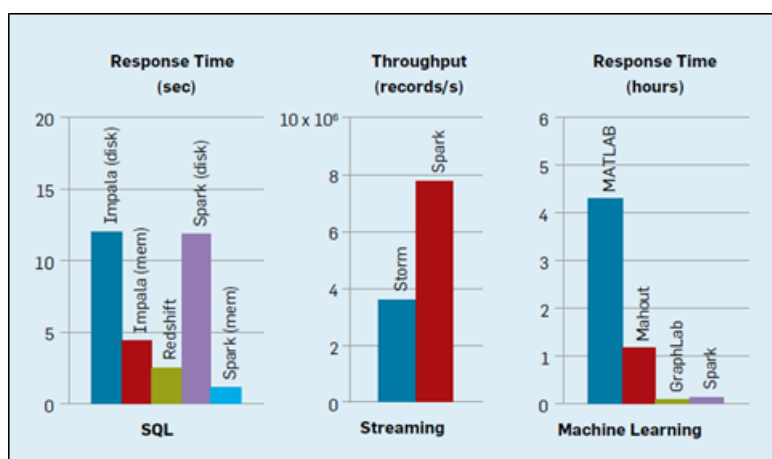


Figure 4.1: Performance of Spark compared to other tools [ZXW+16]

4.2 Scikit-Multiflow (Stream Processing engine)

Scikit-multiflow [MRBA18] is a framework for learning from data streams and multi-output learning in Python. It provides many state-of-the-art data generators, learning models and evaluators for different stream learning problems including single or multiple output and multi-label. Recent years have witnessed the emergence of Free and Open-Source software in the research community. Especially in the field of machine learning, researchers have benefited from the availability of different frameworks that provide tools for faster and easier development, allow reproducibility and replicability of results and promote collaboration. Scikit-multiflow was developed based on this principle. It is

a Python framework to implement algorithms and perform experiments in the field of Machine learning and evolving data streams. Scikit-multiflow is inspired by the popular frameworks scikit-learn, MEKA [RRPH16] and MOA [BGHP18].

Scikit-learn features various classification, regression and clustering algorithms including random forest, gradient boosting, support vector machines, k-means and DBSCAN, and is designed to operate well with existing Python numerical and scientific packages NumPy and SciPy. As a multi-output streaming framework, scikit-multiflow serves as a bridge between research communities that have flourished around the above mentioned popular frameworks, providing a common ground where they can thrive. Scikit-multiflow helps with the democratization of Stream learning by bringing this research field closer to machine learning community. It is important to notice that scikit-multiflow complements scikit-learn, whose primary objective is batch processing of data, expanding the set of free and open source tools for stream learning. Special focus in the design of scikit-multiflow is to make it friendly to new users and familiar of experienced ones. Scikit-multiflow contains leaning methods, stream generators, evaluation methods and change detectors. The base class in scikit-multiflow is StreamModel which contains the following abstract methods for the subclass to implement:

- `fit`: Trains a model in batch processing fashion. Works as an interface to batch methods that implements a fit function, like in scikit-learn methods.
- `partial_fit`: Incrementally trains a stream model.
- `predict`: Predict the target value in supervised learning methods.
- `predict_proba`: Calculates per class probability in classification problems.

StreamModel object interacts with two other objects: a Stream object and a StreamEvaluator object. The Stream object provides a continuous flow of data on request. The StreamEvaluator performs various tasks: queries the stream of data for new data, trains and tests the model on incoming data and continuously tracks the performance of the model. There are also various ways to create a stream of data. A stream of data can be created either from a data source (Apache Kafka) or from a file source (txt, csv, etc.). Kafka is an extensive framework to handle data streams (as messages). Kafka [Gar13] has the capability of merging streams from different sources and combining it into a single stream of data. The data in the stream is also aligned to the time of arrival to Kafka, and hence, even in the combined stream, the messages will be processed in FIFO manner. Kafka can also be used to break down a data stream into multiple streams to be processed by multiple stream processing engines. On the other hand, a file source is much simpler and operates on very basic functionalities. A message subscriber waits for messages from the device. When it receives a message, it reads it and appends it to a file. A 'FileWatcher' is implemented to look for new data in the file and when it finds new data, it reads it and sends it to the stream processing engine for analysis. In our thesis, we will be using the file source to create data streams as it is easier and less complicated to implement and fulfils our requirements.

4.3 Python Flask (enabling REST APIs):

Flask [Gri18] stands out from other web development frameworks because it lets developers have full control of their applications. The key to this freedom is that from the very start Flask was designed to be extended. It comes with a robust core that includes the basic functionality that all web applications need. Flask is a small framework by most standards, small enough to be called a “micro-framework”. This quality of Flask makes the source code readable and understandable. But being small does not mean that it lacks functionalities. Flask has three main dependencies. The routing, debugging and Web Server Gateway Interface (WSGI) subsystems come from Werkzeug; the template support is provided by Jinja; and the command line integration comes from Click. The best part about Flask is that it is really easy and simple to create and deploy web applications without any other server, like Apache Tomcat. Flask was designed to be easy to use and extend. The idea behind Flask is to enable developers to build a solid foundation for web applications of different scales. Developers are free to plug in any extensions needed for the application. One is also free to build own modules which makes Flask great for all kinds of projects. Flask is one of the most polished and feature rich frameworks available for web development.

All Flask applications must create an application instance of an object of class ‘Flask’. The web server passes all the requests it receives from clients to this object for handling. Clients, such as web browsers, send requests to the web server, which in turn sends them to the Flask instance. The Flask application should understand what code it needs to execute for the URL requested, so it keeps a mapping of URLs to Python functions. The association between a URL and the function that handles it is called a route. For the thesis, we will be creating these routes based on our requirements using a Flask application.

4.4 MQTT (Message Protocol):

Wireless sensor networks (WSNs) pose novel challenges compared to traditional networks. To cope up with such challenges, a new communication paradigm has emerged. Compared to other variants, publish/subscribe systems are common and wide-spread in distributed computing. MQTT [HTS08] is also based on the publish/subscribe paradigm. It is designed in such a way that it can run on low-end and battery operated sensors, actuators and devices and operate over networks with constrained bandwidth. These WSNs are the base for IoT devices. Millions of devices communicate with each other and share data. Within a WSN, a large number of battery-operated devices with limited amount of storage and processing capabilities, collect information about the environment and send them to gateways for further processing. Hence, for this type of communication, we require a protocol which uses a lot less bandwidth and is also energy efficient. MQTT provides the solution to this. MQTT uses the data-centric communication approach, in which the data collected is delivered to the consumers not based on the network addresses, but rather as a function of their contents and interests. Publish/Subscribe messaging systems are well-known examples of data centric

communication. These features are achieved by decoupling the various communicating components from each other such that it is easy to add new data sources/consumers or to replace existing modules. MQTT is designed especially for operation on low-cost and low-power devices running over bandwidth-constrained wireless networks.

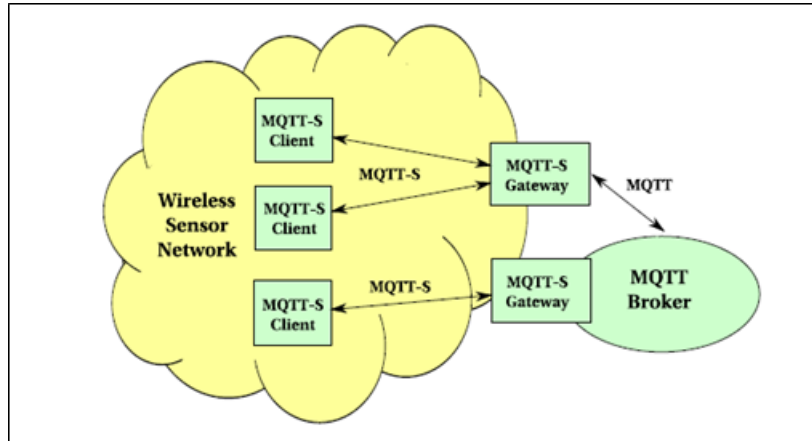


Figure 4.2: Architecture of MQTT [HTS08]

The figure above shows the MQTT architecture. There are two types of components: MQTT clients and MQTT gateways. The clients are on the wireless network side and enable the devices to access the pub/sub services of the MQTT broker located on the traditional network. They connect to the gateway using the MQTT protocol and the gateway connects to the broker. One of the weaknesses of wireless networks is their high link failure rates. Links between the device and gateway can fail at any given time, thus, disconnecting the device from the broker. Hence, it is highly desirable that a device can connect to multiple gateways so that if the connection to one gateway fails, it can connect to the broker via another gateway. Another reason for requiring presence of multiple gateways is the low transmission capacity of the wireless links. Links in the proximity of gateways could become congested if large number of devices exchange messages with a gateway. Hence, having more than one gateway helps remedy that situation because it provides load balancing. MQTT allows clients to connect to multiple gateways and use it to their advantage. As mentioned before, MQTT is based on the publish/subscribe paradigm of communication. The process of registering an interest and getting data from a particular device is called subscription, and the interested party is called subscriber. Components which want to produce certain information do so by publishing their information, these components are called publishers. A so-called broker ensures that the data gets from the publishers to the subscribers. As explained before, MBP is the platform to manage IoT devices. These devices send their data to MBP. To access this data, a subscriber of a particular device could be created and whenever this device produces new data, the subscriber can receive it. For the thesis, the analytics framework will be using a MQTT subscriber to get live data from sensors and use it for creating streams of data for analysis as explained before. The MBP platform provides topics to which a subscriber can subscribe to. MBP uses the sensor

ID as the topic name and publishes new messages on this topic. The MQTT subscriber implements a callback function which is invoked whenever a new message arrives. The implementation of an 'onMessage' function registers a handler for incoming messages. Whenever a subscriber receives a new message, this method is invoked. As a developer, one can further process this message in the implementation of this 'onMessage' function.

5 System Architecture

In the last chapter, we discussed the technologies which are used to create the framework. In this section, we discuss the high-level design of the analytics framework. The framework accepts requests from the client and performs the required action based on the request type. The request generally contains what algorithm is to be applied and on which device the analysis is to be performed. The analytics framework can be deployed on the same machine as the MBP, or any other machine. The only requirement to run the framework is that Python should be installed on the machine. A MBP admin, who has the access to the MBP platform, is able to send requests to the analytics framework from the same user interface. The framework performs the desired analysis (stream or batch processing) on the requested device. The framework creates analysis models and stores it on the file system. These models can also be later retrieved and used. In the following sections, we discuss the architecture of the framework and how it communicates with the MBP platform.

5.1 Basic Blocks of the System:

The image below shows the basic blocks of the analytics framework:

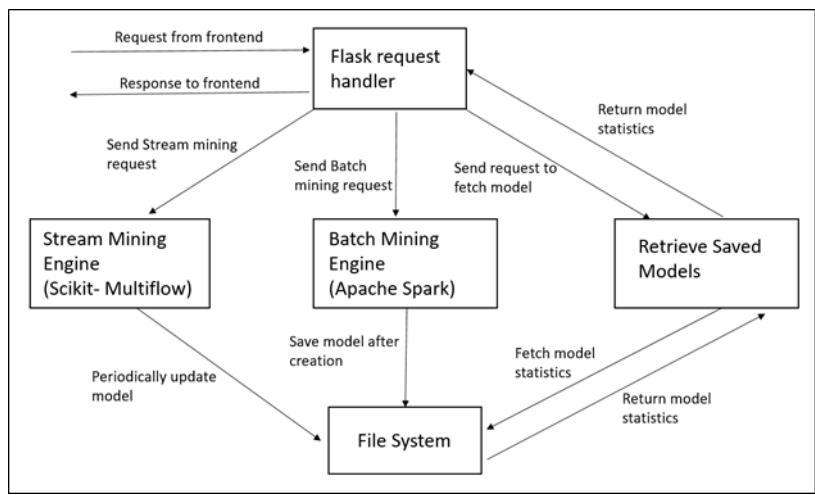


Figure 5.1: Basic blocks of the analytics framework

Flask helps in creating micro-services in Python to handle requests and provide appropriate responses to it. This communication is done using the REST API. After flask gets a request from the MBP, based on the request parameters, it either initiates a stream processing engine or the batch processing engine. Users can also view the models already created and stored on the File system. Users can view the statistics about the model and also use the model further. As the name suggests, the batch processing engine is used for on-demand batch processing of data. It uses Apache Spark as a base for creating machine-learning models. This engine creates models and saves them for later use. The stream mining engine is used for data stream mining. It reads the stream of data from the device, updates the model according to new data and then saves the model periodically on the file system. The current status of the stream model can also be retrieved and viewed on the user interface.

5.2 Technologies and Communication with MBP:

The figure below shows the technologies used in different parts of the MBP and the analytics framework, and also how the communication happens between these two systems:

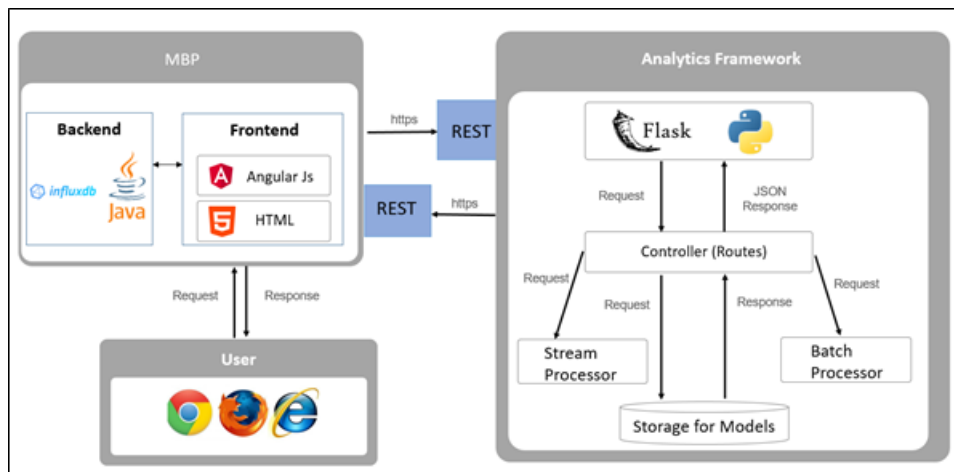


Figure 5.2: Technologies used in MBP and the analytics framework

There is no separate user interface for the analytics framework. The current user interface of the MBP is being extended with more functionalities to access the analytics framework. As one can see in the image, the frontend uses Angular and HTML codes. MBP has its own backend which is based on Java. The MBP uses this backend to manage devices and sensors and store data from them into the database. When a user requests for an analysis model, an https request from the MBP is sent to the analytics framework. This request is made using a REST API call. After receiving the request, the controller on the analytics framework maps the request to the respective engine for processing. Flask is used to create these request handlers (controllers) and the processing engines are created in Python. MBP also provides REST APIs for fetching historical data from devices. Therefore, when the

batch-processing engine gets a request to create an analytics model, it makes a REST API call to the MBP for fetching the historical data for that device. The communication between the two systems is mainly supported by REST calls. When MBP receives data from the sensors/devices connected to it, it publishes the data to a MQTT topic. Therefore, to get the live data from sensors, a MQTT subscriber can be created which will subscribe to the relevant sensor topic and get the live data. This mechanism is used for stream mining of live data from the sensor.

5.3 Sequence Diagram:

A sequence diagram depicts the interaction between objects and also the sequence of activities performed by the objects. The sequence diagram describes how and in what order the objects in the system function.

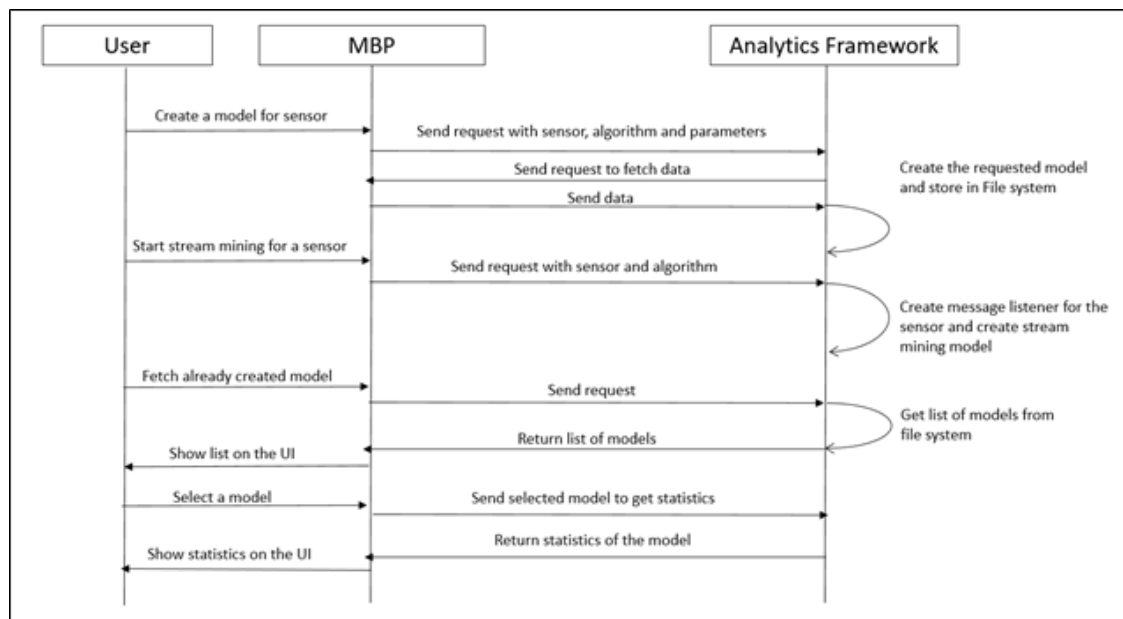


Figure 5.3: Sequence diagram showing sequence of commands execution

The image above shows the sequence diagram for interaction between the MBP and the analytics framework. Looking at the image, one can see how the interaction takes place between the two systems. Users can send a request to create a model for a particular sensor. The MBP gets this request from the user interface and sends it to the analytics framework. If it is a batch processing request, the analytics framework sends a request back to the MBP for getting the historical data of that particular device. The MBP sends the data back to the analytics framework for the relevant device and the data is used to create a model. The analytics framework creates the requested model and stores it into the file system. If the user makes a request for stream processing, the analytics framework creates a stream

mining object for that device. The stream mining object consists of a message subscriber to that device and a mining model. The message subscriber fetches live data and the mining model is updated based on the data received. This mining model is periodically stored on the file system. Users also have the option to view the models already created on the user interface. Users send a request to get the models that are already created and stored on the file system. The MBP forwards this request to the analytics framework and the analytics framework fetches the list of models and sends it back. Users can then select any model from the list and view the statistics about the model. Users also have the ability to use this model for further use as just creating and storing the models will not be beneficial. The details of how users can create, fetch and use the models from the user interface is covered in detail later in the Implementation chapter of the thesis.

6 Implementation

In the previous chapter, we discussed the high-level system architecture of the analytics framework. In this chapter, we discuss the implementation of the analytics framework. Topics include how a model can be created, how to access the statistics of a particular model and how to use it. The existing MBP framework frontend was extended to access the analytics framework and provide the functionalities listed before. The MBP platform is used to send requests to the analytics framework and also for viewing results. The analytics framework provides REST APIs which are used to implement the functionalities we aim for. Examples of REST calls to the analytics framework from the MBP platform will be provided in this chapter. We will now discuss the functionalities provided by analytics framework and how they are presented in the MBP user interface.

6.1 Implementation of functionalities on MBP user interface:

1. Showing all the existing models in the MBP user interface.

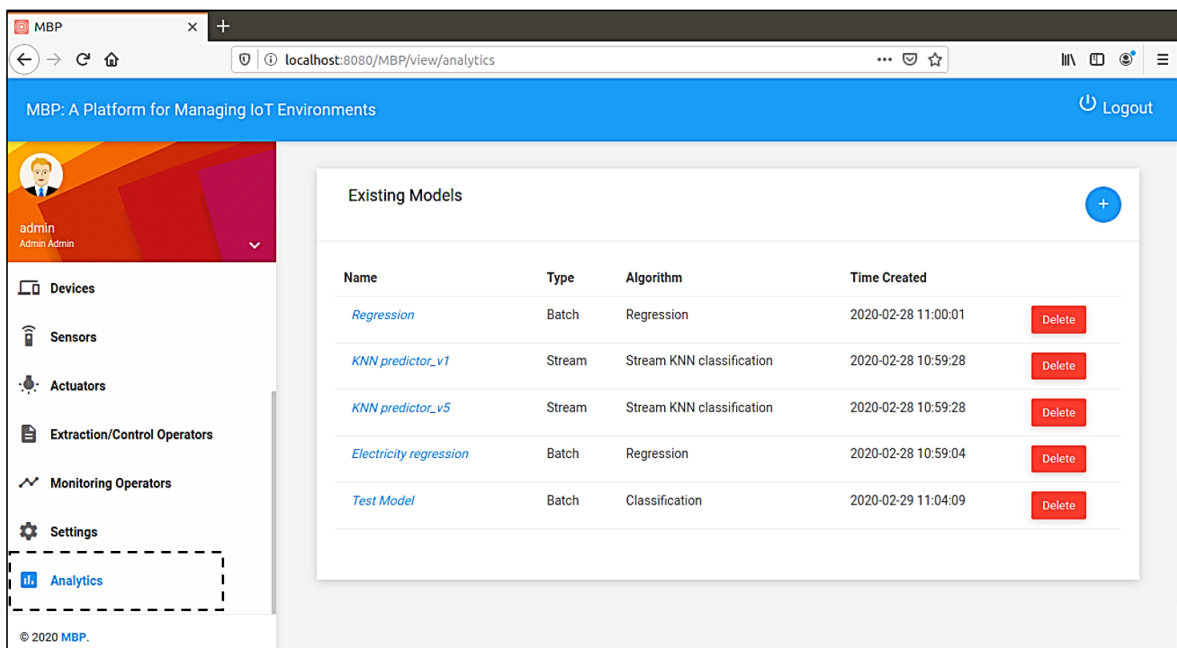


Figure 6.1: New Analytics menu item on MBP user interface

A new menu item “Analytics” has been introduced on the MBP user interface. Clicking this menu item sends a REST call to the analytics server. The analytics server returns a JSON message containing a list of all the models, their type, the algorithm used to create the model and the time of creation of the model. These statistics are presented in a tabular form in the user interface.

2. Creating a Batch processing model.

The ‘+’ button on the top right corner of the User interface can be used for creating a model. Clicking on the button opens a form and the user needs to provide the input values for the model. The following image shows the input fields of the form.

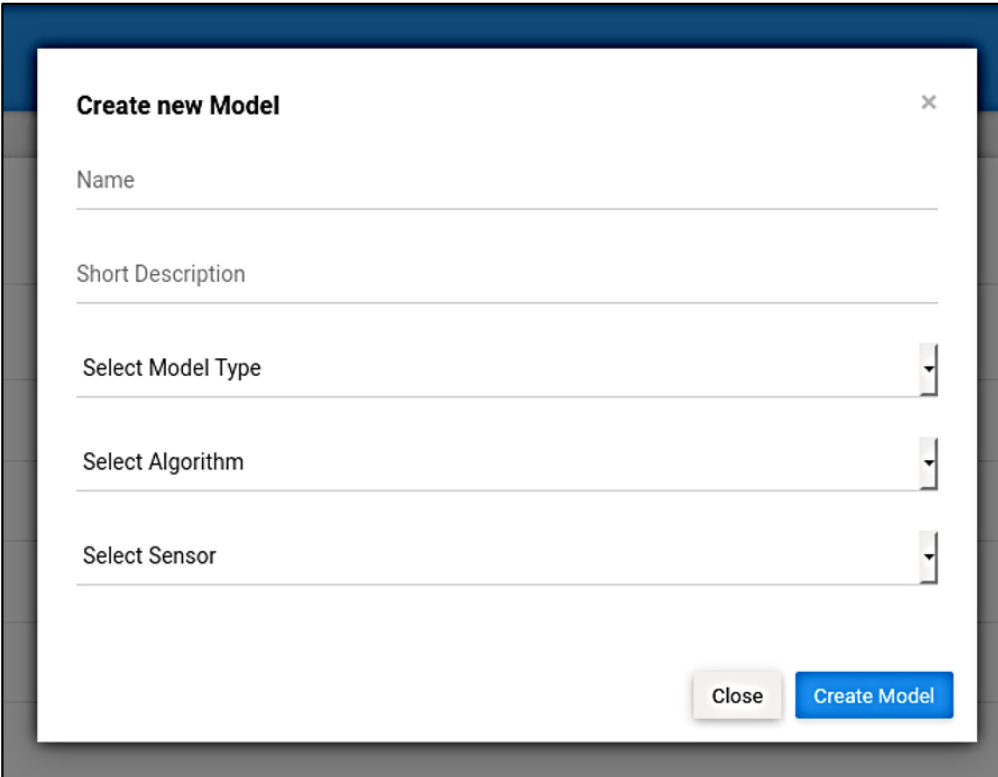
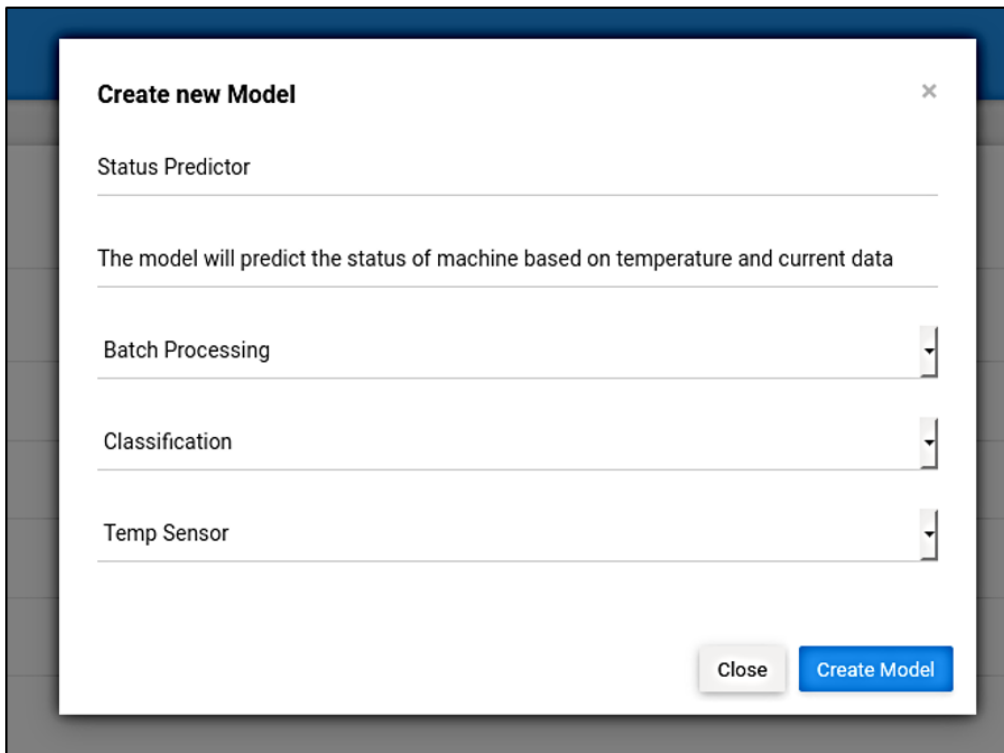


Figure 6.2: Form to create analytics model

Users must provide the name of the model to be created, a small description of what the model does, choose the type of model to be created (batch or stream), select an algorithm from the list of available algorithms and select a sensor from the list of available sensors. Clicking the “Create Model” button will send a REST request to the analytics server to create the requested model. The following image shows a sample form with filled values for creating a batch processing model.



The image shows a 'Create new Model' dialog box. It has a title bar with a close button (X). The main content area contains a text input field with 'Status Predictor', a description field with 'The model will predict the status of machine based on temperature and current data', and three dropdown menus: 'Batch Processing', 'Classification', and 'Temp Sensor'. At the bottom right, there are two buttons: 'Close' and 'Create Model'.

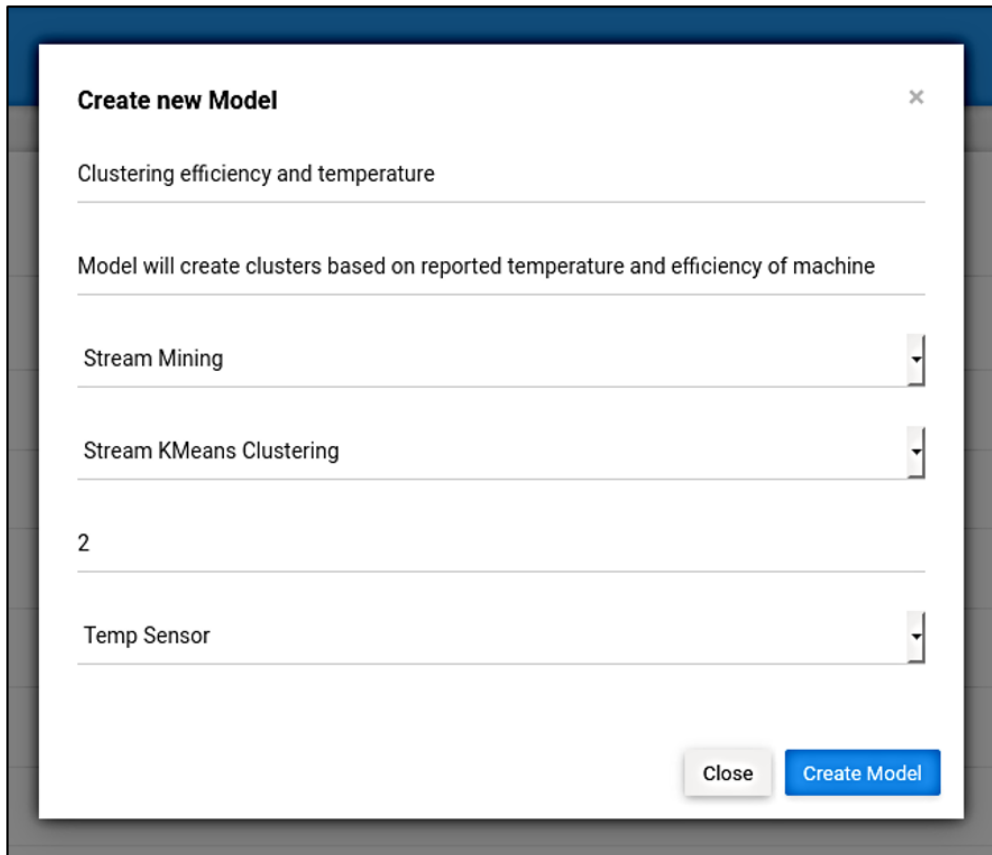
Figure 6.3: Sample input to create a batch processing model

For example, the REST request sent to the analytics server could look like:

`http://localhost:5000/createmodel?algorithm=Classification&sensorid=5e26cb10016d0404ec39ebee&name=Status Predictor&description=The model will predict the status of machine based on temperature and current data`

3. Creating a Stream processing model.

For creating a Stream Processing model, the same form can be used. Users will have to select “Stream Mining” as the type of model. To create a stream mining model, users will have to provide an extra input, which is the “Time (in days)”. This specifies how long the algorithm would run on a given data stream. The analytics framework is by default designed to store a snapshot of the stream mining model every 30 minutes. This is a property which can be configured while setting up the analytics framework. The following image shows a sample form with filled values for creating a stream mining model.



Create new Model [x]

Clustering efficiency and temperature

Model will create clusters based on reported temperature and efficiency of machine

Stream Mining

Stream KMeans Clustering

2

Temp Sensor

Close Create Model

Figure 6.4: Sample input to create a stream mining model

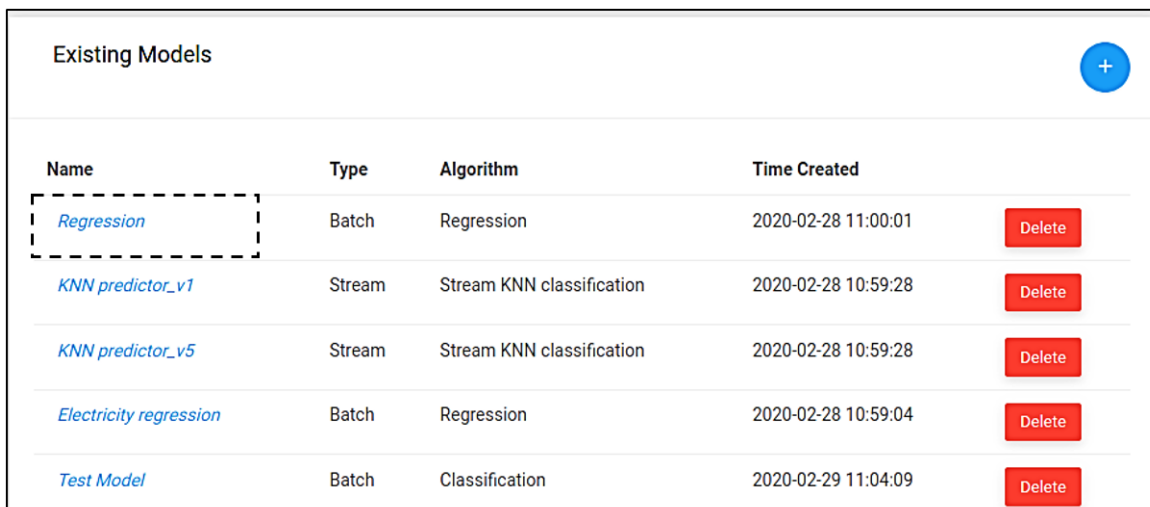
For example, the REST request sent to the analytics server could look like:

```
http://localhost:5000/createmodel?algorithm=Stream KMeans Clustering&sensorid=5e26cb10016d0404ec39ebea&name=Clustering efficiency and temperature&time=2&description=Model will create clusters based on reported temperature and efficiency of machine
```

4. Viewing statistics of a particular model.

To access the statistics about an existing model, user needs to click on the name of the model. The name is highlighted to indicate that it is a clickable link.

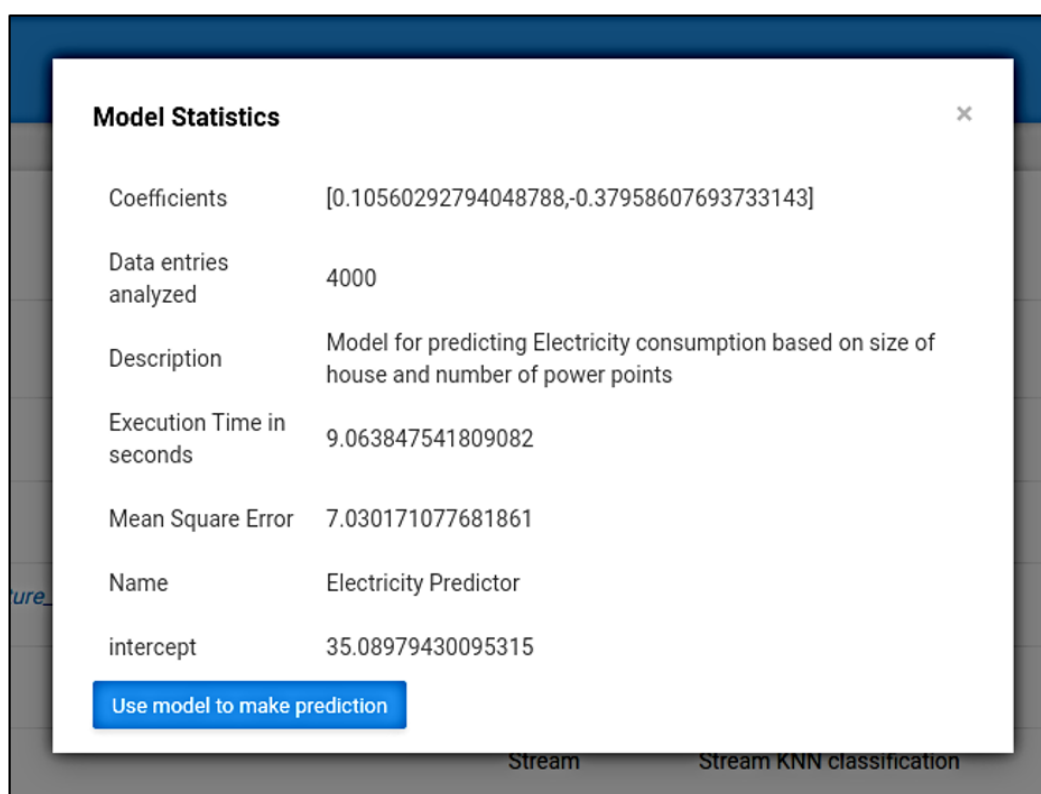
6.1 Implementation of functionalities on MBP user interface:



Name	Type	Algorithm	Time Created	
Regression	Batch	Regression	2020-02-28 11:00:01	Delete
KNN predictor_v1	Stream	Stream KNN classification	2020-02-28 10:59:28	Delete
KNN predictor_v5	Stream	Stream KNN classification	2020-02-28 10:59:28	Delete
Electricity regression	Batch	Regression	2020-02-28 10:59:04	Delete
Test Model	Batch	Classification	2020-02-29 11:04:09	Delete

Figure 6.5: Access an existing model

Clicking the link opens a new window which shows the statistics of the model.



Model Statistics [Close]

Coefficients	[0.10560292794048788,-0.37958607693733143]
Data entries analyzed	4000
Description	Model for predicting Electricity consumption based on size of house and number of power points
Execution Time in seconds	9.063847541809082
Mean Square Error	7.030171077681861
Name	Electricity Predictor
intercept	35.08979430095315

[Use model to make prediction](#)

Figure 6.6: View Statistics of selected model

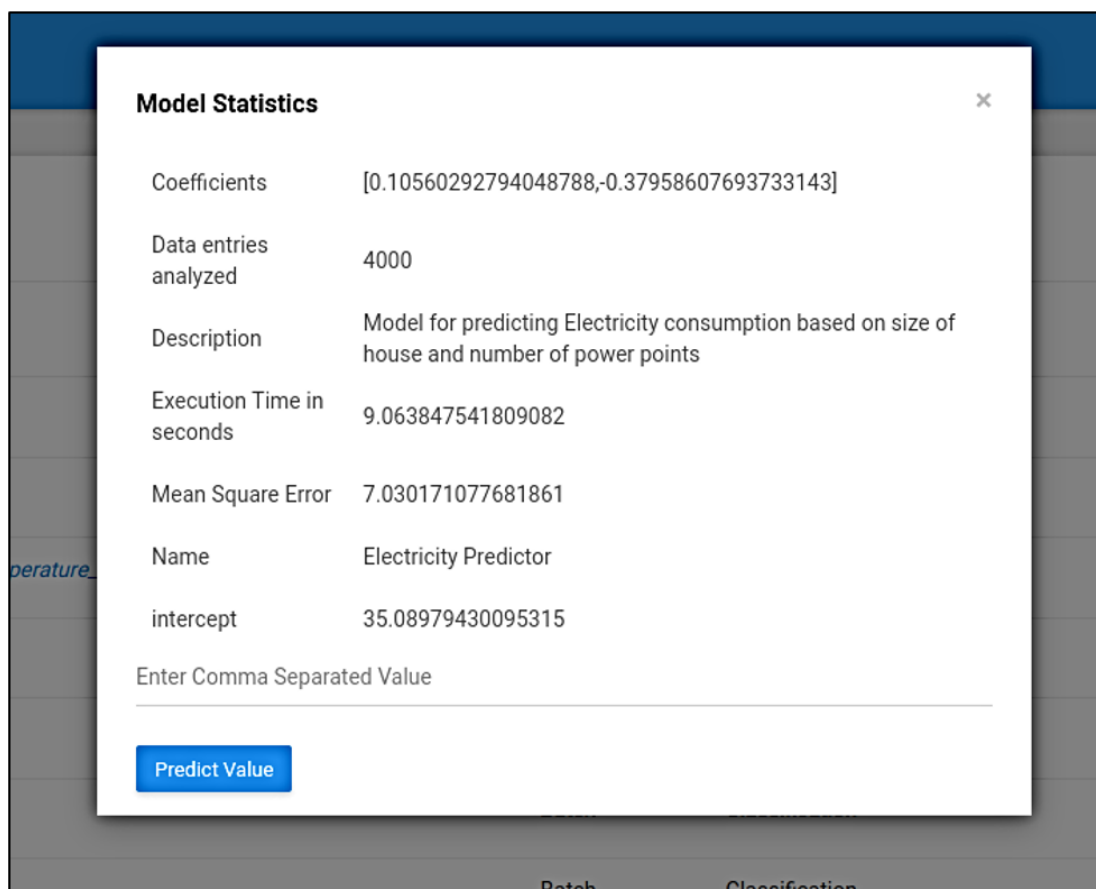
For example, the REST request sent to the analytics server could look like:

http://localhost:5000/getstatistics?model_name=Electricity Predictor

5. Using a model to make predictions.

If a model is created based on an algorithm which can be used to make a prediction (Regression, Classification), then the model can be used to make predictions based on input values. The user will see the “Use model to make prediction” button only when a model has the capability to make predictions.

Clicking the button would show a field where users can enter the values to make a prediction. The following image shows the input field.



The image shows a 'Model Statistics' dialog box with the following information:

Coefficients	[0.10560292794048788,-0.37958607693733143]
Data entries analyzed	4000
Description	Model for predicting Electricity consumption based on size of house and number of power points
Execution Time in seconds	9.063847541809082
Mean Square Error	7.030171077681861
Name	Electricity Predictor
intercept	35.08979430095315

Below the statistics is an input field labeled 'Enter Comma Separated Value' and a blue 'Predict Value' button.

Figure 6.7: Form to sent value to make prediction

Users have to enter comma separated values to make a prediction. For example, this model predicts electricity consumption based on the size of a house and number of power points in the house. So, for a house of size 90 square meters and having 10 power points, a sample input would be “90,10”. Clicking the “Predict Value” button will send a request to the analytics server to predict the result. The following image shows the predicted result value based on the inputs.

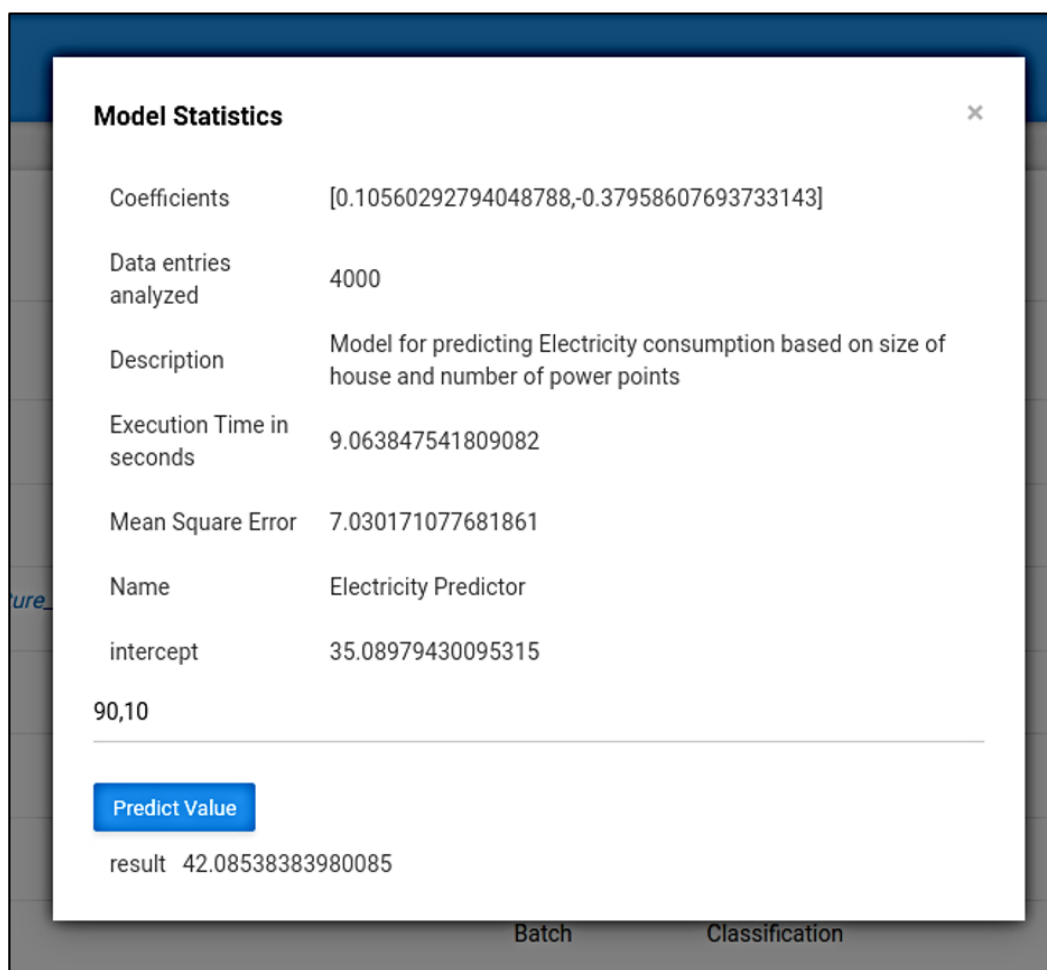


Figure 6.8: Predicted result based on input values

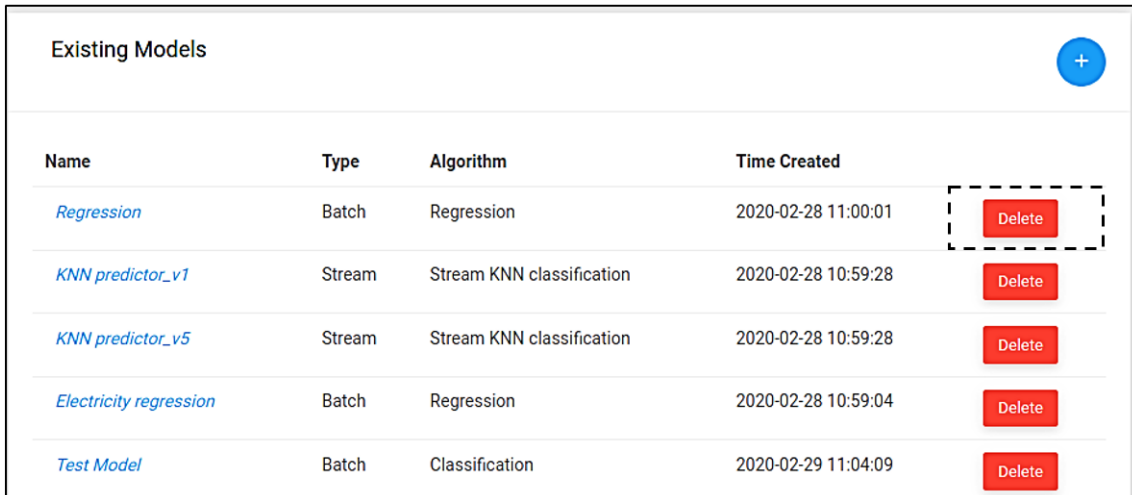
So, according to the selected model, a house of size 90 square meters and having 10 power points will consume 42.085 units of electricity.

In this example, the REST request sent to the analytics server looks like:

http://localhost:5000/getprediction?model_name=Electricity Predictor&value=90,10

6. Deleting a model

A user may want to delete a previously created model. This may be due to the number of models, or the performance of models. Users may not like the statistics of a model and may wish to delete it as it would serve no purpose to store it longer. Especially in case of Stream Mining models, there will be a model generated every 30 minutes. So, at the end of 5 days there will be 240 models. Not all models will be useful and hence, users may want to keep a few out of them and delete the rest after viewing their performance and statistics.



Name	Type	Algorithm	Time Created	
<i>Regression</i>	Batch	Regression	2020-02-28 11:00:01	Delete
<i>KNN predictor_v1</i>	Stream	Stream KNN classification	2020-02-28 10:59:28	Delete
<i>KNN predictor_v5</i>	Stream	Stream KNN classification	2020-02-28 10:59:28	Delete
<i>Electricity regression</i>	Batch	Regression	2020-02-28 10:59:04	Delete
<i>Test Model</i>	Batch	Classification	2020-02-29 11:04:09	Delete

Figure 6.9: Delete button to delete the model

There is a “Delete” button provided next to every model which can be used to delete the particular model. Clicking on the button shows a confirmation dialog box to confirm the deletion of model.

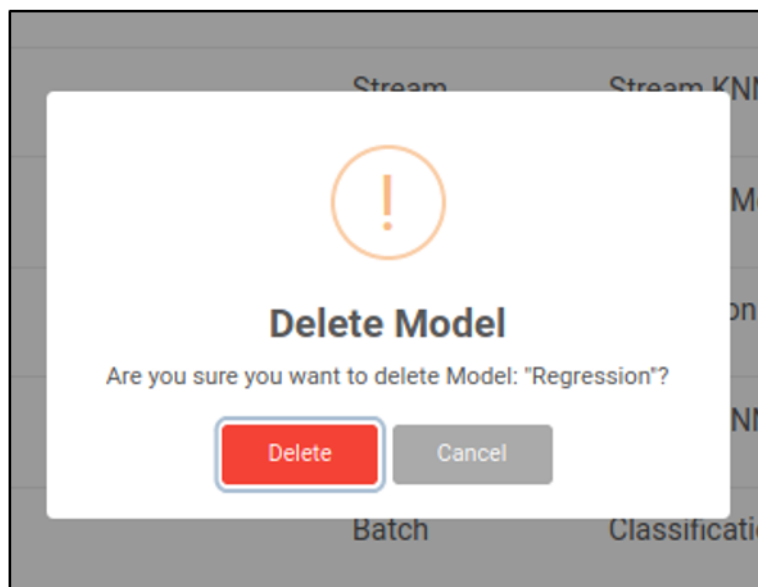


Figure 6.10: Confirmation box to delete the selected model

Clicking on “Delete” will confirm the deletion and will send a request to the analytics server to delete the model.

The REST request for deletion sent to the analytics server looks like:

`http://localhost:5000/deletemodel?model_name=Regression`

6.2 Implementation of functionalities on the analytics server:

A Python Flask application is used to implement the functionalities on the analytics server. A Flask application must implement an application file which acts as an entry point for all the requests made to the server.

This file contains all the routes which are exposed by the REST API to serve different requests. The routes provided by the analytics server are:

- `@app.route(/createmodel", methods = ['POST'])` parameters: name, description, algorithm, sensorid, time
- `@app.route(/getstatistics")` parameters: model_name
- `@app.route(/getprediction")` parameters: model_name, value
- `@app.route(/getstreamalgorithms")`
- `@app.route(/getbatchalgorithms")`
- `@app.route(/getmodels")`
- `@app.route(/deletemodel", methods = ['DELETE'])` parameters: model_name

Note that name of all the routes and parameters are in lower case letters. It is a good practice as REST routes and parameters are case-sensitive. So, to avoid any confusion, it is recommended to use all lower case characters for naming REST API routes and parameters. Also, the request method type is only defined for “createmodel” and “deletemodel” routes. As the default request method type is “GET”, we do not need to specify the method type for GET requests. The request method type should only be defined for types other than GET.

6.3 Algorithms available on the analytics framework:

There are a numerous number of machine learning algorithms one can use to create and store models. For the thesis, we have chosen four batch processing algorithms and three stream mining algorithms. The algorithms were chosen based on their variety in types of machine learning algorithms. The algorithms supported by the analytics framework are listed below:

6.3.1 Batch Processing:

Linear Regression

Regression [Gan18] is a method of modelling a target value based on predictors. This method is generally used for forecasting and finding out cause and effect relations between various variables. Regression techniques mostly differ based on the number of independent variables and type of relationship between the variables. Linear regression is a type of regression in which the number of independent variables is one and there is a linear relationship between independent (x) and dependent (y) variable. Based on the data points, a line is plotted that models the points the best. Linear regression is based on supervised learning. The line can be modelled based on the linear equation below:

$$Y = a_0 + a_1 * x$$

The motive of linear regression algorithm is to find the best values for a_0 and a_1 . The cost function helps us to figure out best possible values for a_0 and a_1 which would provide the best fit line for the data points. Since we want the best values, we convert this problem into a minimization problem where the goal is to minimize the error between the predicted and actual value.

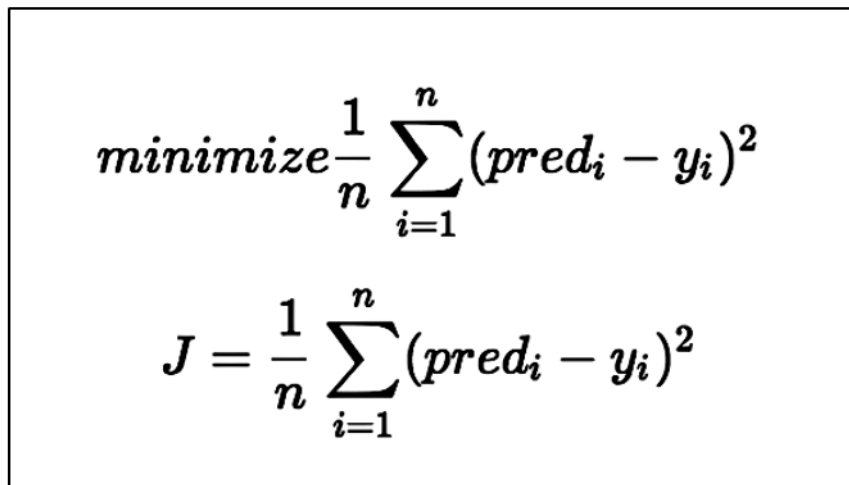

$$\text{minimize } \frac{1}{n} \sum_{i=1}^n (\text{pred}_i - y_i)^2$$
$$J = \frac{1}{n} \sum_{i=1}^n (\text{pred}_i - y_i)^2$$

Figure 6.11: Minimization and Cost function

The difference between predicted values and ground truth measures the error difference. The error difference over all data points in squared and summed and then divided by the total number of points. Therefore, this cost function is also known as Mean Square Error (MSE). Using the MSE value the values of a_0 and a_1 are adjusted so that the minimum value of MSE is obtained.

Decision tree Classification

Decision tree algorithm [Cha19] belongs to family of supervised learning algorithms. Unlike other supervised algorithms, decision tree can be used to solve regression and classification problems. The goal of using a Decision tree is to create a training model that can use to predict the class or value of the target variable by learning simple decision rules inferred from the training data. In decision trees, traversing starts from the root node for predicting a class label. The values of the root attributes are compared with the record's attribute. On the basis of comparison, the corresponding branch to the value is followed. Each node in the tree acts as a test case for some attribute and each edge descending from the node corresponds to the possible answers to the test case. This process is recursive and is repeated for every subtree rooted at the new node.

There are some assumptions made while creating a decision tree. In the beginning, the complete set of data is considered as the root. Feature values are preferred to be categorical. If the values are continuous then they are discretized prior to building the decision tree. Records are distributed recursively on the basis of attribute values. Order to placing attributes as root or internal node of the tree is done by using some statistical methods. The decision of making splits heavily affects the tree's accuracy. Decision trees use multiple algorithms to decide to split a node into two or more sub-nodes. If a dataset consists of N attributes then deciding which attribute to place at the root or at different levels of the tree as internal nodes is a complicated step. Random selection of node to be root is not a solution as it may negatively affect the accuracy of tree. Use of some criteria like Entropy, Information gain, Gini index, Gain ratio, Reduction in variance and Chi-square can help solving the problem. These criterions will calculate values for every attribute. The values are then sorted and the attribute with a higher value is placed at the root.

K Means Clustering

Clustering [LVV03] is one of the most common exploratory data analysis technique used to get an intuition about the structure of the data. It can be seen as the task of identifying subgroups in the dataset such that data in the same subgroup are similar while data in other subgroups are different. These subgroups are called clusters. The decision of which similarity measure to use is application specific. Clustering is considered an unsupervised learning method as we do not have the ground truth to compare the output of clustering algorithms to evaluate its performance. We only try to investigate the structure of the dataset by grouping data points into distinct clusters.

K Means clustering is an iterative algorithm that tries to partition the data into K distinct clusters where each point belongs to only one group. It tries to make the inter-cluster data points as similar as possible and also keeps the clusters as far as possible. It assigns data points to a cluster such that the sum of squared distance between the data points and the cluster's center is minimum. This algorithm initializes the centers by shuffling the dataset and then randomly choosing K data points as cluster centers. Then, it keeps iterating until

there is no change in the cluster centers. When new data points are added to the cluster, the center is recalculated by taking the average of all the data points in that cluster. Since K means clustering use distance based measurements to determine the similarity, it is recommended to standardize the data to have a mean of zero and standard deviation of one since mostly the features in a dataset would have different units of measurement. K Means clustering algorithm is very popular and is used in a variety of applications like market segmentation, image segmentation, document clustering, etc.

FP growth (Frequent Pattern Mining)

FP-growth algorithm [Bor05] is one of the fastest algorithms for frequent pattern mining. It is based on a prefix tree representation of the dataset called a frequent pattern tree or FP tree, which saves considerable amount of memory for storing the transactions. This tree structure will maintain the association between the itemsets. The database is split using one frequent item. This fragmented part is known as “pattern fragment”. The items of these fragmented pattern are analyzed. Thus with this method, the search for frequent items in comparatively reduced. Similar to other algorithms for frequent pattern mining, FP growth preprocesses the database as: an initial scan of frequencies of the items (support). All items with a lower support than the user-defined minimum support as discarded, since they can never be part of a frequent itemset. The next step is to construct the FP tree. The root of the tree is represented as null. The next step is to scan the database. The item with maximum count is taken at the top, the next item with lower count and so on. The branches of the tree is constructed with items in descending order of count. Although the algorithm does not depend on this order, various experiments have shown that it leads to slower execution time than a random order. An ascending order of items leads to the worst execution time. Since FP tree is built top down, the parent is already know when the children are created. Thus, it can be passed down in the recursion where the parent pointers of children are set directly. As a consequence, the nodes of the FP tree can be kept very small. The nodes contains only fields for item identifier, counter, pointer to parent node and pointer to child node.

After obtaining the FP tree, additional pruning needs to be carried out to simplify the tree which speeds up the projections. Pruning is achieved by traversing the levels of the FP tree from top to bottom. The processing starts at the level following the first level that has a support less than minimum support. This level and the following ones are traversed and for each node the first ancestor with an item having sufficient support is determined. The parent pointer is updated to this ancestor, removing the nodes corresponding to infrequent items. If neighboring nodes receive the same parent, they are merged. There are many advantages of the FP growth algorithm over the Apriori algorithm for mining frequent patterns. FP growth needs to scan the dataset only twice. Also, pairing of items is not done in this algorithm which makes it faster. However, it requires high computing power as it stores the tree in main memory. When the dataset is large, FP tree requires huge main memory.

6.3.2 Stream Mining:

K Means Clustering

Clustering of streaming data [AHWP07] has been one of the most studied data mining technique, significantly the K means method. The objective is minimizing the average distance from data points to the center of the cluster they belong to. The proposed algorithm performs a single scan of the data and uses small spaces. It requires $O(nk)$ time where k is the number of centers and n is the number of data points. The algorithm starts with clustering the dataset into $2k$ according to the memory available, and then on the second level it clusters the $2k$ clusters into k clusters. Data Streams require online mining, where data mining should be performed in a continuous way. Further, the system should also have the capability to perform offline analysis based on user requirements. The clusters are stored at particular moments in time known as snapshots.

When a new data point arrives, it is either absorbed by an existing cluster, or a new cluster of its own is created. The first preference is to absorb the data point into currently existing clusters. However, in cases where the point is an outlier, or beginning of a new cluster due to evolution of the stream, it does not belong to any of the existing clusters. In these cases, a new cluster is created. For making this decision, the cluster feature vector is evaluated to see if the point lies outside the maximum boundary of the cluster. However, the number of clusters is fixed, so to create a new cluster the number of existing clusters must be reduced to create space in memory. This can be done by either deleting a cluster, or merging two clusters. The algorithm first analyzes whether it is safe to delete any of the current clusters as outliers, if not then the two clusters are merged. Sometimes it might be tempting to delete the cluster with the least number of data points in it. However, this might often lead to wrong and misleading results. In many cases, especially in the cases of concept drift, a given cluster would be a point of interest in the past history of the stream, but no longer be an active cluster in the current stream.

An ideal goal would be to estimate the average timestamps of the last m points arriving in each micro cluster. However, this increases the memory requirement by a factor of m . Such an implementation would result in a decrease in the number of clusters stored as the space available for them would be less. Hence, effectiveness of the algorithm would be reduced. The average timestamp can be calculated by using data about the timestamps of points stored in a cluster M . We note that the timestamp data allows the calculation of the mean and standard deviation of the arrival time of data points in M . When the smallest such timestamp of a micro-cluster is below the user defined threshold, this shows that the cluster was not updated with new data points for a long time and hence, it can be eliminated and a new cluster with unique id with the new point can be formed. In some cases, no one of the micro clusters can be readily deleted. This happens when the relevancy timestamp of all clusters are sufficiently recent and above the defined threshold. In such cases, the clusters closest to each other are merged with no longer has an id, but an id list with ids of the merged clusters. Thus, merged clusters can be identified later if needed.

KNN Classification

KNN stands for K nearest neighbors. The KNN classification [RC18] selects K nearest training data samples and then predicts the test sample with the major class among k nearest training samples. Data streams are assumed to be potentially infinite, with instances arriving in some order and in rapid succession. Because of this, a classifier has a limited amount of time in which to process each instance and will never have access to all instances simultaneously. KNN stream classification updates and adapts a short term memory so it contains only the current concept, while also retaining a record of all past concepts in a long term memory. New data points are added to the short term memory, which reduces the size whenever there is a concept drift. This is done by evaluating multiple window sizes and retaining the one with minimum test-train error. Both the short term and long term memory induce a classifier. The final prediction is made by the memory with higher current average accuracy. The classifiers are ranked by their Hamming score, the final prediction is made by the classifier with the highest. Using two memories, KNN classifier takes advantage of the short term sliding window to react quickly to abrupt drift while also having the option to fall back to long term memory in cases of recurring drifts. KNN classifier adopts a lazy learning approach that is it incrementally updates the model with addition of each new instance.

Hoeffding Tree Classification

A decision tree is a directed acyclic graph in which each node is either a decision node with two or more successors or a leaf node. Therefore, a decision node has some condition based on attribute values, and the leaf nodes are labeled with a class label. Learning decision trees from data streams is one of the most difficult problems in the data mining community. The basic idea of the Hoeffding tree system comes from the observation that a small number of examples are sufficient enough to select the correct splitting test and split a leaf. Hoeffding tree algorithms can manage a huge amount of data using less computational resources with a performance similar to a batch decision tree, given enough examples.

In Hoeffding tree classification [Gam10], a decision tree is learned by replacing leaf nodes with decision nodes recursively. Each leaf stores sufficient statistics about the attributes. The sufficient statistics are those needed by the heuristic evaluation function that evaluates the merit of split tests based on the attribute values. When a data point is available, it traverses the current tree from root to leaf, evaluating the appropriate attribute at each node and following the relevant branch corresponding to the attribute values in the data point. When the data reaches the leaf, the required statistics are updated. Then, each possible value based on the attribute-value is evaluated. If there is enough support in favor of one of the tests over all the others, the leaf is transformed to a decision node. The new decision node will have as many descendent leaf nodes as the number of possible values of the test chosen previously. The decision nodes only store the information about the split test at that node. Hoeffding bound is used for deciding sample size to observe before converting any

leaf node into a decision node. The evaluation of Hoeffding bound for every data sample would be very costly. Hence, it is not efficient to compute it every time a new data point arrives. The algorithm computes the attribute evaluation function only when a minimum number of examples has been observed since the last evaluation. This minimum number of examples is a user-configurable parameter.

6.4 Storing Analytics Models:

After a model is created, it needs to be stored as a file for further usage. There are many ways to store a model and every way has its own advantages and disadvantages. In this thesis, we will be using PMML and Pickle files to store the machine learning models.

6.4.1 PMML (Predictive Model Markup Language):

PMML [GZLW+09] package is used to export a variety of predictive and descriptive models to a file. PMML is an XML-based language and has become the standard to represent not only predictive and descriptive machine learning models, but also the data pre and post processing. It allows for the interchange of machine learning models among different tools and environments, avoiding proprietary issues and incompatibilities. PMML provides users with an open standard representing data mining models. This enables users to share saved models between different applications and environments. Not only PMML can represent a wide range of statistical techniques, but it can also represent the input data as well as the transformations necessary to transform the data. PMML follows a very intuitive structure to describe a data mining model.

The overall structure of PMML can be described by the following elements:

- **Header:** The header element contains the general information about the PMML document, such as description, copyright information, name and version. It also contains an attribute for a timestamp which can be used to specify the date of creation of the model.
- **Data Dictionary:** The data dictionary element contains a definition of all the data fields used by the model. It is in the data dictionary that a field is defined as categorical, continuous or ordinal. The data dictionary is also used for describing a list of missing, valid and invalid values relating to the input data.
- **Data Transformation:** Transformation allows for the mapping of user data into a more appropriate form to be used by the data mining model. PMML defines several kinds of data transformations like Normalization, Value mapping, Aggregation etc.

- **Model:** The Model element contains the definition of the data mining model used. Models usually have a model name, function name (regression, classification) and technique-specific attributes. Model representation begins with a mining schema and continues with the actual representation of model. The actual representation of a model contains Mining schema, targets and Model specifics.

6.4.2 Pickle File:

The one major drawback of PMML files is that not every type of model can be saved as a PMML file. Moreover, a user may want to save more statistics about a model than a PMML file saves. For cases like this, using a “Pickle” file is ideal. The Python pickle module [Fou20] is used for serializing and de-serializing an object structure in Python. The concept of serialization is simple. Users have a data structure in memory that they want to save, reuse or send to another application. In situations like these, use of pickle files makes the storage of files easy. Pickle serializes the object to be stored before writing it to a file. The pickle module implements binary protocols for serializing and de-serializing. Pickling is a way to convert objects into a byte stream. The idea behind pickling is that this byte stream contains all the information necessary for the reconstruction (unpickling) of the object.

A pickle file can be used to store practically anything:

- All native datatypes like Boolean, Integers, Strings, Floating point numbers etc.
- List, tuples and sets containing any combination of native datatypes.
- Functions, classes and instances of classes.

Usage of pickle files help us to eliminate the limitations of PMML files. Any machine learning model of choice can be stored, with any desired statistics and metadata about the model. The biggest advantage of using a pickle file is that it is highly customizable and space-efficient.

7 Validation and Discussion

In the previous chapter, we discussed the implementation details of the analytics framework. We saw how users can view existing models, create a new model and use models to make predictions. We also saw, what kinds of models are offered by the analytics framework and what algorithms are available to create models. In this chapter, we will be discussing some real world scenarios to show the use of analytics in IoT and how this analytics framework with MBP enables achieving these real world goals. Smart cities use IoT devices, such as connected sensors, lights and meters to analyze data. Data analytics on this data can be then used to improve infrastructure, utilities, services and more. Industrial IoT is used across several industries, such as manufacturing (Industry 4.0), logistics, oil, aviation, transportation and other industrial sectors and the use cases which are typical for these industries. As of now, be it batch or stream processing, an analysis can only be performed on a single sensor data. Data merging from multiple sensors, or collecting different statistics about same machine from different sensors is not supported.

7.1 Modeling energy usage in Smart Cities:

Migration from rural to urban areas leads to the emergence of urbanization and sustainability problems. Management and monitoring of resources and infrastructures are getting more important today in cities. Energy consumption is increasing with the growing population in cities and this results into high energy demands. Therefore, energy efficiency and planning is becoming a challenge in bigger cities. Smart cities and smart systems aims at improving city life and making it more efficient. We present an idea how IoT analytics can be used to predict electricity usage and help in planning and modeling of electricity usage. This is a great example of how batch processing can be used in smart cities by collecting the data from different houses.

In a Smart City, every house can be equipped with a sensor which reports the electricity usage of that house. Also, metadata about the house, like size of house, number of power outlets, number of stories, age of house, etc. can be stored. Each sensor has a SensorID which is used to identify which house the sensor belongs to. This data can be used to create a predictive model which can be further used to predict the amount of energy used by a particular house. This model can also be used to predict the energy requirements of any new house to be constructed based on the parameters of that house. This idea would help in better planning and modeling of energy usage in Smart cities. To show the implementation

of this model, a simulation of data was created and then, based on that, a Regression model based on size of house and number of power outlets in the house. So, according to the data collected and the model created, a house of size 90 square meters and having 10 power points will consume 42.085 units of electricity.

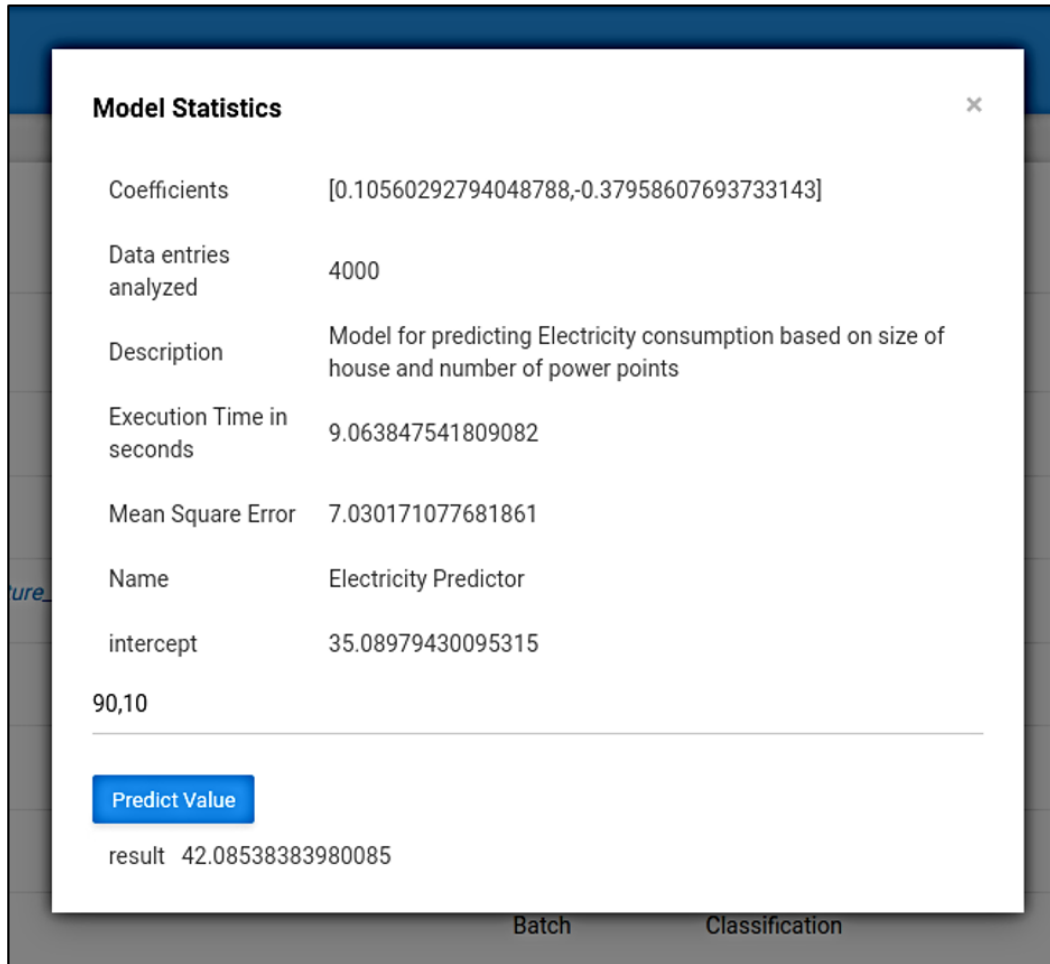


Figure 7.1: Predicted Electricity consumption value

Based on the model, energy consumption plans can be made, houses can be reported whether their energy usage in normal, above or below the average energy usage. House owners can be made aware about what should be their energy consumption and what the actual consumption is. Efficiency ratings can be given to buildings. Connecting more home devices will help, e.g., functions like dimming of lights or switching them off or modifying multiple settings will be possible. Every house can mark some devices as “important” or “critical”, which guarantees power supply at all times. Power supply to other devices can be stopped if needed. Hence, a complete blackout can be avoided in cases of power failure or any other problem. So, we can see here that merely collecting power consumption data and predicting usage can lead to many benefits in planning and modeling energy usage in

smart cities. This is just the tip of the iceberg, there are many other use cases which can be implemented in smart cities to model energy usage in a better way and will result in numerous benefits for both the house owners and the energy providers.

7.2 Predictive maintenance in Smart Industries:

Predictive maintenance [LS19] relies on real time monitoring and diagnosis of system components, production chains and processes. Predictive maintenance is a classic case of the use of stream mining in IIoT (Industrial IoT). The primary strategy is to take action when some parts or items show certain behavior that usually results in machine failure, downtrend in quality or degraded performance. Initially, predictive maintenance was motivated by the execution of system checks at predefined intervals to analyze the health of equipment. But, during recent years, the emergence of data stream mining provides the capability of analyzing data as it is generated and eliminates the chances of missing an anomaly in machine behavior which was very common missed in periodic analysis. This makes predictive maintenance easier and more effective. In this sense, it became an essential component of Industry 4.0 applications and environment. The goal of predictive maintenance in all kinds is to recognize untypical system behavior or undesired trends at an early stage. Wear and tear of any equipment is unavoidable, predictive maintenance helps in predicting at an early stage the date of failure or date when the performance goes below the required quality standards. This helps in planning of maintenance and outages by a company. The impact of random outages can be really costly for a company, predictive maintenance helps in reducing the amount of unplanned outages by a considerable amount. This helps companies perform their business in a better and smoother way.

Sensors on different machines and equipment needs to be installed to collect statistics (ex: Temperature, performance, current, intensity of vibration, efficiency, etc.). Using these statistics, models can be created which can detect anomalies and predict the point in time when these machines require a maintenance before the performance goes below an expected level of quality. A regular fault detection problem is a binary classification problem which aims to predict whether a system state (statistics at a particular point in time) corresponds to faulty or a fault-free state. The analytics framework provides the option to create a stream classification algorithm (Stream K-nearest neighbors) which can be used for this purpose. This classification can also be implemented as a scale of system state (from good to critical) to predict the health of a machine based on various statistics collected from it. In the example below, we used a KNN prediction model to predict the state of an equipment. We used the parameters current, temperature and efficiency to make predictions about the state of the equipment. Using the model, we can predict the state of the equipment (on a scale Good (1) to Critical (5)). So, according to the model, the equipment with 23 units of current, temperature of 34 degree Celsius and efficiency of 65% is in a good (status:1) condition.

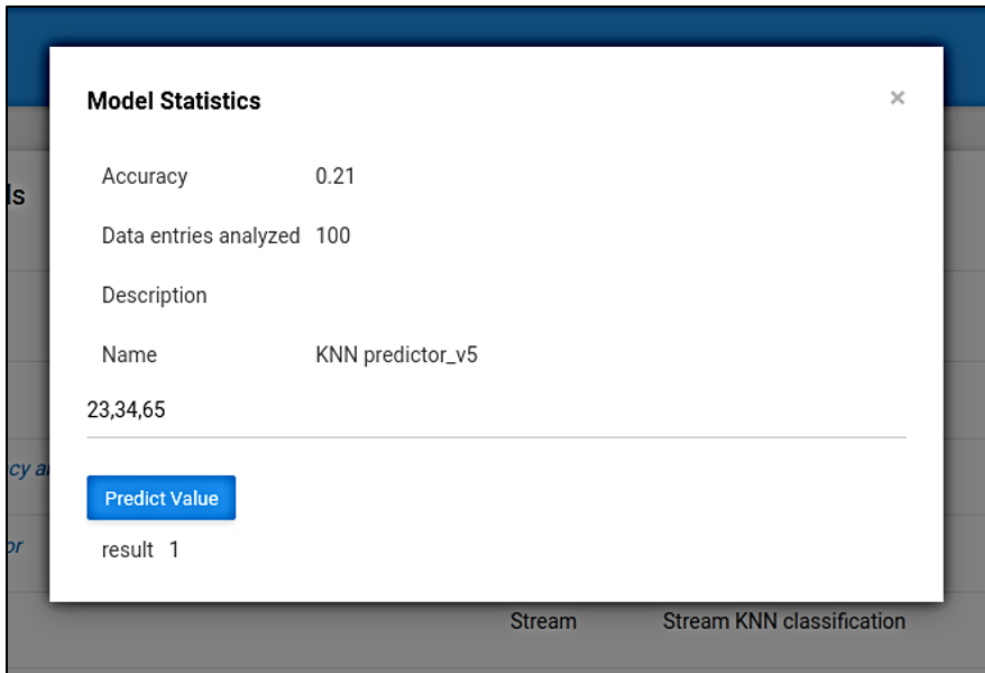


Figure 7.2: Predicted Status of the device

7.3 Descriptive analysis of components in Smart Industries:

Descriptive analysis is a basic form of analytic insight, but is useful in many scenarios. They allow users to describe and aggregate incoming data. Even calculations as simple as mean and standard deviation for a large data set can make sense. In a connected factory use case, descriptive analytics might be used to answer many questions. Finding frequent patterns between different parameters, like temperature, flow rate, RPM, current etc. can help answering the question “What scenario leads to what result?”. Using sensor data reporting temperature, current and efficiency of a machine, we created a frequent pattern mining model using the analytics framework.

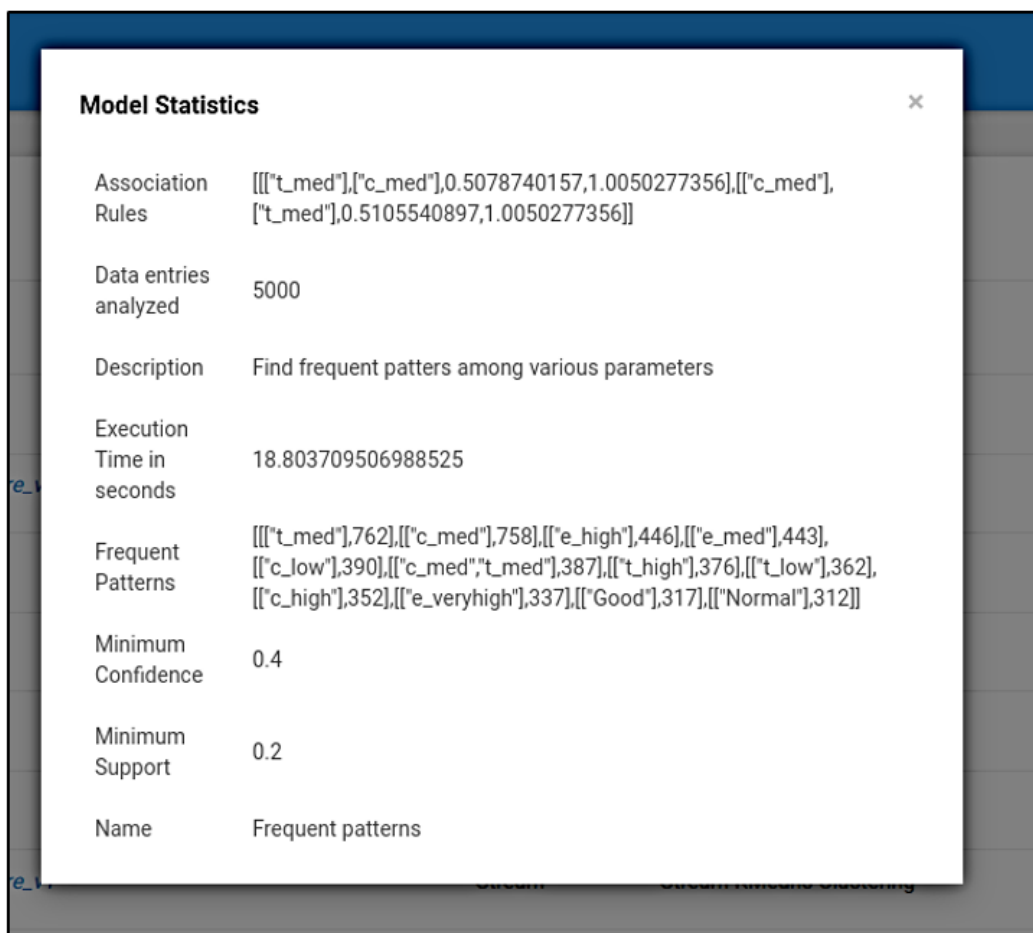


Figure 7.3: Finding frequent patterns between different parameters

The results show the association rules between “t_med and c_med” with a confidence of 0.5 and “c_med and t_med” with confidence of 0.51. These results clearly show that there is a strong relation between the current and temperature of the equipment. This analysis will be helpful in analyzing the problems caused by high temperatures in the equipment.

As we saw above, there are many real world use cases of analytics in IoT. The above presented scenarios are just few of the many possibilities of using analytics in IoT. IoT data opens the door for numerous opportunities for making better changes in smart environments. Data management, modeling and analytics tools are the core enablers of these opportunities. IoT itself is making the life in smart cities, industries and other field far better than what it used to be. Using analytics with IoT can unlock many possibilities and usages which we have never imagined. Analytics dramatically increase the visibility and understanding of usage and behavior in smart environments. We have now entered the age where the everyday objects and home appliances communicate with and control each other. Analyzing the data generated and understanding the behavior has become really important. A wide range of sensors, machine and devices generate massive amounts of data, requiring a whole new

level of data modeling and analysis. The above use cases represent some of the scenarios where integration of MBP with the analytics framework will help producing models which will turn out to be useful in many scenarios.

8 Conclusion

In this thesis, we have created an analytics framework for the MBP IoT platform. MBP is a platform which enables automatic provisioning and configuration of devices in the Internet of Things (IoT). In the emerging so-called “smart” world of IoT, the integration of devices, sensors, and actuators has become more and more important. IoT benefits from data coming from a huge number of connected devices, which helps in creation of these smart environments. MBP itself makes the collection and management of huge data from different sensors and devices easier. After adding data analytics capabilities to this platform, the possibilities are limitless. Moreover, the development of analytics frameworks for different IoT applications is urgently needed. We envision that this thesis is offered as a mere baby step in a potentially fruitful direction. In this thesis, we showed how analytics can be used in IoT and how can one build a framework which works seamlessly with the IoT management platform MBP. We also showed some real-world application of the framework to show how one can use the results of analytics for creating smart environments. For example, applications of the analytics results on data acquired can include activity recognition to identify health problems, identify energy consumption patterns and energy saving planning and predicting equipment maintenance to avoid unplanned outages and expensive repairs and ensure efficient operations of various industries.

In general, the analytics framework can aid effective and in-time decision-making for individual applications by creating and using models based on the data generated by the devices. Creating machine learning models is a complex process that requires continuous fetching of data from the devices which have easy access to data. The problem of fetching and integrating data from devices and sensors is solved by the MBP platform and then this data is used by the analytics platform to create both batch processing and stream mining models.

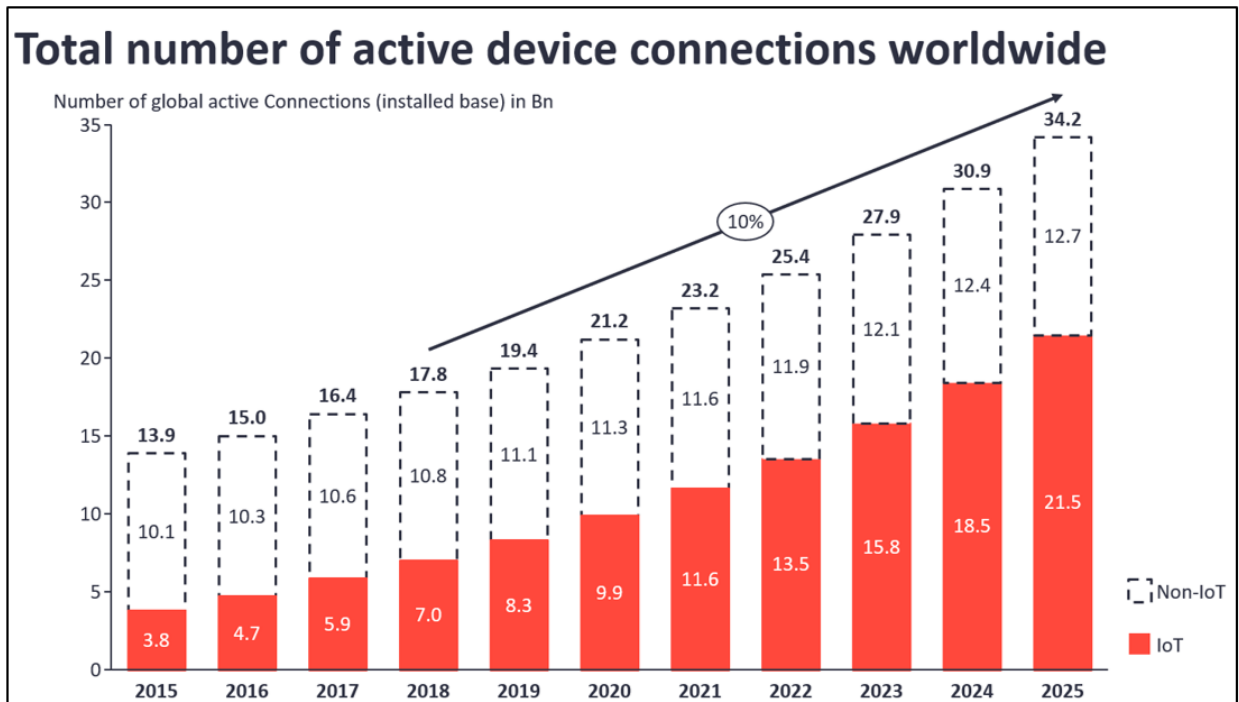


Figure 8.1: Total number of connected devices with share of IoT devices [Lue20]

As we can see in the image above, the share of IoT devices is increasing rapidly among all the connected devices around the world. These devices create data at a never imagined speed. GBs of data is being generated by a single device per hour. Storing and managing this amount of data is really hard. A huge amount of storage is needed to store the data from devices. However, the raw data is useless for a user until we can derive insights from it. This is where analytics comes into the picture. Data can be stored for a shorter period of time (few days to a week) and a batch processing model can be created using the data. In cases where data is too much to handle, stream mining can be used where data would be analyzed on the fly and then the user may or may not choose to store the complete data. That is one of the many benefits of stream data mining. The analytics platform provides a lambda architecture, enabling both batch and stream processing capabilities which gives users the choice of mining models they would prefer to make for different sensors and devices. Also, users can make both batch and stream mining models for the same sensor and then also compare the results. This will assist the users to choose which method to choose in which scenario.

With the number of increasing devices resulting into a huge amount of data, it is clear that the importance of data mining in IoT environments will be increasing. However, more devices and more data also bring more challenges and difficulties in data mining. As the world is evolving into a digitally smarter connected platform, the challenges in data mining will also be increasing. However, these challenges can be addressed using better methodologies, better data cleaning tools and better processing resources. In this thesis,

the use of lambda architecture helps us in reaping the benefits of both batch processing and stream mining of data which gives users the flexibility of choosing what kind of data mining model they want. The analytics framework provides the functionalities of creating a model, viewing statistics about a model, using a model to make predictions and deleting a model. All the functionalities integrated in the existing MBP user interface enables the existing MBP user to access the analytics framework and use the functionalities as desired. As IoT becomes more integrated into our daily life, data analytics has become really important in helping users draw key insights without having to do any of the heavy lifting. Data analytics is no longer an add-one, but an integral part of any IoT solution. The full capability of IoT in the real-world use lies in applying analytics to the data generated and using the knowledge to create a smarter, better and more efficient world.

9 Future Work

In this thesis, we presented how analytics can be used with the MBP IoT platform to create models and derive insights from raw sensor data. However, the analytics framework is just a prototype showing the implementation and use of analytics in IoT. The IoT and the new world of smart systems are ushering into an era where people, devices, machines, sensors and businesses are all connected and able to interact with each other. Data management and analytics tools are the core enablers of these new opportunities. Without analytics, users can just scratch the surface of the real value of the huge data created by the IoT devices. We created an analytics framework for the MBP platform with basic functionalities in this thesis, but many different adaptations, tests and scenarios have been left for the future. In this chapter, we will discuss and present some of the ideas which can be worked on in the future.

In this thesis, we used Apache Spark as a Batch processing engine. It uses a query optimizer to achieve high performance. In this thesis, we have included four machine learning algorithms. However, Apache Spark provides a lot of other algorithms with easy implementations, which can be used to create various different kinds of models. Also for stream data mining, new algorithms and models can be integrated in the analytics framework. There are no limitations to the framework, so new analytics engines can also be integrated into the same analytics framework to provide more options to the user for creating models. In the thesis, we created models with minimal parameter tuning. Machine learning models are as good as their accuracy. A model which is not accurate enough cannot be trusted to make business decisions and hence, would not be of great use. So, more focus on the accuracy of models can be put to ensure high quality modeling. This can be done by tuning the various parameters used by the models. Also, more options on the user interface can be given to the user for creating models. Making options like number of clusters to be created for clustering, minimum support and minimum confidence for finding frequent patterns configurable, would help users to create personalized models according to their needs. There are many parameters which every algorithm uses for creating models. These parameters are user-configurable and would make the models more accurate and also user centric. To do this, the user interface would need to be extended and adapted for different algorithms to provide input fields for various parameters. Adding these functionalities would make the analytics framework more useful and ready for the real world.

In addition to the statistics collected about the models by the analytics framework, more statistical parameters can be added to the models for better understanding and benchmarking. For any system, benchmarking plays a very important role. Benchmarking is a sensible exercise to establish baselines, define best practices and identify improvement opportunities.

It helps in gaining an independent perspective about how well the system performs. Some statistics like data size or execution time are already recorded by the analytics framework for every model. More parameters like CPU utilization or memory requirements can also be added to the set of statistics which would help in benchmarking of models under different circumstances.

In addition to creating models, integrating an alert system in the analytics framework would also be very useful. Reporting of events is always a good idea. Various events on the analytics framework like finishing model creation, model successfully created, model creation failed etc. can be reported to the relevant user. Also, events like an anomaly detection could also be reported to the user. Alerts can be sent to users in various forms. Alerts can be published to a topic which users can subscribe to, an alert board can be created on the user interface where all the alerts are posted, emails can be sent to the relevant user in case of an event or a push notification to the user can also be sent. Reporting enables the user to know what is going on in the system.

Currently, the analytics server supports only data from one particular sensor. When creating a model, only one sensor deployed on MBP can be selected. This limits the usage of the analytics server. Sensors are commonly used stand-alone or in device to collect data and detect changes in the environments. If you happen to be using any kind of smart device like a phone or a watch, you have likely seen how the sensors can be useful in providing valuable insights about your day to day life. Sensors generate data at all times, so it is really important to have an efficient integration system in place to ensure that the data is more useful in deriving insights. The IoT is the future of technology that helps in deriving insights to regulate and understand the things in a considerably stronger way. There are various open-source tools available to join and merge data from different sensors. Integrating such a tool to the analytics framework could be helpful in integrating the data. Apache Kafka is a nice example of such tools. Kafka is really efficient in building real-time data pipelines that reliably get data between systems or applications. It can also be used to perform transformations on the data if required. Also, creating such a tool which merges data from different sources is not a very difficult task. So, some extra code can be written to handle data from different sensors and merge them before sending them for analysis. Another idea could be creating a database for merged data. A subscriber can be created which listens to all the sensors reporting different parameters like temperature, current, pressure of a particular device. When new data is received, it can merge it and store it into a database which contains tables for all connected devices. The columns of the table would be the parameters reported by the device. This way we will have a relational database containing data from different sensors for particular devices. This database can directly be used by the analytics server to create models. Hence, there can be various approaches to joining and merging sensor data which can be adopted, but using them would be really helpful in creating multi-dimensional models, which would help deriving useful insights.

The analytics framework created in this thesis can have a numerous number of use cases. It would serve as a base for more advanced analytics environments. Adapting the analytics framework to IoT systems would help in creating smart environments. The analytics

framework provides REST APIs for interaction to other systems which makes it loosely coupled to other systems. This also makes extending and adapting to the analytics framework easier. As REST is used, one huge analytics server can be used for providing analytics services to many different IoT platforms. There can be a huge number of changes and improvements that can be made to the framework. Above, we listed some of the features which are not very complicated and can be implemented in the future.

References

- [AB18] A. Alavi, W. G. Buttler. *Data Analytics for Smart Cities*. CRC Press, 2018 (cit. on p. 9).
- [AHR14] E. Auschitzky, M. Hammer, A. Rajagopaul. “How big data can improve manufacturing”. In: *McKinsey & Company* 822 (2014).
- [AHWP07] C. C. Aggarwal, J. Han, J. Wang, S. Y. Philip. “On clustering massive data streams: a summarization paradigm”. In: *Data Streams*. Springer, 2007, pp. 9–38 (cit. on p. 47).
- [AKK+16] M. Akcin, A. Kaygusuz, A. Karabiber, S. Alagoz, B. B. Alagoz, C. Keles. “Opportunities for energy efficiency in smart cities”. In: *2016 4th International Istanbul Smart Grid Congress and Fair (ICSG)*. IEEE. 2016, pp. 1–5.
- [Ali15] AlixPartners. *Internet of Things: New Challenges and opportunities for semiconductor supply chain*. <https://www.alixpartners.de/veroeffentlichungen/insights/internet-of-things-new-challenges-and-opportunities/>. 2015 (cit. on pp. 9, 19).
- [ASF16] The Apache Software Foundation. *Apache ODE™ – The Orchestration Director Engine*. 2016. URL: <http://ode.apache.org>.
- [AZR+17] A.-R. Al-Ali, I. A. Zualkernan, M. Rashid, R. Gupta, M. Alikarar. “A smart home energy management system using IoT and big data analytics approach”. In: *IEEE Transactions on Consumer Electronics* 63.4 (2017), pp. 426–434 (cit. on p. 24).
- [BGHP18] A. Bifet, R. Gavaldà, G. Holmes, B. Pfahringer. *Machine learning for data streams: with practical examples in MOA*. MIT Press, 2018 (cit. on p. 27).
- [Bif10] A. Bifet. “Adaptive stream mining: Pattern learning and mining from evolving data streams”. In: *Proceedings of the 2010 conference on adaptive stream mining: Pattern learning and mining from evolving data streams*. Ios Press. 2010, pp. 1–212 (cit. on p. 14).
- [Bor05] C. Borgelt. “An Implementation of the FP-growth Algorithm”. In: *Proceedings of the 1st international workshop on open source data mining: frequent pattern mining implementations*. 2005, pp. 1–5 (cit. on p. 46).
- [Cha19] N. S. Chauhan. *Decision Tree Algorithm — Explained*. <https://towardsdatascience.com/decision-tree-algorithm-explained-83beb6e78ef4>. 2019 (cit. on p. 45).

References

- [DH00] P. Domingos, G. Hulten. “Mining high-speed data streams”. In: *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*. 2000, pp. 71–80.
- [Dud01] Duden. *Die Deutsche Rechtschreibung*. 22., völlig neu bearbeitete und erweiterte Auflage. Aktualisierter Nachdruck. Dudenverlag, 2001. ISBN: 3-411-04012-2.
- [DWW13] G. Ding, L. Wang, Q. Wu. “Big data analytics in future internet of things”. In: *arXiv preprint arXiv:1311.4112* (2013) (cit. on p. 9).
- [Fou20] P. S. Foundation. *pickle — Python object serialization*. <https://docs.python.org/3/library/pickle.html>. 2020 (cit. on p. 50).
- [Gab12] M. M. Gaber. “Advances in data stream mining”. In: *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 2.1 (2012), pp. 79–85.
- [Gam10] J. Gama. *Knowledge discovery from data streams (Chapter 8)*. CRC Press, 2010 (cit. on p. 48).
- [Gan18] R. Gandhi. *Introduction to Machine Learning Algorithms: Linear Regression*. <https://towardsdatascience.com/introduction-to-machine-learning-algorithms-linear-regression-14c4e325882a>. 2018 (cit. on p. 44).
- [Gar13] N. Garg. *Apache Kafka*. Packt Publishing Ltd, 2013 (cit. on p. 27).
- [Gri18] M. Grinberg. *Flask web development: developing web applications with python*. Ö’Reilly Media, Inc.”, 2018 (cit. on p. 28).
- [GZK05] M. M. Gaber, A. Zaslavsky, S. Krishnaswamy. “Mining data streams: a review”. In: *ACM Sigmod Record* 34.2 (2005), pp. 18–26.
- [GZLW+09] A. Guazzelli, M. Zeller, W.-C. Lin, G. Williams, et al. “PMML: An open standard for sharing models”. In: *The R Journal* 1.1 (2009), pp. 60–65 (cit. on p. 49).
- [Han06] D. J. Hand. “Data Mining”. In: *Encyclopedia of Environmetrics* 2 (2006) (cit. on p. 13).
- [Han98] D. J. Hand. “Data mining: statistics and more?” In: *The American Statistician* 52.2 (1998), pp. 112–118 (cit. on p. 10).
- [HBS+16] P. Hirmer, U. Breitenbücher, A. C. F. da Silva, K. Képes, B. Mitschang, M. Wieland. “Automating the Provisioning and Configuration of Devices in the Internet of Things.” In: *CSIMQ* 9 (2016), pp. 28–43 (cit. on p. 16).
- [Heb08] G. Hebrail. “Data stream management and mining”. In: *Mining massive data sets for security* (2008), pp. 89–102.
- [HTS08] U. Hunkeler, H. L. Truong, A. Stanford-Clark. “MQTT-S—A publish/subscribe protocol for Wireless Sensor Networks”. In: *2008 3rd International Conference on Communication Systems Software and Middleware and Workshops (COMSWARE’08)*. IEEE. 2008, pp. 791–798 (cit. on pp. 28, 29).

- [ILG07] E. Ikonomovska, S. Loskovska, D. Gjorgjevik. “A survey of stream data mining”. In: *Proceedings of 8th National Conference with International participation, ETAI*. 2007, pp. 19–21.
- [JM17] T. John, P. Misra. *Data Lake for Enterprises*. Packt Publishing Ltd, 2017.
- [KAST15] Z. Khan, A. Anjum, K. Soomro, M. A. Tahir. “Towards cloud based big data analytics for smart future cities”. In: *Journal of Cloud Computing* 4.1 (2015), p. 2.
- [KK12] Z. Khan, S.L. Kiani. “A cloud-based architecture for citizen services in smart cities”. In: *2012 IEEE Fifth International Conference on Utility and Cloud Computing*. IEEE. 2012, pp. 315–320.
- [Kle17] S. Klein. *IoT Solutions in Microsoft’s Azure IoT Suite*. Springer, 2017 (cit. on p. 21).
- [KMM+15] M. Kiran, P. Murphy, I. Monga, J. Dugan, S. S. Baveja. “Lambda architecture for cost-effective batch and speed big data processing”. In: *2015 IEEE International Conference on Big Data (Big Data)*. IEEE. 2015, pp. 2785–2792 (cit. on p. 14).
- [KY+18] K. Kumari, S. Yadav, et al. “Linear regression analysis study”. In: *Journal of the Practice of Cardiovascular Sciences* 4.1 (2018), p. 33.
- [LGS17] P. Lade, R. Ghosh, S. Srinivasan. “Manufacturing analytics and industrial internet of things”. In: *IEEE Intelligent Systems* 32.3 (2017), pp. 74–79 (cit. on p. 23).
- [LS19] E. Lughofer, M. Sayed-Mouchaweh. *Predictive Maintenance in Dynamic Systems: Advanced Methods, Decision Support Tools and Real-World Applications*. Springer, 2019 (cit. on p. 53).
- [Lue20] K. L. Lueth. *Number of connected devices*. <https://iot-analytics.com/state-of-the-iot-update-q1-q2-2018-number-of-iot-devices-now-7b/>. 2020 (cit. on p. 58).
- [LVV03] A. Likas, N. Vlassis, J. J. Verbeek. “The global k-means clustering algorithm”. In: *Pattern recognition* 36.2 (2003), pp. 451–461 (cit. on p. 45).
- [MAS10] M. MOHAMED, Z. Arkady, K. Shonali. “Data stream mining”. In: *M. Oded, R. Lior. Data Mining and Knowledge Discovery Handbook*. New York: Springer (2010), p. 761.
- [MRBA18] J. Montiel, J. Read, A. Bifet, T. Abdessalem. “Scikit-multiflow: a multi-output streaming framework”. In: *The Journal of Machine Learning Research* 19.1 (2018), pp. 2915–2914 (cit. on p. 26).
- [Nas11] H. H. Nasereddin. “Stream Data Mining.” In: *IJWA* 3.2 (2011), pp. 90–97.
- [PPLC17] G. Poghosyan, I. Pefkianakis, P. Le Guyadec, V. Christophides. “Extracting usage patterns of home IoT devices”. In: *2017 IEEE Symposium on Computers and Communications (ISCC)*. IEEE. 2017, pp. 1318–1324.

References

- [PW09] M. Pilgrim, S. Willison. *Dive Into Python 3*. Vol. 2. Springer, 2009.
- [Raj18] U. R. Rajeev Srinivasan. “Amazon Web Services – Lambda Architecture for Batch and Stream Processing”. In: (2018) (cit. on p. 20).
- [RC18] M. Roseberry, A. Cano. “Multi-label kNN classifier with self adjusting memory for drifting data streams”. In: *Second International Workshop on Learning with Imbalanced Domains: Theory and Applications*. 2018, pp. 23–37 (cit. on p. 48).
- [RRPH16] J. Read, P. Reutemann, B. Pfahringer, G. Holmes. “Meka: a multi-label/multi-target extension to weka”. In: *The Journal of Machine Learning Research* 17.1 (2016), pp. 667–671 (cit. on p. 27).
- [RVA16] H. Reijers, I. Vanderfeesten, W. van der Aalst. “The effectiveness of workflow management systems: A longitudinal study”. In: *International Journal of Information Management* 36.1 (Feb. 2016), pp. 126–141. DOI: [10.1016/j.ijinfomgt.2015.08.003](https://doi.org/10.1016/j.ijinfomgt.2015.08.003).
- [Sti15] C. L. Stimmel. *Building smart cities: analytics, ICT, and design thinking*. Auerbach Publications, 2015.
- [TAG+17] P. Ta-Shma, A. Akbar, G. Gerson-Golan, G. Hadash, F. Carrez, K. Moessner. “An ingestion and analytics architecture for iot applied to smart city use cases”. In: *IEEE Internet of Things Journal* 5.2 (2017), pp. 765–774 (cit. on p. 9).
- [TBA+14] R. Tönjes, P. Barnaghi, M. Ali, A. Mileo, M. Hauswirth, F. Ganz, S. Ganea, B. Kjærsgaard, 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 (cit. on p. 10).
- [WCL+05] S. Weerawarana, F. Curbera, F. Leymann, T. Storey, D. F. Ferguson. *Web Services Platform Architecture : SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and More*. Prentice Hall PTR, 2005. ISBN: 0131488740. DOI: [10.1.1/jpb001](https://doi.org/10.1.1/jpb001).
- [WHC+16] J. Wang, J. Huang, W. Chen, J. Liu, D. Xu. “Design of IoT-based energy efficiency management system for building ceramics production line”. In: *2016 IEEE 11th Conference on Industrial Electronics and Applications (ICIEA)*. IEEE. 2016, pp. 912–917.
- [WZ02] M. Wojciechowski, M. Zakrzewicz. “Methods for batch processing of data mining queries”. In: *Proc. of the 5th International Baltic Conference on Databases and Information Systems*. 2002.
- [YSHM19] A. Yassine, S. Singh, M. S. Hossain, G. Muhammad. “IoT big data analytics for smart homes with fog and cloud computing”. In: *Future Generation Computer Systems* 91 (2019), pp. 563–573.

- [ZXW+16] M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin, et al. “Apache spark: a unified engine for big data processing”. In: *Communications of the ACM* 59.11 (2016), pp. 56–65 (cit. on pp. 25, 26).

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

place, date, signature