National and Kapodistrian University of Athens
Faculty of Physics
Department of Electronics, Computers, Telecommunications and Control

Master Thesis
Control and Computing

“Big Data Analytics for the Cloud”

Nikolaos Gkatzios
Student ID: 2015505

Supervisor
Assistant Professor Anna Tzanakaki

Athens, June 2017

Master Thesis in Control and Computing

Title: “Big Data Analytics for the Cloud”

Author: Nikolaos Gkatzios

Student ID: 2015505

Supervisor: Assistant Professor Anna Tzanakaki

Evaluation Committee: Assistant Professor Anna Tzanakaki,

Associate Professor Dionysios I. Reisis,

Associate Professor Hector E. Nistazakis

Master Thesis submitted June 2017

Key words: Big Data, Cloud, IoT, Clustering, Forecasting

ABSTRACT

This master thesis comprises three distinct parts. The first part includes an introduction to the Big Data challenge, the description of the Big Data management problem as well as an overview of existing architectures and algorithms addressing this problem. The second part presents processing of a real dataset that is formed from measurements collected by sensors installed on trains. A study of four different clustering algorithms, namely KMeans, Birch, Mean Shift, DBSCAN, is conducted together with an investigation on the performance, of two different Neural Networks (MLP, LSTM), on forecasting. The latter involves implementation of a specific solution focusing on Big Data from IoT devices using Sitewhere, an open source IoT platform.

ACKNOWLEDGEMENTS

I would like to express my gratitude to Professor Anna Tzanakaki, my thesis supervisor, for her guidance and encouragement. The door to Prof. Tzanakaki’s office was always open whenever I had a question about my research or writing. She allowed this paper to be my own work, while with her useful critiques steered me in the right direction.

I would also like to extend my thanks to Dr. Markos Anastasopoulos for his general advices and his suggestions concerning the way I should approach certain aspects of this research work.

Finally, I wish to thank my parents for their continuous support throughout the process of researching and writing this thesis. This accomplishment would not have been possible without them.

Author
Nikolaos Gkatzios

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Big Data is a relatively new term used for datasets, so large or complex, that render commonly used software tools and traditional data processing methodologies inadequate to manage them. Therefore, in order to be able to extract useful information from these large, diverse, and complex datasets, new mining techniques are required.

The work for this master thesis is divided into three parts. The first part focused on the study and presentation of scalable solutions for data processing architectures for the *Big Data* challenge. The second focuses on the processing of a dataset comprising measurements that were collected by different sensors, which were installed on a train. The last part focused on is the setup of a server of the open source IoT platform *SiteWhere* [1], the dispatch of data to the server, the storage of the data to a NoSQL database and the processing of these data in a Spark instance.

In Chapter 2, the architecture and the capabilities of *SiteWhere* as a holistic solution for IoT management is presented. Chapter 3 introduces the basic notions of the terms "*Big Data*" and "*Cloud*". It also presents different solutions for the *Big Data* challenge along with the scientific trends on this topic. In Chapter 4, a study of various Clustering algorithms (KMeans, Birch, Mean Shift, DBSCAN), which are used to process the real dataset collected from onboard train sensors, takes place. Chapter 5 introduces the notion of "time-series forecasting" and investigates the behavior of two different types of Neural Networks (MLP, LSTM) with respect to this notion. Chapter 6 presents the work that took place on the *SiteWhere* platform. The chapter begins with the description of the dispatch of data to the server and continues with the visualization, on Grafana, of the train data that were stored in InfluxDB, a database that *SiteWhere* supports. Following this, the retrieval of the data from the database and their processing (through KMeans Clustering and Forecasting with MLP) on a Spark instance takes place and finally a comparison between that process and the one on the local system is presented. Chapter 7 provides a summary and highlights some of the conclusions that were derived and presented in the previous chapters.

It should be noted that the dataset that was processed, is part of a larger dataset, which was available, with measurements from sensors installed on a train. This part of the dataset comprises a single measurement per second, for the duration of a day, for the following parameters:

i. External Temperature (measurement spot 1)
ii. External Temperature (measurement spot 2)
iii. Longitude
iv. Latitude
v. Velocity
vi. CO_2 Level (inside the wagons)
vii. Total Power

Chapter 2

General, Multi-Purpose Solution for Big Data

Plenty of tools, for *Big Data* management, have been reported in the last years, and many more, new ones, are being developed. The first processing frameworks, like Hadoop for example, were designed to work on a physical distributed system. Only big organizations and companies could afford to have their own infrastructures to process *Big Data*, and even they had to overcome the issue of scalability.

More recently a massive turn to the *Cloud* can be observed. This happened for two main reasons. Firstly, the number of technologies that allow processing of data in the *Cloud*, has increased. Secondly, the cost of scaling through the allocation of new virtual machines is much lower than buying and maintaining physical machines. It is becoming obvious that nowadays, the most efficient way to manage *Big Data* is through the utilization of the *Cloud*.

A complete solution for *Big Data* management should rely on scalable platforms that can be deployed at the *Cloud* and can implement the following functions. First, they should have the ability to collect data. This can be achieved via the remote connection of the platform with physical devices that produce the data, for example IoT devices, or through the remote connection with other computers. A second function should be the storage of the data, which have been collected, to a database for future or further analysis. Another capability should be the retrieval of the collected data from the database and their processing. An additional service that should be provided is the processing of the collected data at the same time that they are collected *(real time analysis)*. In the case of IoT, one more, desirable function should be the control of the devices connected to the platform.

There are several platforms that have been developed and meet the requirements discussed above. The one, that has been identified as the most suitable to be used and experiment with, for this master thesis, is *SiteWhere* [1], an open source IoT platform. The following figure presents the general architecture of an earlier version of the *SiteWhere* server.

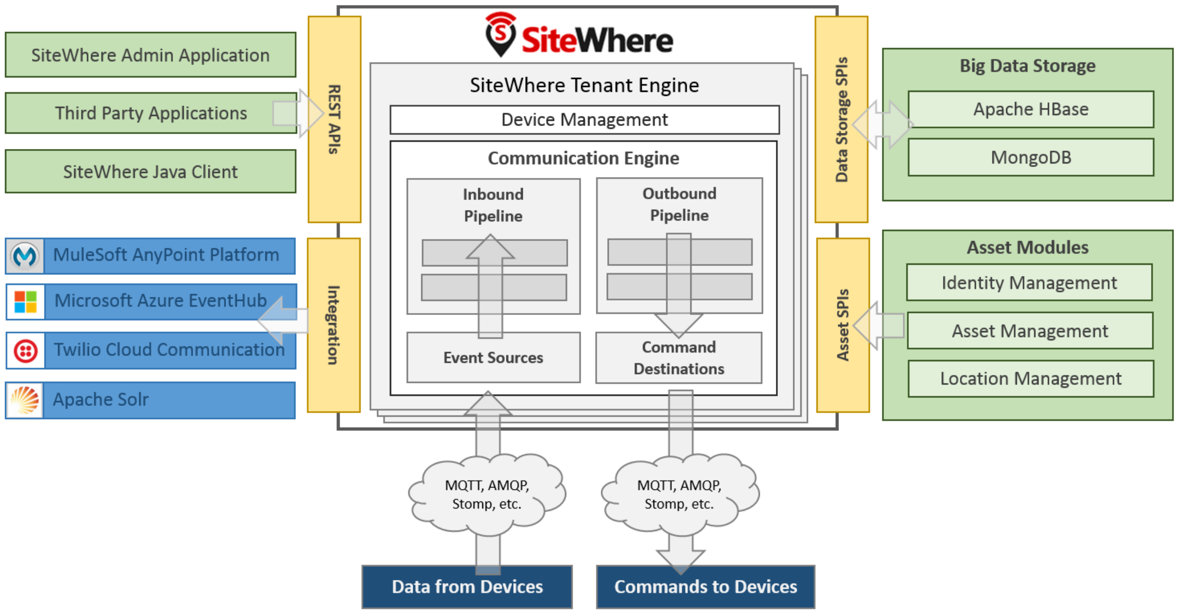


Figure 1: SiteWhere Architecture Diagram.

The *SiteWhere* server is the main application that coordinates every other *SiteWhere* component. It is deployed as a Web Application Archive *(WAR)* file and it uses Apache Tomcat as the core server. *SiteWhere* combines different systems and technologies, that have already been tested and allow the data management, without the need to develop from scratch all functionalities needed for the requires data collection and processing.

In the third paragraph of this chapter, the necessary functions for a platform that could be considered as a solution for *Big Data* management were introduced. The rest of this chapter is dedicated to a brief description of how *SiteWhere* implements those functionalities.

As stated above, the first essential function for systems that process data is the collection of these data. *SiteWhere* can be “fed” with data in many different ways. One option is the usage of REST services. Any authenticated user can perform various operations like create or update data on every entity that he/she has permission and it is registered into the system. An alternative approach is to get data directly from the IoT devices. By default, every connected device can send events (measurements, location, alerts, etc.) to the server through a connection with an MQTT (Message Queue Telemetry Transport) broker, such as HiveMQ or Mosquitto. AMQP (Advanced Message Queuing Protocol) can be used instead of MQTT with some configuration changes. Last, but not least, *SiteWhere* provides the option to accept HTTP requests. This function allows the server to get data from external platforms. The HTTP payload can be parsed by a Groovy script and finally stored in the database like a device event, independently from the format of the external data.

The necessity of storage is achieved by one of the three data storage technologies that are currently supported by *SiteWhere*. The first one is MongoDB, a NoSQL database. The second is the Apache HBase, another distributed NoSQL database. The last one is a time series database relying on a hybrid approach including MongoDB and InfluxDB. This allows the usage of Grafana, an open source platform for monitoring time series analytics.

Since the data have been stored, one could retrieve and store these locally and then process them. *SiteWhere*, allows sending the data into Spark, where the whole data analysis can take place, through the integration of a cluster-computing framework known an Apache Spark. However, the events are streamed via Hazelcast, an open source in-memory datagrid. Every *SiteWhere* Server also acts as a Hazelcast instance, by default. The event stream can be processed initially by Spark Streaming and then used as input to Spark. Spark has available libraries for Machine Learning algorithms (MLlib) and Graph processing (GraphX). In the case where the hybrid approach of data storage is selected, Grafana can also be used for visualization of data and deeper understanding of the events.

Last, but not least, *SiteWhere* provides interaction with the connected devices via the MQTT broker, through REST (REpresentational State Transfer) services. Moreover, the integration with openHAB, an open source software that allows building automation rules, gives *SiteWhere* the opportunity to control directly every openHAB connected device.

Furthermore, *SiteWhere* provides more functionality through the support of Android Development Kit, Arduino Development Kit and Java Agent. The development of android applications that can connect with the server, via an MQTT broker, and send data from the device's sensor is possible. The notification of an important event could also be sent form the server to the android device since their connection is bidirectional. Arduino devices could also easily connect with a *SiteWhere* instance and interact with it. The Java Agent that is provided enables the connection with any device, which can run Java. For example, Raspberry Pi platforms could connect with the server to send data or even to receive commands. Moreover, Node-RED integration, a software for wiring together hardware devices, makes the development of device flows easier, since it eliminates the need for coding.

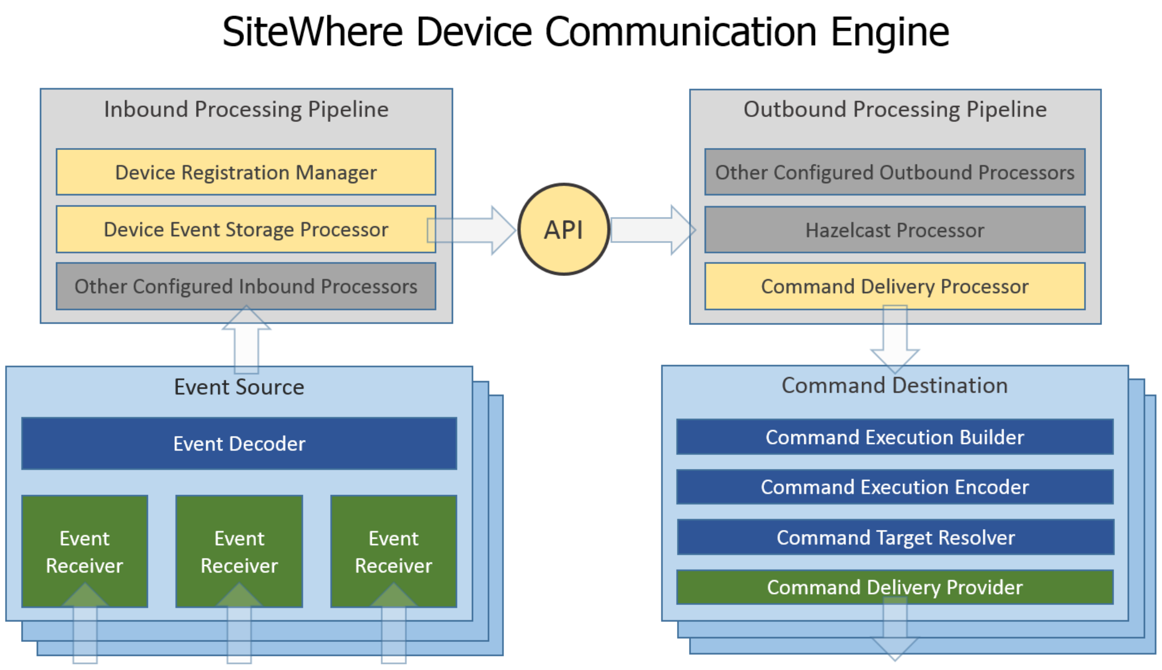

SiteWhere Device Communication Engine
Inbound Processing Pipeline
Device Registration Manager
Device Event Storage Processor
Other Configured Inbound Processors
API
Outbound Processing Pipeline
Other Configured Outbound Processors
Hazelcast Processor
Command Delivery Processor
Event Source
Event Decoder
Event Receiver
Event Receiver
Event Receiver
Command Destination
Command Execution Builder
Command Execution Encoder
Command Target Resolver
Command Delivery Provider

Figure 2: Flowchart of events from SiteWhere-Device bidirectional communication.

Finally, the integration of Mule AnyPoint, an ESB (Enterprise Service Bus) platform, broadens even more the capabilities of the platform as far as IoT is concerned. The communication between different protocols becomes much easier. In addition to this, Mule gives the option of using event processing flows, instead of writing code, through a graphical environment. This leads to an easier way to design complex interactions, triggered by event data.

SiteWhere constitutes an excellent solution for the problem that this master thesis is concerned with. Measurements of different quantities, like temperature, power, CO_2 levels, velocity, coordinates, and many more are taken by devices installed on trains. These measurements can be collected automatically from a *SiteWhere* instance and stored to a database. The real-time visualization of the data that have been collected can be realized with the help of Grafana, in many different ways. Furthermore, part of the data processing can be performed directly by Apache Spark, since the volume of the data is quite large.

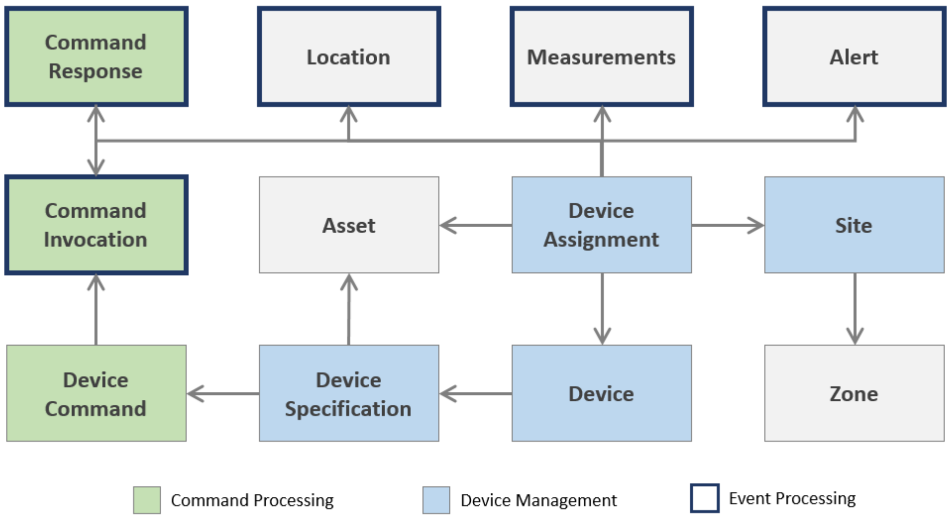


Figure 3: Diagram that represents the core objects, in the SiteWhere model, and their relationships.

In conclusion, *SiteWhere* imprints the relationship between interconnected IoT devices through precisely modeling the concepts related to tracking device data. The usage of this model facilitates the development and deployment of a complete IoT platform, which addresses the various aspects of the *Big Data Challenge* spanning across all functionalities involved from data collection to the data processing. However, the main advantage of *SiteWhere* is still considered that it is open source, and anyone could expand its abilities to the direction needed to solve the task at hand.

In the following chapter, a more thorough investigation about a great number of existing options for *Big Data* management will be presented, along with their pros and cons. In addition, an overview of the algorithms that are used for *Big Data* analysis will be introduced as this is also a part where this master thesis focuses on.

Chapter 3

Theoretical Background – Literature Review

In this chapter, some important concepts linked with the terms *"Cloud Computing"* and *"Big Data"* will be introduced in order to provide a clearer and more complete understanding of this work. Moreover, the trends presented by the scientific community related to *Big Data Challenge* will be presented. The first section focuses on *Cloud Computing*, while the second section analyzes the new infrastructures and algorithms that are used for the management of *Big Data*.

3.1 Cloud Computing

Cloud Computing is a term with over 20 definitions. The National Institute of Standards and Technology (NIST) gave the following.

"Cloud Computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. [2]"

So, *Cloud* grands a variety of computing resources such as servers, storage, or even applications (e-mail, backup, multimedia services, etc.) and allows hosting of these services in an expandable, scalable, flexible, and readily accessible manner. According to NIST, *Cloud* model is composed of five essential characteristics, three service models, and four deployment models [2].

1. **Essential Characteristics:**
   i. *On-demand self-service.*

Automatically and without human interaction, computing capabilities can be provisioned unilaterally by the consumer.

ii. *Broad network access.*

Capabilities are available over the network and accesses through a variety of client platforms (e.g., mobile phones, tablets, laptops, and workstations).

iii. *Resource pooling.*

The provider's computing resources are pooled. According to consumers demand those resources are dynamically allocated. The customer may know the location of provisioned resources at a higher level (e.g. country, state, or datacenter) but not the exact location.

iv. *Rapid elasticity.*

In order to be able to scale rapidly in proportion with demand, capabilities can be elastically provisioned and released.

v. *Measured service.*

Due to the different nature of Cloud services, and the dynamic assignment of resources, a metering capability is needed at a level of abstraction suitable to the type of service. Transparency of the utilized service for both the consumer and provider is guaranteed because resource usage can be monitored, controlled, and reported.

2. Service Models:

i. *Software as a Service (SaaS).*

It allows multiple consumers to use simultaneously the provider's applications, which are accessible from various client devices, running on a cloud infrastructure.

ii. *Platform as a Service (PaaS).*

It provides the hosting of applications, as well as programming languages, libraries, services, and tools for their development.

iii. *Infrastructure as a Service (IaaS).*

It grands access to computing resources, such as processing, storage, networks, etc., where the consumer is able to deploy and run arbitrary software.

3. **Deployment Models:**

   i. *Private cloud.*

   A single organization, consisted of multiple consumers, uses solely the cloud infrastructure. It may exist on or off premises of the cloud provider.

   ii. *Community cloud.*

   A specific community of consumers, usually with shared concerns, uses exclusively the cloud infrastructure. It may exist on or off premises of the cloud provider.

   iii. *Public cloud.*

   The cloud infrastructure is provisioned for open use by the general public. It exists on the premises of the cloud provider.

   iv. *Hybrid cloud.*

   The cloud infrastructure is a synthesis of two or more distinct cloud infrastructures (private, community, or public).

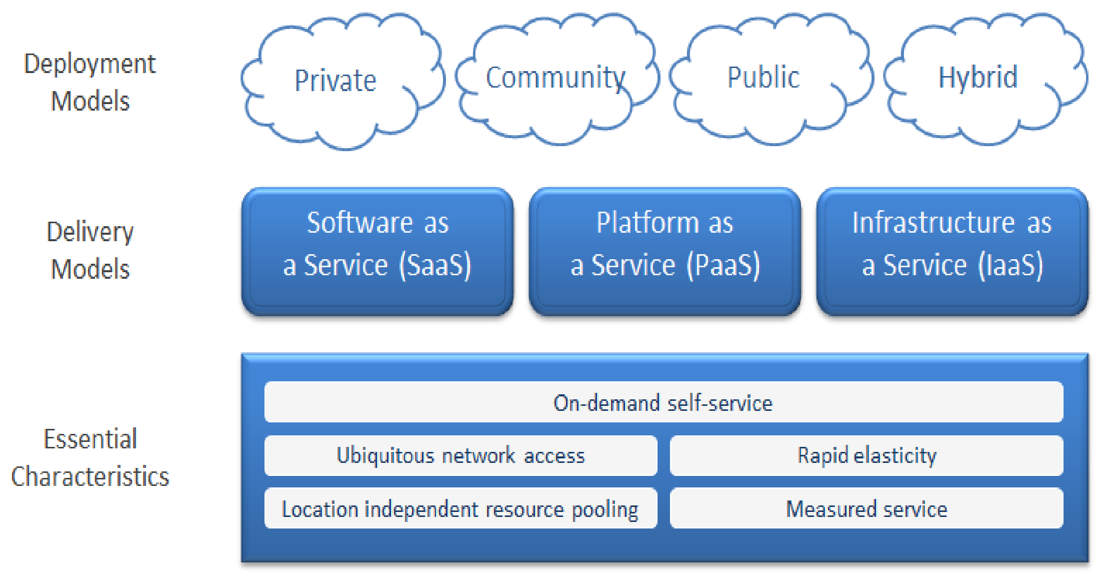

Deployment Models
Private
Community
Public
Hybrid
Delivery Models
Software as a Service (SaaS)
Platform as a Service (PaaS)
Infrastructure as a Service (IaaS)
Essential Characteristics
On-demand self-service
Ubiquitous network access
Rapid elasticity
Location independent resource pooling
Measured service

Figure 4: NIST Working Definition of Cloud Computing [3].

3.2 Big Data

As mentioned in the previous chapters, the traditional analysis approach of *Big Data* has been proved insufficient, and new processing methodologies and architectures are required. It should be mentioned that the term *“Big Data”* although it tries to capture the fact that an enormous amount of data are daily produced, it could be misleading when it comes to the information management challenge, as one could wrongfully focus only on the volume of the data. The updated 3Vs definition came in the foreground from Gartner in 2012.

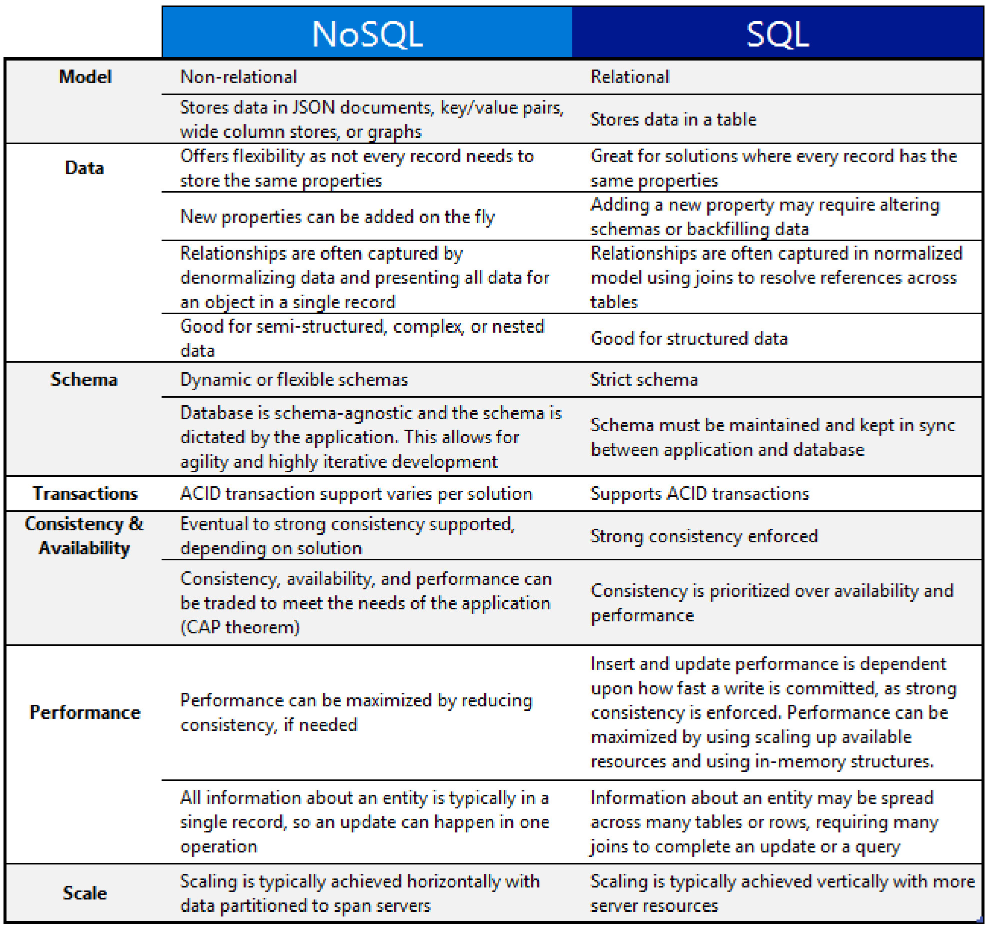
“Big Data is high volume, high velocity, and/or high variety information assets that require new forms of processing to enable enhanced decision making, insight discovery and process optimization. [4]”

The 3Vs have been expanded nowadays with *Variability* and *Veracity* into 5Vs.

The development of a scalable solution to cope with *Big Data* is not trivial. It would be helpful to divide the whole management into 3 different main sections. The first one is about data storage. “Where”, “How”, and “In what form” are the main questions that have to be addressed. This section will be referred to as *Data Model*. The second section is the *Processing Frameworks* in which the processing of the data will take place. The last section refers to the *Algorithms* that will be used to process the data. It should be mentioned that an optimal solution for every problem is not possible.

3.2.1 Data Models

The volume of the data is growing larger year by year. The higher velocity required for processing *Big Data* with the need to be able to store heterogeneous and unstructured data were the causes of the recent increase of NoSQL Databases (currently more than 255 [5]). There are many groups of NoSQL databases. The main differences between SQL and NoSQL Databases are presented at the table 1 and as pointed in [6] the better choice depends on the specific use case under consideration.



	NoSQL	SQL
Model	Non-relational	Relational
	Stores data in JSON documents, key/value pairs, wide column stores, or graphs	Stores data in a table
Data	Offers flexibility as not every record needs to store the same properties	Great for solutions where every record has the same properties
	New properties can be added on the fly	Adding a new property may require altering schemas or backfilling data
	Relationships are often captured by denormalizing data and presenting all data for an object in a single record	Relationships are often captured in normalized model using joins to resolve references across tables
	Good for semi-structured, complex, or nested data	Good for structured data
Schema	Dynamic or flexible schemas	Strict schema
	Database is schema-agnostic and the schema is dictated by the application. This allows for agility and highly iterative development	Schema must be maintained and kept in sync between application and database
Transactions	ACID transaction support varies per solution	Supports ACID transactions
Consistency & Availability	Eventual to strong consistency supported, depending on solution	Strong consistency enforced
	Consistency, availability, and performance can be traded to meet the needs of the application (CAP theorem)	Consistency is prioritized over availability and performance
Performance	Performance can be maximized by reducing consistency, if needed	Insert and update performance is dependent upon how fast a write is committed, as strong consistency is enforced. Performance can be maximized by using scaling up available resources and using in-memory structures.
	All information about an entity is typically in a single record, so an update can happen in one operation	Information about an entity may be spread across many tables or rows, requiring many joins to complete an update or a query
Scale	Scaling is typically achieved horizontally with data partitioned to span servers	Scaling is typically achieved vertically with more server resources

Table 1: Main differences between SQL and NoSQL [6]

The main representatives of *relational database management systems (RDBMSs)* are Oracle Database, MySQL and Microsoft SQL Server. The Kyoto Encyclopedia of Genes and Genomes *(KEGG)* is consisting of 15 main databases, all of them maintained in an Oracle Database [7].

The two most popular *Document Databases* are MongoDB and CouchDB both with many prominent users. They are quite similar but MongoDB has a wider variety of indexing than CouchDB and it can also handle event streaming [8]. MongoDB is used in various kinds of "problems" such as storing recorded video data for live video streaming [9], finding frequent routes of taxi trip events [10], or storing heterogeneous IoT data [11].

Neo4j is the most known *Graph Database* and it is considered as the most efficient solution for storing and querying graph data. The main contribution of a novel framework for scalable Network-based learning methods, presented in [12], is the local implementation of the methods by using Neo4j. Neo4j made possible the whole analysis without the restriction of having the entire network in the main memory. It must be noticed that when the *Louvain method* is implemented on top of a graph, OrientDB seems like the best solution [13].

Riak and Oracle NoSQL belong to *Key-value store databases*. They provide almost the same features, apart from the improved support for streaming in the case of Oracle NoSQL [8]. Oracle NoSQL database can be enhanced with OWL-NOSQL, a semantic reasoner, which will add functionality such as uncover hidden relationships in the data [14].

Another big group of NoSQL databases is the *Wide Column Store*. Two of the main representatives of this group are Cassandra and HBase. Both fault tolerant and with their data distributed across several servers. Cassandra has been tested and used extensively these last years. There are a few noteworthy “extensions” of Cassandra. One of them is the implementation of join operation [15], which is not supported on this kind of NoSQL databases. Another one is the extension of Cassandra query language in such a way that preserves the native syntax while it provides the addition of spatial queries [16]. HBase has also been widely used. Due to the growth of devices that are equipped with GPS (Global Positioning System), an issue for fixed networks have arisen as far as location based services are concerned. This issue is about the update of all devices on real time, everywhere inside a fixed network and the ability to ensure appropriate response time for queries. A novel solution is presented in [17], where HBase has been used.

With the huge growth of IoT and the need for forecasting and time series analysis, InfluxDB and kdb+ have been developed and are characterized as *Time Series / Streaming Databases*, optimized for *Big Data* analysis. Kdb+ is being used by many Wall Street’s trading engines. Historic data can help a lot with real time analysis, on this particular field, but there is an issue due to the lack of compatibility with SQL. An interesting approach to bridge the gap is presented in [18].

Evaluation of even a small number of NoSQL databases, of different categories, through benchmarking is a difficult task and there are not many relevant studies. The only benchmark so far is the *Yahoo Cloud Serving Benchmark (YCSB)*. There is no such thing as an optimal NoSQL database but the “winner” depends mainly on the use case under consideration [19], [20].

3.2.2 Processing Frameworks

The first part of this subsection will serve as a presentation and description of the inner architecture of some popular, state of the art, big data processing frameworks. The second part has been dedicated to the comparison of variety of frameworks.

3.2.2.1 Frameworks Presentation

HADOOP

Hadoop is an open-source, Java-based programming framework that process large data sets, through a distributed computer environment. At the core of Apache Hadoop there is a storage part, known as *Hadoop Distributed File System (HDFS)*, and a processing part, which is a *MapReduce* programming model [21].

Every Hadoop cluster includes a master node and multiple worker nodes. The master node oversees the storage of the data at *HDFS*, through the Name Node, and the execution of parallel computations on these data using *MapReduce*, through the Job Tracker. Worker nodes are the VMs and their job is to store data and execute the computations. Every worker node runs a data node service, under the Name Node, and a task tracker service, under the Job Tracker, which communicates with master nodes to take orders. Figure 5 depicts a Hadoop cluster.

In large Hadoop clusters, in order to prevent potential loss of data and file system corruption a setup of two Name Nodes to manage *HDFS* nodes is introduced. One Name Node server host the file system index, while the other generate snapshots of the Name Node’s memory structures [22].

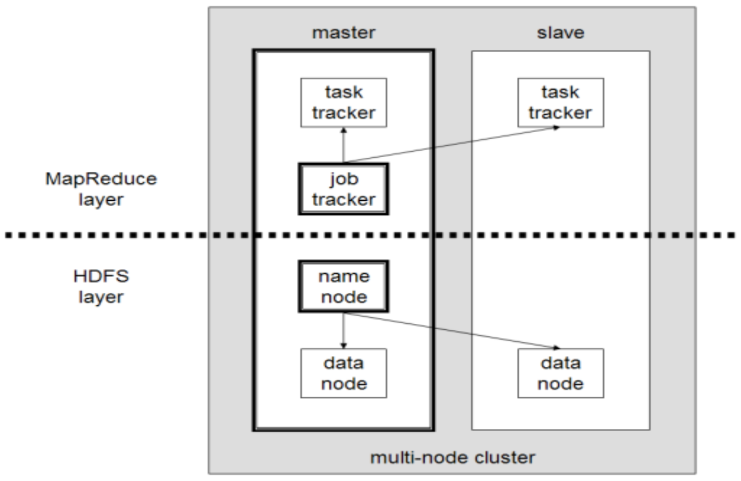


Figure 5: A multi-node Hadoop cluster [22].

SPARK

Apache Spark is an open-source cluster-computing framework. There is an abstraction in the core of Spark, the *Resilient Distributed Dataset (RDD)*. *RDD* is an interface for data transformation and refers to the data stored either in persisted store or in cache or in another *RDD*. Spark supports two kinds of operations, transformations and actions, while the main and only tool for data manipulation is the *RDD* [23].

The system currently supports three cluster managers. A *standalone*, a simple manager cluster included in Spark, *Apache Mesos*, a general cluster manager that can also run Hadoop MapReduce and service applications, and *Hadoop YARN*, the resource manager in Hadoop 2.

A Spark cluster consists of a Driver (main) program and Executors. Driver translates *RDD* into the execution graph, schedules tasks and controls their execution. It is also the place where SparkContext is created. Executors read, write, store data and perform all the data processing. The Spark Architecture is visualized in figure 6.

SparkContext object is the coordinator of all Spark applications, which run on a cluster as independent processes. Once connected to cluster manager, Spark acquires *executors* on nodes in the cluster, sends the application code to them and after that the tasks that they have to run.

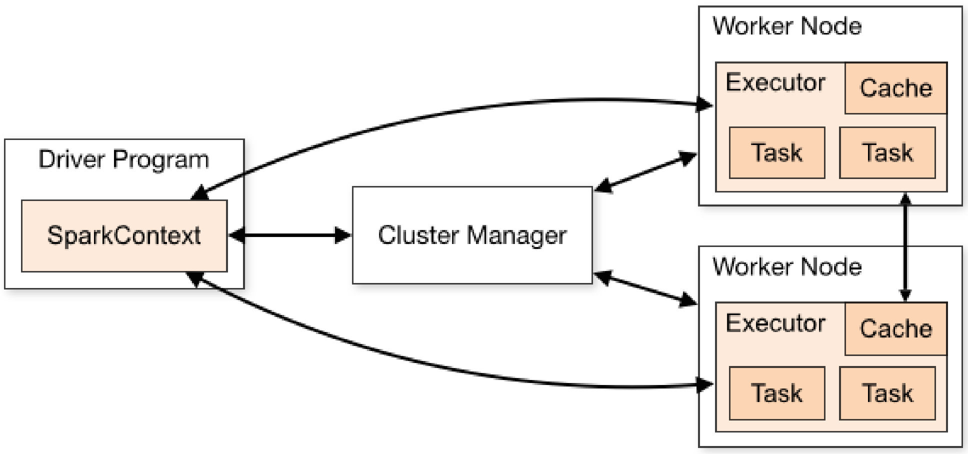


Figure 6: Spark Architecture visualization [24].

Each application gets its own executor processes, which run tasks in multiple threads and are “killed” only after the application has finished its job. This has the advantage of isolating applications from each other, which leads to easier parallelization, but it also means that there is no communication between Spark applications, which constrains us to use an external storage system in case we want to share data across different Spark applications.

The driver program schedules tasks on the cluster and for that reason it must communicate with its executors during the whole time it takes to finish the job, and it would be preferred to run close to the worker nodes, if possible even on the same local area network.

It should also be pointed out that Spark is agnostic to the underlying cluster manager. That means Spark could run even on a cluster manager that also supports other applications with the condition that it could acquire executor processes, which would be able to communicate with each other [24].

STORM

Apache Storm is a distributed real-time computation system for processing large volumes of high-velocity data [25]. The core unit of data for Storm is the tuple, an immutable set of Key/Value pairs, and an unbounded sequence of tuples form a stream. Spout wrap a streaming data source and emits tuples while Bolt receives tuples, performs arbitrary computations, reads/writes data, and optionally emits additional tuples. Storm executes spouts and bolts as individual tasks that run in parallel on multiple machines.

A Storm application is designed as a topology, a network of spouts and bolts, in the shape of a *directed acyclic graph (DAG)* representing the data flow and streaming computation. Graph vertices are spouts or bolts while graph edges are streams [26].

A Storm cluster has two types of nodes *Nimbus (master node)* and *Supervisor (worker node)*. *Nimbus* is responsible for uploading computations for execution, assign tasks to worker nodes, and monitoring failures. Each *Supervisor* follows instructions given by the *Nimbus* and runs one or more worker processes, which are separate JVM processes, on its node. Each worker process runs one or more tasks parallel (spouts/bolts) and executes a subset of topology.

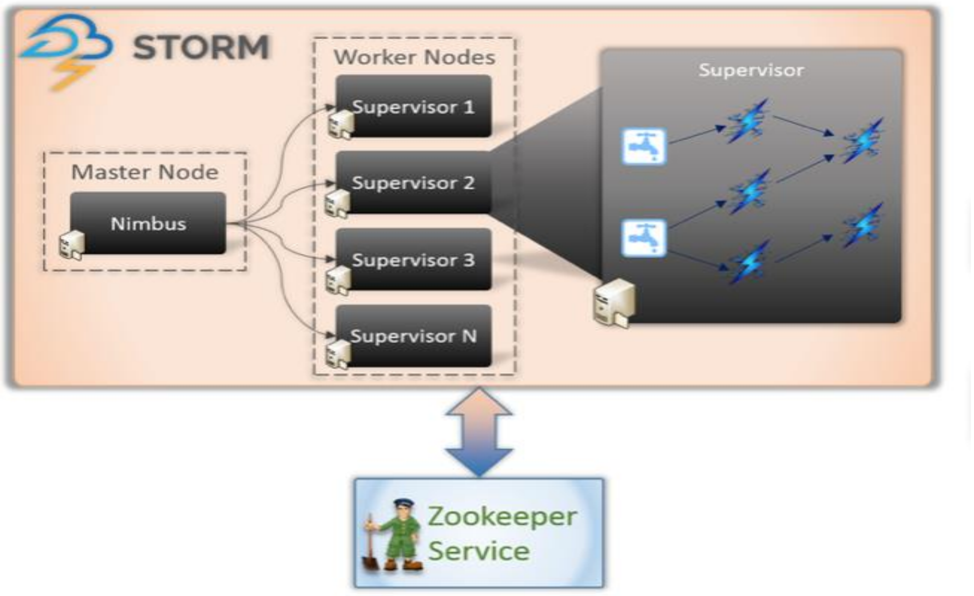


Figure 7: Storm Architecture visualization [27].

Apache Storm is stateless in nature and for that reason Apache ZooKeeper is used to monitor the working node status and manage the cluster state. Coordination between *Nimbus* and the *Supervisors* is achieved through ZooKeeper. This enables Storm to restart a failed *Nimbus*, which will continue its work from where it has stopped, and that is the reason why Storm can process real-time data in the best possible and quickest way [28]. Figure 7 illustrates the Storm Architecture.

ESPER

Esper is an open-source, Java-based software product (recently has been ported to the *C#* programming language) for Complex event processing *(CEP)* and event series analysis. *CEP* is trying to identify relevant relationships, patterns or even data abstractions included in phenomenally unrelated events and fire an immediate response [29].

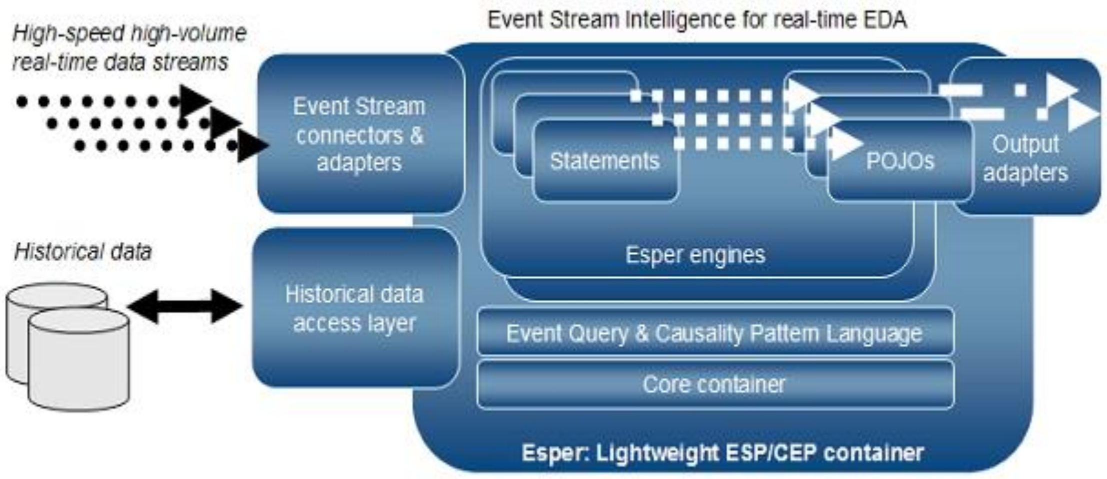

High-speed high-volume real-time data streams
Event Stream Intelligence for real-time EDA
Event Stream connectors & adapters
Statements
POJOs
Output adapters
Esper engines
Historical data
Historical data access layer
Event Query & Causality Pattern Language
Core container
Esper: Lightweight ESP/CEP container

Figure 8: Stream processing with Esper engines [30].

Esper introduces *Event Processing Language (EPL)* by which someone could implement Event-driven programming. *EPL* also provides pattern semantics, which help with the expression of causality relating to time between events. Esper was designed to address the issue of using classical database architectures to query a huge amount of events, with considerable correlation between them, which were impossible to store [30]. The general architecture of Esper is illustrated in figure 8.

It should be noticed that Esper could be combined with Storm and run inside a Storm bolt, instead of running on a single machine, and that gives Esper the benefit of increased scalability.

3.2.2.2 Frameworks Comparison

Batch processing and *Stream processing* are the two main categories in which processing models of *Big Data* are being classified. Different frameworks are used for each processing model. The most widely used frameworks are going to be presented along with their pros and cons.

BATCH PROCESSING

Hadoop is one of the most known programming frameworks for batch processing. It has the ability to handle huge datasets, offers scalability, and due to the high usage of hard disks, and not memory, low cost. However, the usage of hard disks makes Hadoop fairly slow and because HDFS was designed for mostly immutable files it is not very suitable for systems requiring concurrent write operations.

An interesting implementation of Hadoop framework is presented in the following work [31], which gives a significant boost on Hadoop timings. Classic Hadoop implementations perform shuffle only by running the reduce task. iShuffle separates the shuffle from reduce task and by that it lessens the time for a job to complete.

The performance of Hadoop depends a lot on data locality since to transfer data through a network takes a lot of time. The main approaches to address this issue are the Hadoop default scheduler (HDS) and the state-of-the-art balance-reduce scheduler (BAR). A new methodology is proposed, the heuristic bandwidth-aware scheduling (BASS), in order to combine Hadoop with Software-defined networking (SDN) [32].

Apache Spark is another framework for batch processing. Spark executes batch processing jobs about 10 to 100 times faster than Hadoop, offers scalability and also ensures lower latency computations through the high utilization of memory of the spark cluster. It also makes possible to perform on the same cluster except of batch processing different tasks like streaming processing, through the extension Spark Streaming, and machine learning.

A very recent use of Apache Spark, in the back-end, was for filtering large datasets of combustion engines and the scalability evaluation for Spark has shown greater benefits of higher number of workers for big size files [33].

Spark was also used for speeding up deep learning in mobile data analytics. Through MapReduce iterations, a distributed version of deep learning, on many Spark workers, were achieved. Each worker, was learning a part of the model, and all together built the final deep model [34].

<u>STREAM PROCESSING</u> [35], [36], [37]

Apache Storm is a distributed real-time computation system for stream processing. It handles data with very low latency and that is why it is suited for real-time processing. Storm is expanded through an abstraction called *Trident*, which leads to more flexibility but higher latencies.

Scalability has an important role on every big data processing platform. The default implementation for scaling in Storm has some drawbacks. An interesting approach of how to deal with an overloaded topology is presented at [38] where the procedure is, when needed, automatically launched.

Another issue of stream processing platforms is the change of volume or even velocity of the input stream. Because of this variance, it would be desirable from the platform to react on real time and change its size. Storm does not provide this functionality by default. A tool has been developed for monitoring key elements of the structure and based on the information that it gathers decides whether the structure has to grow or shrink [39].

The capability of Storm to act as a real time streaming processor is highlighted in this study [40] where a hybrid real time intrusion detection system is developed with the aid of two neural networks (CC4 instantaneous NN and Multi-Layer Perceptron NN).

Storm and Esper have been used in a study that has been conducted on real data from traffic monitoring of bus traces in the city of Dublin. The approach of this study has shown interesting results that outperforms state-of-the-art techniques with regards to the amount of tuples that could be processed [41].

Apache Samza is another framework for stream processing. Tightly tied to *Kafka*, it is able to provide fault tolerance and state storage. The use of Yarn compels Samza to run on a Hadoop cluster.

Multi-stream joining is becoming an important task for Internet companies. There are two models to join streams, All-In-One and Step-By-Step. Both have their pros and cons. A study for the trade-off of these models has been conducted using Samza [42].

When dealing with large volumes of data the quality has a major role. The challenge of detecting and fixing in real time any issues regarding the consistency of data is addressed in [43]. A different model for Samza is proposed which leads to the optimization of the deployment and as a result to a reduced cost.

Apache Flink is a native streaming system but it can also handle batch processing using the abstraction that everything is a stream. As a result, a batch can be considered as a special case of stream.

The heterogeneity, and the complex relationship between data, give graph analytics an improved role in the *Big Data* era. Flink was used for the implementation of the Extended Property Graph Model [44], which achieved, on social network data, scalability up to 11 billion edges.

Machine learning algorithms have started to be the basis of optimization in industry. A great challenge is the processing of the data to take place as close to the data sources. An OpenTOSCA-based toolchain is presented in [45], which can be deployed very close to the data sources and uses Apache Flink for processing.

Spark Streaming is an extension of Apache Spark capable for stream processing. It comes with all the advantages of Spark but it is only capable for mini batch processing.

A study about Smart Cities uses Apache Spark Streaming instead of hierarchical Smart Grid. It is stated that with Spark it could be achieved a monitor and reaction system for water flows, self-powered through water turbines [46].

	SPARK STREAMING	**STORM**	**SAMZA**	**FLINK**
Main Scope	Streaming	Streaming	Streaming	Streaming
Processing Model	mini batches	event at a time, [mini batches (Trident)]	event at a time	event at a time
Latency	seconds	tens of millis, (seconds with Trident)	hundreds of millis	hundreds of millis
State Management	stateful	stateless [stateful (Trident)]	stateful	stateful
Fault Tolerance	Exactly Once	At Least Once, [Exactly once (Trident)]	At Least Once	Exactly Once

Table 2: Stream Processing Frameworks comparison.

There is a different behaviour of these systems depending a lot at the processing method as can be seen from these experiments [47], [48]. Each platform has a lot of tuning options and with the right tuning for the task at hand it could lead to noteworthy changes of the performance.

3.2.3 Algorithms

Algorithms not only have to solve a problem, but also solving it in a reasonable time, and with reasonable computational resource usage. *Big Data*, primarily due to their large volume, have led to the development of new algorithms. Machine learning algorithms are widely used for *Big Data* analysis.

DATA PRE-PROCESSING/PREPARATION

The first step, and a very important one, in designing a machine learning task is the representation of each pattern in the computer. The encoding of related information that inhabit in the unprocessed (raw) data is achieved through the pre-processing stage [49].

Pre-processing of data is a necessity because real word data are generally incomplete, noisy and inconsistent. There are numerous tasks in data pre-processing splitting into two main categories, data preparation and data reduction techniques [50].

Due to imperfection of data, it is common to require a cleaning task responsible to fill in missing values or even ignore some of the data. Noise is another important issue and is being addressed using data polishing methods or noise filters.

The volume of data can cause problems, as too much data may lead to the “curse of dimensionality” problem. This problem is addressed by various techniques such as feature selection, space transformation, instance reduction/selection/generation or discretization.

Finally yet importantly, in supervised learning, if there is a significant imbalance in the number of representatives for each class, it is going to lead on a classifier biased towards the majority class/classes. For that reason, a resampling of the data is required to bring balance [51].

DATA PROCESSING

i. Correlation

In large collections of data, there is a high probability that values are somehow correlated. For data streams, in particular, we have two different cases of correlation. The first case involves correlations across different streams (cross-correlation) and the second involves correlations across time at the same stream (auto-correlation). The first one provides information on any hidden variable across different data streams while the latter discovers any regular pattern or any periodicity in time series streams [52].

ii. Regression

The regression task is basically a curve fitting problem (supervised learning). From a set of training points we proceed to the estimation of a function f, whose graph should fit the data. With that function available, we can predict the output value of every unknown input. A handful number of problems can be considered as regression problems [49].

iii. Forecasting

Machine learning is all about prediction. When problems involve time, the term used is "Forecasting". Time series forecasting is not time series analysis. In the former, the goal is to predict (forecast) future values of a time series by using information from that series, while in the latter the goal is to understand the underlying causes by developing models that describe accurately this time series. The main issues that should be addressed for a time series forecasting are the volume of available data for training, the time window of a valid prediction, the possibility to update the forecast and the frequency of the forecasts. Supervised learning techniques and Neural Networks can be used for forecasting. A major challenge with forecasting is that it fails to give good predictions over a long time horizons [53], [54].

iv. Classification

In classification, there is a number of known classes and the purpose is to determine in which one of these classes, an unknown pattern should be placed on (supervised learning). The first step to create a *classifier* is to create a training set and then, based on those training data, to design a function *f*, which has to predict the output label, for any given input. This function *f* is the classifier and from the moment it has been trained, is capable to make predictions.

v. Clustering

Clustering, or unsupervised learning, is used when there are no labels for the training data. The task of clustering is to assign points into a number of clusters (groups). Points on the same cluster should be more similar between them, than points belonging to different clusters. For some clustering algorithms the number of clusters should be provided, while for others the number of clusters is considered to be a free parameter recovered from the data. There is a small number of different metrics, which quantifies similarity. Each metric may result in different clustering. Due to the nature of the problem, an optimal clustering is an NP-hard task and for that reason, any clustering algorithm uses approximations, which lead to a suboptimal solution.

Chapter 4

Clustering

The dataset contains information on the route of the Reims tramway. Since many factors change throughout a day, for some specific processing one may first group the routes followed under the same conditions, and then advance with processing.

4.1 Data Preparation for Clustering

As a first step, different plots were produced for the various quantity measurements over time. This step was taken in order to visualize the measurements throughout a day, and to detect the starting and ending time of each tram route.

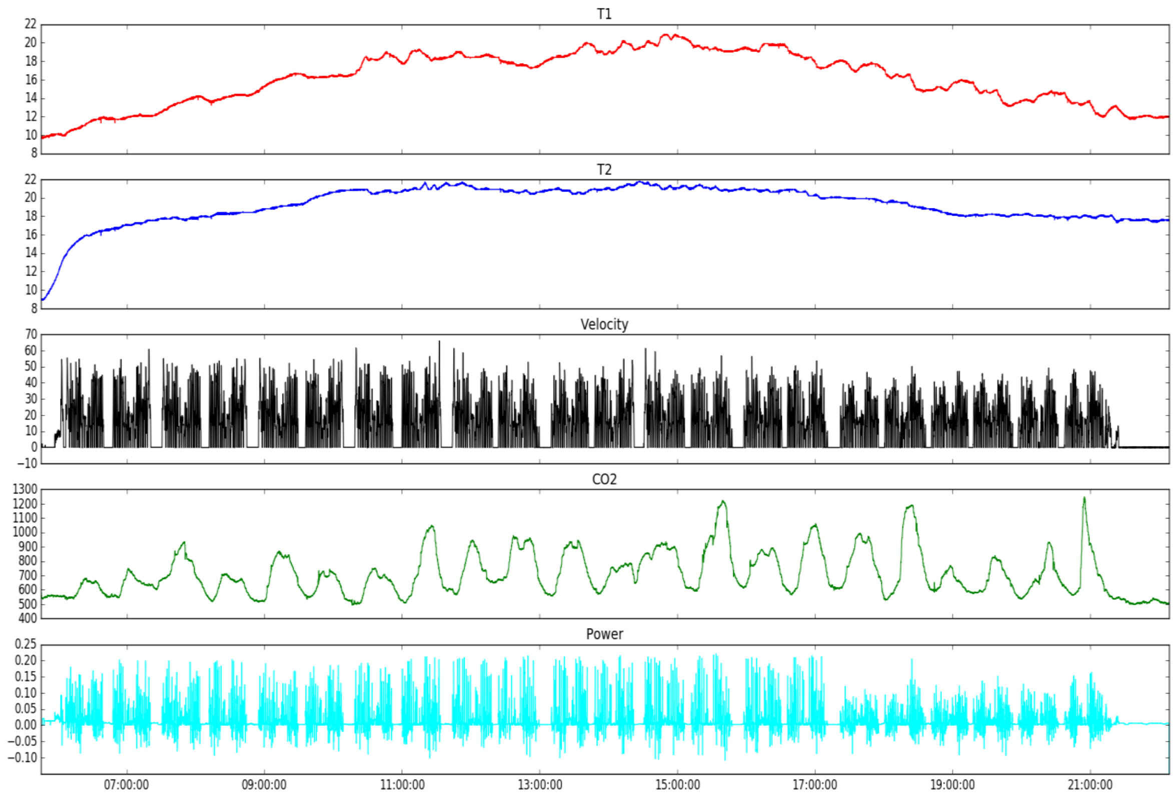


Figure 9: All different measurements across time for one day.

There are two ways to interpret the concept of “route”. Both interpretations were taken into consideration and for that reason three different categories of “routes” were created. One route was formed considering the trip of the tram from the first station to the last station *(Up Route)*. An alternative route was formed from the last station to the first station *(Down Route)*. A third route was formed combining both routes in a round trip from the first station to the last station and back, as a single route *(Full Route)*.

With the aid of the plots *Station-Time and Velocity-Time* the number of daily routes were discovered along with their starting and ending time. The daily number of the routes is 11 and it is the same for all three different “route” categories.

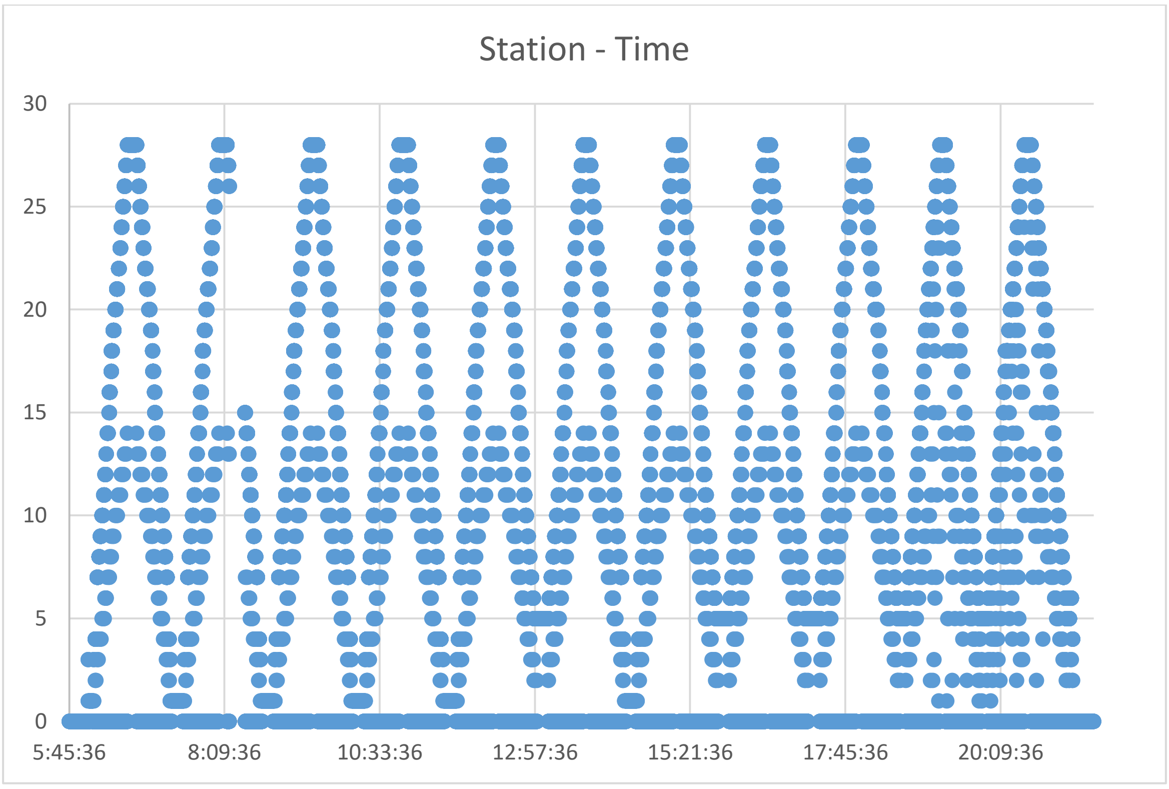
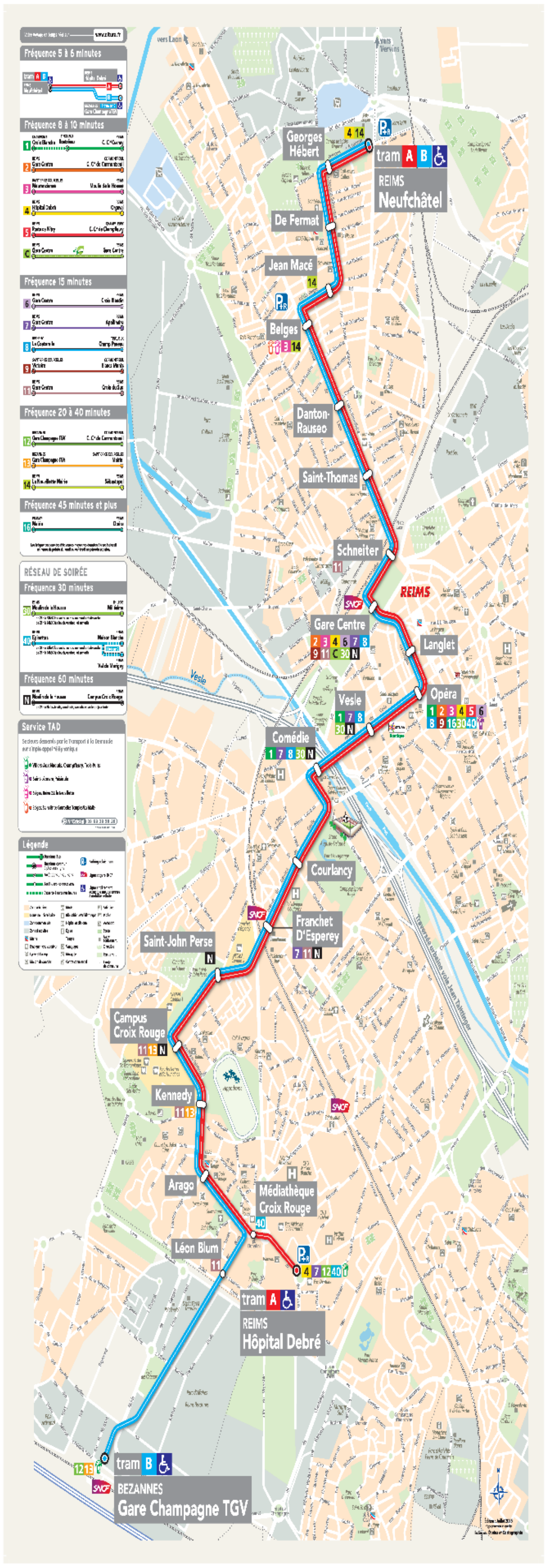


Figure 10: Station-Time plot.

Since the starting and ending time of the routes were identified, and with the usage of latitude and longitude measurements, a plot with the coordinates of the train, along with the duration of the route, were made. A comparison with the official route plan is presented in the following figures.



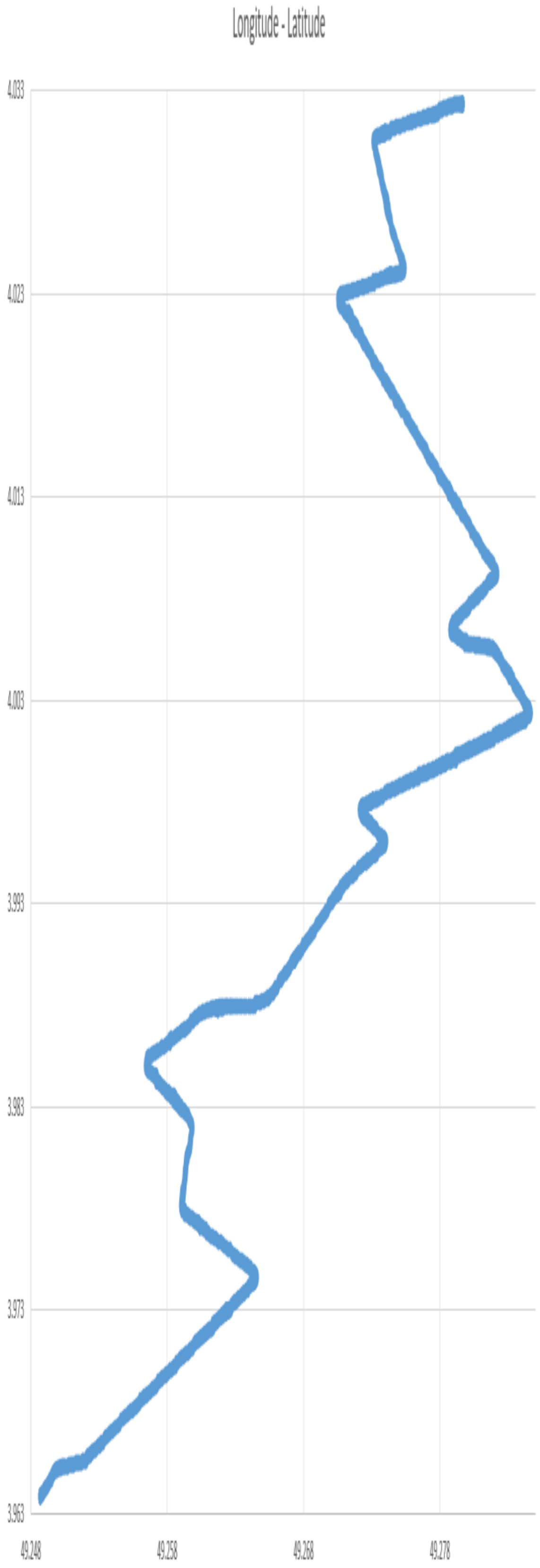


Figure 11: Reims tramway plan [55].

Figure 12: GPS coords plot for a route.

4.2 Clustering Algorithms Results

To be able to group routes that are similar, requires each route to be represented by a feature vector in the same n-dimensional space. For each route the mean values of external Temperatures (T1 and T2), Velocity, and CO_2 Level were computed. Three different feature vectors ([T1, T2], [T1, T2, CO2], [T1, T2, Velocity, CO2]) were used as inputs into four different clustering Algorithms (KMeans, Birch, Mean Shift, DBSCAN).

4.2.1 KMeans

KMeans is a general-purpose clustering algorithm and it is usually considered as the best. KMeans needs to be given the number of clusters. As experimentation, it was decided to run three different scenarios. One for 2 clusters, another one for 3, and lastly, one for 4 clusters.

The results produced from this algorithm are presented in the forthcoming figures. As one of the feature vectors includes four attributes and a 4-dimensional space cannot be visualized, Principal Component Analysis (PCA) was used, for dimensionality reduction, so a visualization of the clustering could be possible.

The results of the clustering may be different before and after the appliance of PCA at the data. This is expected as dimensionality reduction leads to information loss. This loss due to the very low dimension of feature vectors with high probability can be important.

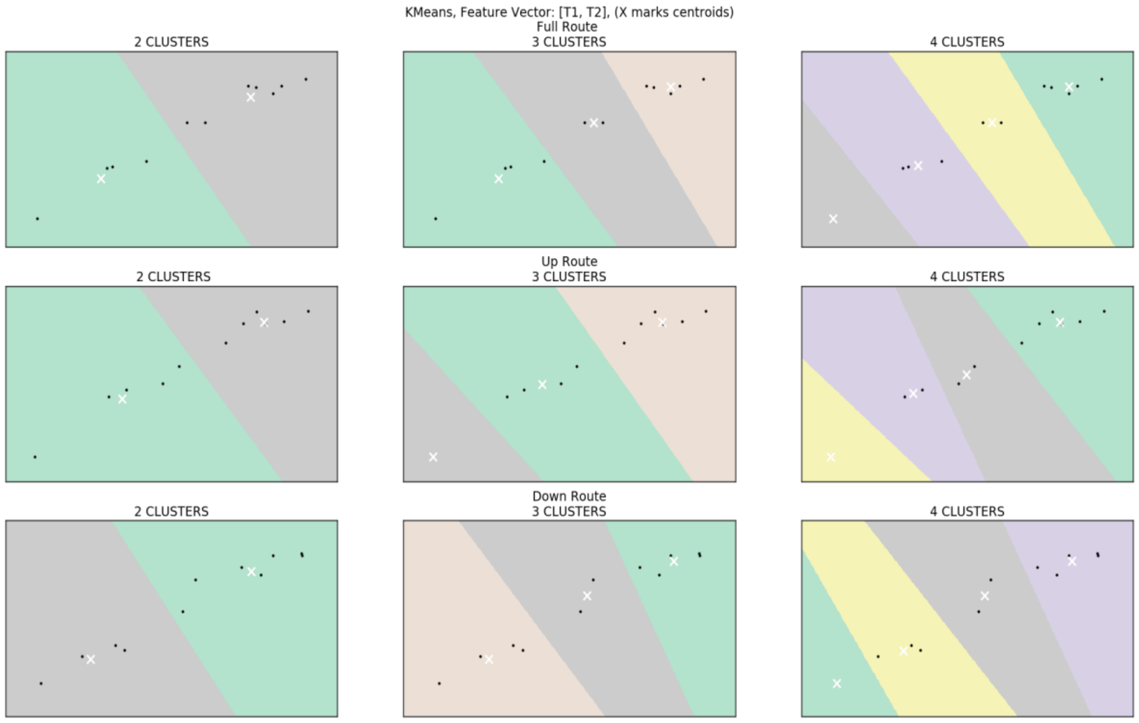


Figure 13: KMeans, Feature Vector: [T1, T2].

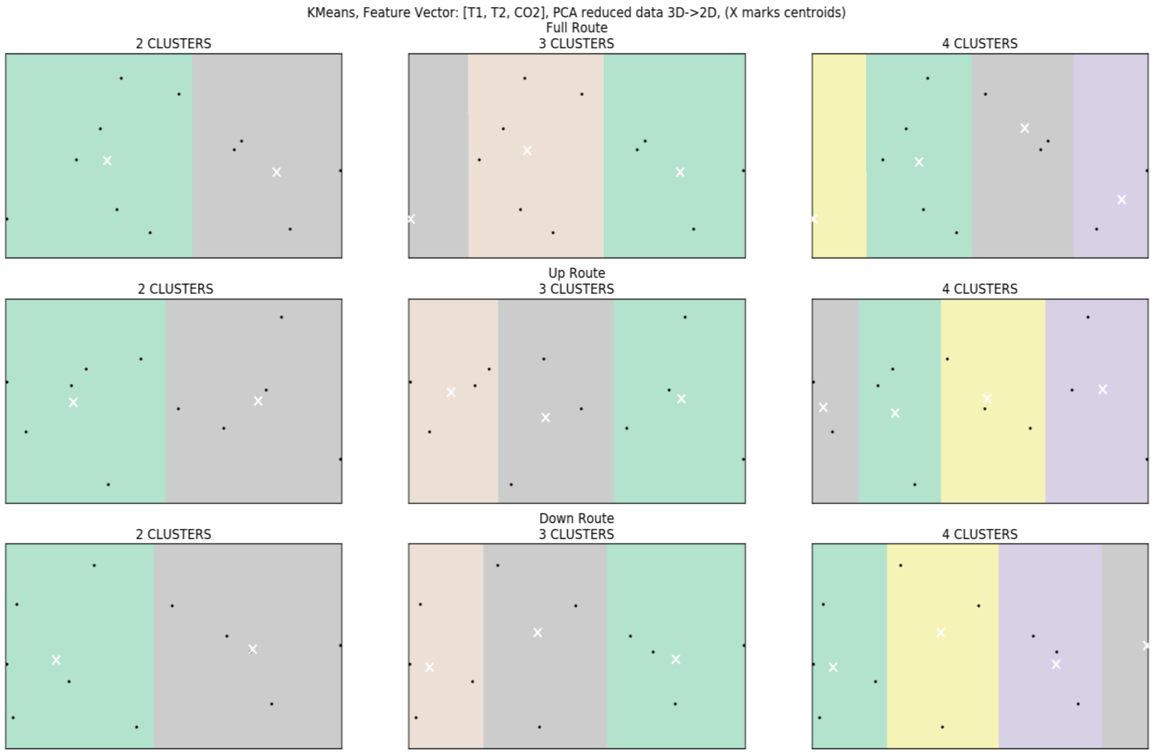


Figure 14: KMeans, Feature Vector: [T1, T2, CO$_2$], PCA reduced data 3D->2D.

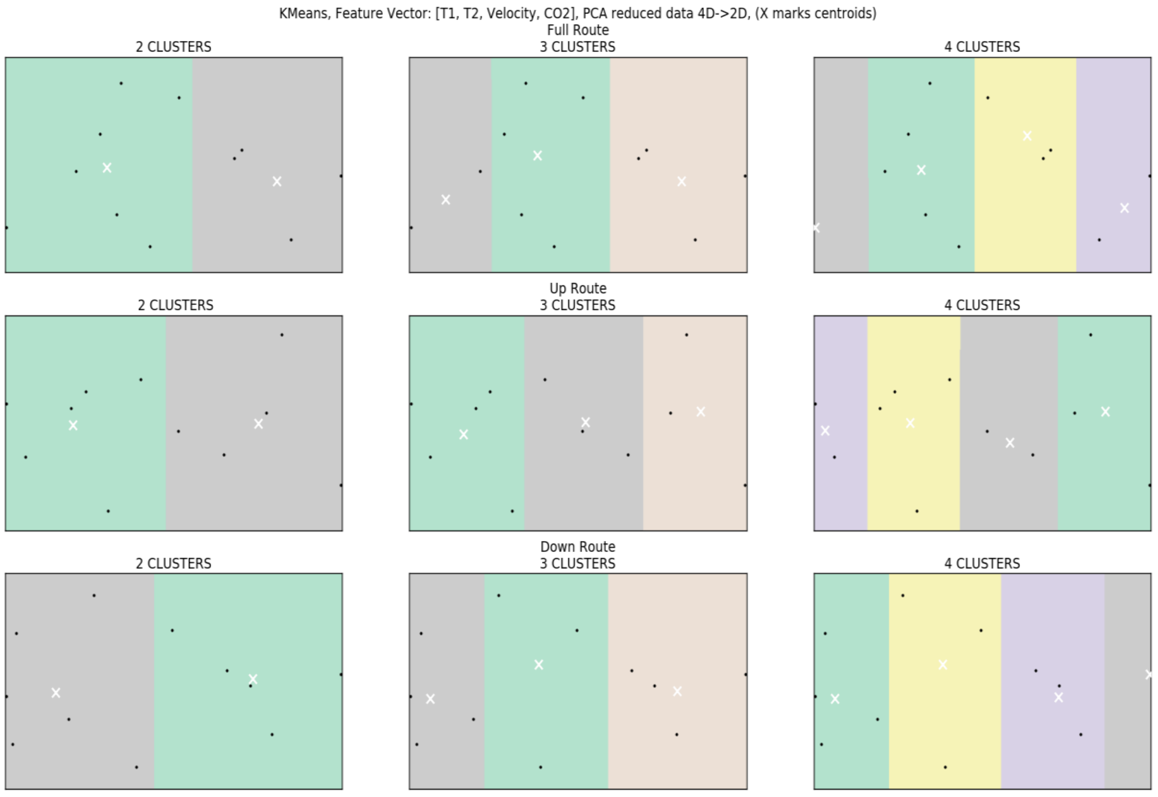


Figure 15: KMeans, Feature Vector: [T1, T2, Velocity, CO_2], PCA reduced data 4D->2D.

4.2.2 Birch

Birch was the second clustering algorithm that was chosen to experiment with. It has been proven to be scalable and although slower than KMeans, it is faster than almost any other clustering algorithm [56].

As the Birch algorithm can be fed with the number of clusters, the same methodology as the one for Kmeans was used. The plots of the clustering results with Birch algorithm, for each different “route” category, are presented at the following figures.

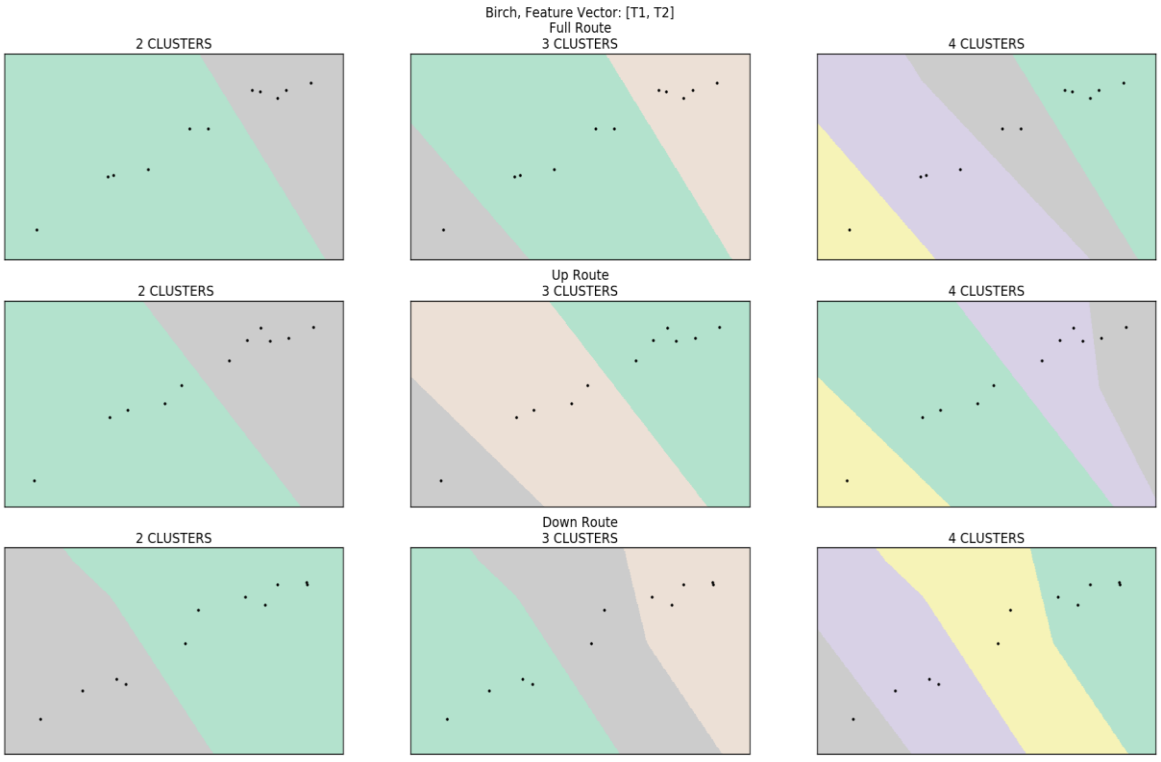


Figure 16: Birch, Feature Vector: [T1, T2].

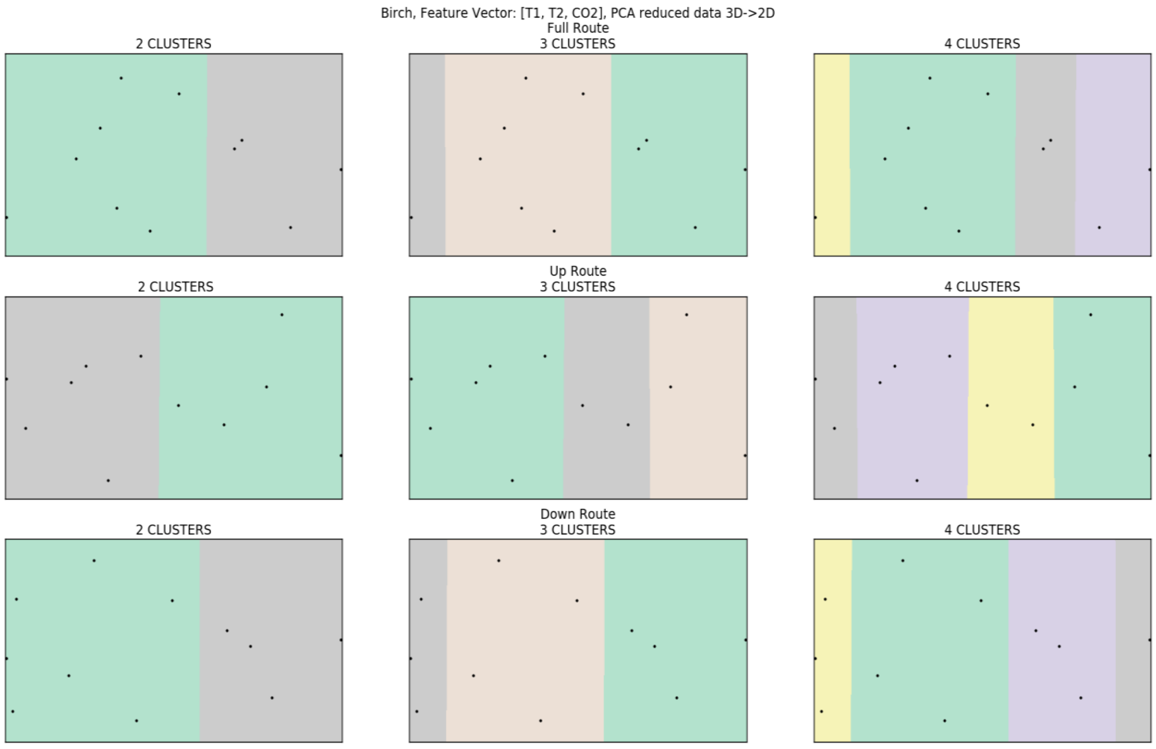


Figure 17: Birch, Feature Vector: [T1, T2, CO2], PCA reduced data 3D->2D.

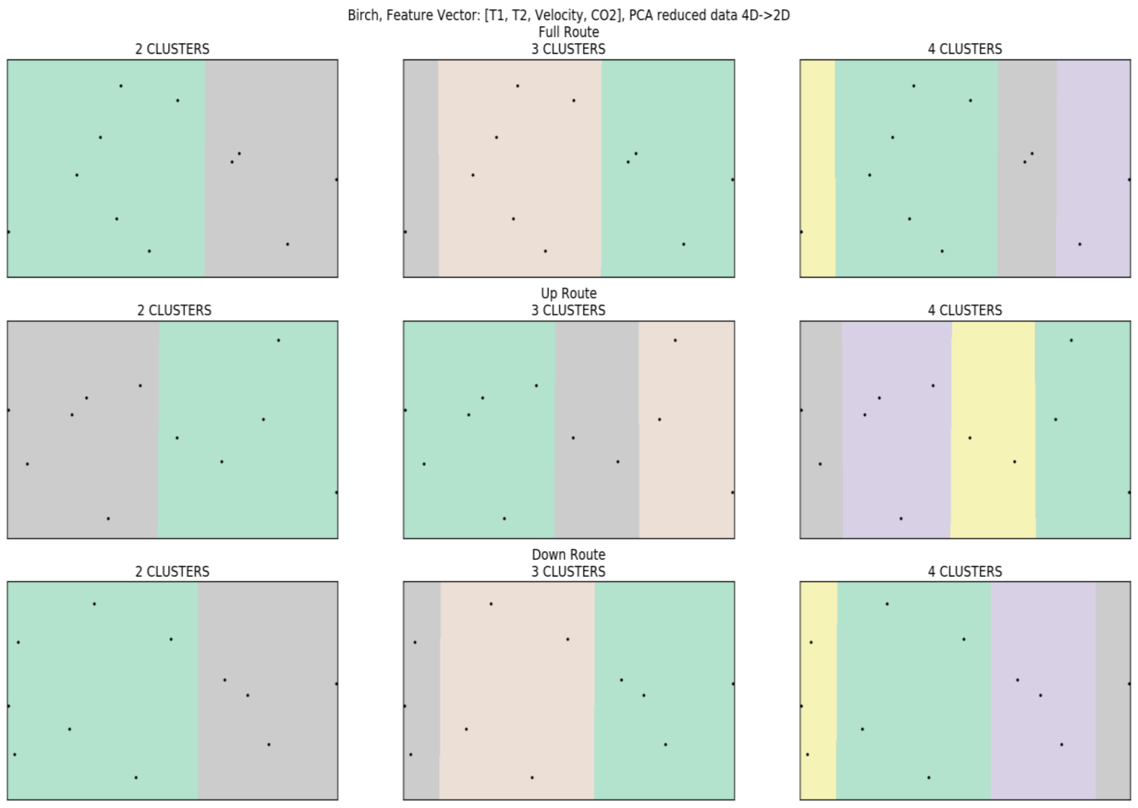


Figure 18: Birch, Feature Vector: [T1, T2, Velocity, CO2], PCA reduced data 4D->2D.

4.2.3 Mean Shift

Mean shift was the next clustering algorithm used. The main difference from KMeans and Birch is that the number of clusters is not provided to the algorithm. On the contrary, the number of clusters has to be discovered, by the algorithm. For that reason, the experiments with Mean Shift algorithm were restricted only to different feature vectors and different route categories. The results are shown in the next figures.

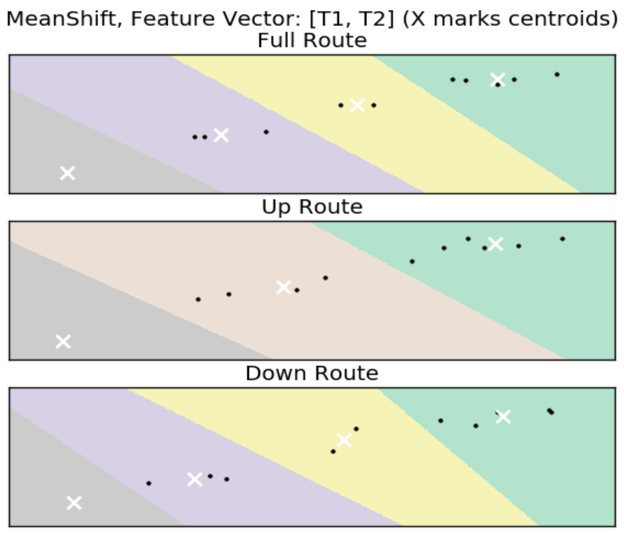


Figure 19: Mean Shift, Feature Vector: [T1, T2].

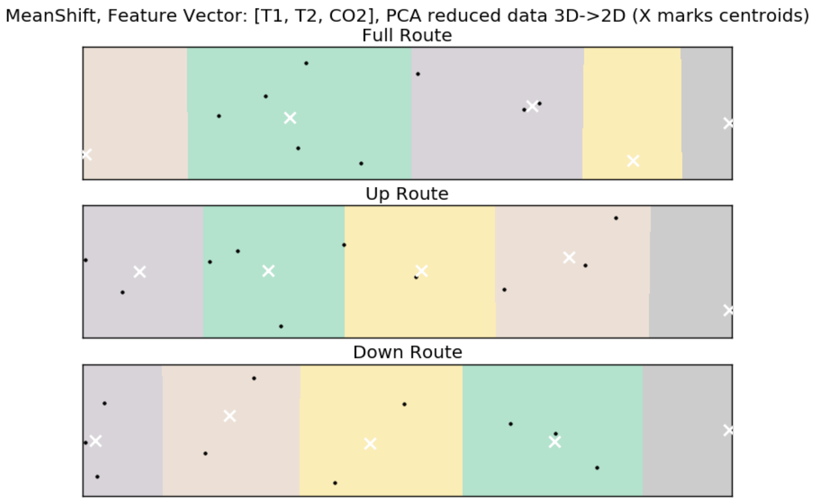


Figure 20: Mean Shift, Feature Vector: [T1, T2, CO2], PCA reduced data 3D->2D.

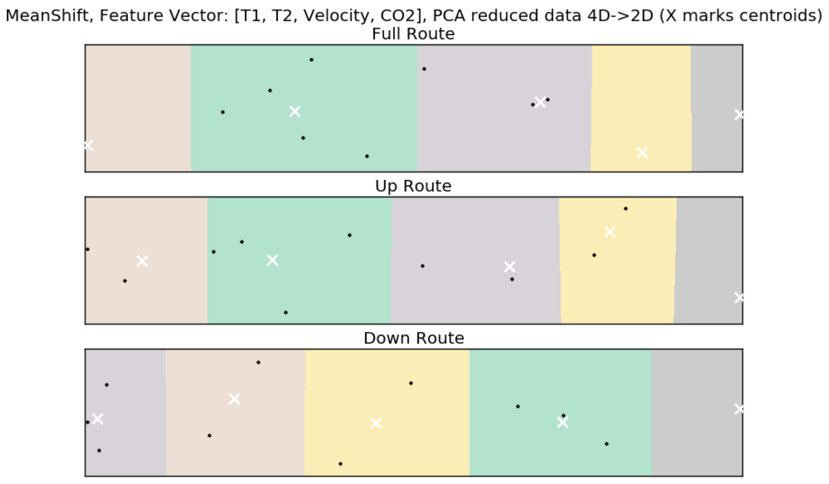


Figure 21: Mean Shift, Feature Vector: [T1, T2, Velocity, CO2], PCA reduced data 4D->2D.

4.2.4 DBSCAN

Due to the small amount of data DBSCAN (Density-Based Spatial Clustering of Applications with Noise) algorithm did not work. The algorithm did not find areas with high density and marked all points (routes) as noise.

4.3 Results Comparison

At the previous section the results of each clustering algorithm, for each feature vector, and for each route category were introduced. Now, a comparison between the algorithms is going to be presented. There are various ways to compare the results of each algorithm. Three of them will be addressed. The first will be to keep the same feature vector and compare the results for each "route" category while the second will be the exact opposite, keep the same "route" category and compare the results for each feature vector. The last one will make a comparison of the algorithms for the same feature vector and route category at the same time.

4.3.1 Same Feature Vector, Different "route" Category

The presentation of results will be categorized by the number of clusters, so a better comparison can be achieved. For every table presented, each cell contains the **id** of a route. That way the comparison about the results of different algorithms is easily perceived.

Feature Vector: [T1, T2] (2D)

As can be seen from table 1, KMeans *(KM)* and Birch *(BI)*, when the number of clusters is *n=2*, gives exactly the same results for *Up* and *Down* routes while there is a small difference when it comes to *Full* route. When *n=3*, KMeans and Birch produce slightly different results for *Full* routes while for *Down* routes they both provide exactly the same. When it comes to Up routes, a complete match between KMeans, Birch and Mean Shift *(MS)* is noticed (table 2). For *n=4*, all three algorithms, Mean Shift *(MS)* too, give the same results for *Full* and *Down* routes. When it comes to *Up* route a significant difference between KMeans and Birch is recorded (table 3).

2 CLUSTERS (FULL)				2 CLUSTERS (UP)				2 CLUSTERS (DOWN)			
CLUSTER A		CLUSTER B		CLUSTER A		CLUSTER B		CLUSTER A		CLUSTER B	
KM	BI	KM	BI	KM	BI	KM	BI	KM	BI	KM	BI
1	1	3		1	1	4	4	1	1	3	3
2	2	4	4	2	2	5	5	2	2	4	4
	3	5	5	3	3	6	6	10	10	5	5
	9	6	6	10	10	7	7	11	11	6	6
10	10	7	7	11	11	8	8			7	7
11	11	8	8			9	9			8	8
		9								9	9

Table 3: n=2, feature vector: [T1, T2], clustering results (KMeans, Birch).

3 CLUSTERS (FULL)						3 CLUSTERS (UP)									3 CLUSTERS (DOWN)					
CLUSTER A		CLUSTER B		CLUSTER C		CLUSTER A			CLUSTER B			CLUSTER C			CLUSTER A		CLUSTER B		CLUSTER C	
KM	BI	KM	BI	KM	BI	KM	BI	MS	KM	BI	MS	KM	BI	MS	KM	BI	KM	BI	KM	BI
1		4	4		1	1	1	1	2	2	2	4	4	4	1	1	4	4	3	3
2	2	5	5	3					3	3	3	5	5	5	2	2	5	5	9	9
	3	6	6	9					10	10	10	6	6	6	10	10	6	6		
	9	7	7	10					11	11	11	7	7	7	11	11	7	7		
	10	8	8									8	8	8			8	8		
11	11											9	9	9						

Table 4: n=3, feature vector: [T1, T2], clustering results (KMeans, Birch, Mean Shift).

4 CLUSTERS (FULL)											
CLUSTER A			CLUSTER B			CLUSTER C			CLUSTER D		
KM	BI	MS	KM	BI	MS	KM	BI	MS	KM	BI	MS
1	1	1	2	2	2	3	3	3	4	4	4
			10	10	10	9	9	9	5	5	5
			11	11	11				6	6	6
									7	7	7
									8	8	8

4 CLUSTERS (UP)											
CLUSTER A			CLUSTER B			CLUSTER C			CLUSTER D		
KM	BI	MS	KM	BI	MS	KM	BI	MS	KM	BI	MS
1	1	-	2	2	-	4	4	-	3		-
				3		5	5			7	
				10		6	6			8	
			11	11		7			10		
						8					
						9	9				

4 CLUSTERS (DOWN)											
CLUSTER A			CLUSTER B			CLUSTER C			CLUSTER D		
KM	BI	MS	KM	BI	MS	KM	BI	MS	KM	BI	MS
1	1	1	2	2	2	3	3	3	4	4	4
			10	10	10	9	9	9	5	5	5
			11	11	11				6	6	6
									7	7	7
									8	8	8

Table 5: n=4, feature vector: [T1, T2], clustering results (KMeans, Birch, Mean Shift).

<u>Feature Vector: [T1, T2, CO2] (3D)</u>

KMeans and Birch for all *n (2, 3, and 4)* gave practically the same results. Mean Shift on the other hand divided the routes to 5 clusters. Even so, the distribution of the routes is very similar with the ones produced from KMeans and Birch. The results for all clustering experiments are presented in the tables below.

2 CLUSTERS (FULL)			
CLUSTER A		CLUSTER B	
KM	BI	KM	BI
1	1	5	5
2	2	7	7
3	3	8	8
4	4	9	9
6	6		
10	10		
11	11		

2 CLUSTERS (UP)			
CLUSTER A		CLUSTER B	
KM	BI	KM	BI
1	1	2	2
3	3	5	5
4	4	6	6
10	10	7	7
11	11	8	8
		9	9

2 CLUSTERS (DOWN)			
CLUSTER A		CLUSTER B	
KM	BI	KM	BI
1	1	4	
2	2	5	5
3	3	7	7
	4	8	8
6	6	9	9
10	10		
11	11		

Table 6: n=2, feature vector: [T1, T2, CO2], clustering results (KMeans, Birch).

3 CLUSTERS (FULL)					
CLUSTER A		CLUSTER B		CLUSTER C	
KM	BI	KM	BI	KM	BI
1	1	2	2	5	5
10		3	3	7	7
		4	4	8	8
		6	6	9	9
			10		
		11	11		

3 CLUSTERS (UP)					
CLUSTER A		CLUSTER B		CLUSTER C	
KM	BI	KM	BI	KM	BI
1	1	2	2	3	3
4	4		5	5	
10	10	6	6	11	11
		7	7		
		8	8		
		9	9		

3 CLUSTERS (DOWN)					
CLUSTER A		CLUSTER B		CLUSTER C	
KM	BI	KM	BI	KM	BI
1	1	5	5	4	4
2	2	7	7	6	6
3	3	8	8		10
10		9	9	11	11

Table 7: n=3, feature vector: [T1, T2, CO2], clustering results (KMeans, Birch).

4-5 CLUSTERS (FULL)														
CLUSTER A			CLUSTER B			CLUSTER C			CLUSTER D			CLUSTER E		
KM	BI	MS	KM	BI	MS	KM	BI	MS	KM	BI	MS	KM	BI	MS
1	1	1	2	2	2	7	7	7	5	5	5	-	-	9
			3	3	3	9	9		6					
			4	4	4				8	8	8			
				6	6									
			10	10	10									
			11	11	11									

4-5 CLUSTERS (UP)														
CLUSTER A			CLUSTER B			CLUSTER C			CLUSTER D			CLUSTER E		
KM	BI	MS	KM	BI	MS	KM	BI	MS	KM	BI	MS	KM	BI	MS
1	1		2	2	2	7	7	7	3	3	3	-	-	1
4	4	4	5	5	5	9	9	9	11	11	11			
10	10	10	6	6	6									
			8	8	8									

4-5 CLUSTERS (DOWN)														
CLUSTER A			CLUSTER B			CLUSTER C			CLUSTER D			CLUSTER E		
KM	BI	MS	KM	BI	MS	KM	BI	MS	KM	BI	MS	KM	BI	MS
1	1	1	4	4	4	7	7	7	5	5	5	-	-	6
2	2	2	6	6					8	8	8			10
3	3	3		10					9	9	9			
10			11	11	11									

Table 8: n=4 or 5, feature vector: [T1, T2, CO2], clustering results (KMeans, Birch, Mean Shift).

Feature Vector: [T1, T2, Velocity, CO2] (4D)

The behavior of all algorithms for the 4D feature vector was similar to the one for the 3D one. Mean Shift, splits also the routes into 5 clusters but the allocation of clusters does not deviate a lot from the results of the other algorithms. The tables below show the results for the 4-dimensional feature vector.

2 CLUSTERS (FULL)			
CLUSTER A		CLUSTER B	
KM	BI	KM	BI
1	1	5	5
2	2	7	7
3	3	8	8
4	4	9	9
6	6		
10	10		
11	11		

2 CLUSTERS (UP)			
CLUSTER A		CLUSTER B	
KM	BI	KM	BI
1	1	2	2
3	3	5	5
4	4	6	6
10	10	7	7
11	11	8	8
		9	9

2 CLUSTERS (DOWN)			
CLUSTER A		CLUSTER B	
KM	BI	KM	BI
1	1	4	
2	2	5	5
3	3	7	7
	4	8	8
6	6	9	9
10	10		
11	11		

Table 9: n=2, feature vector: [T1, T2, Velocity, CO2], clustering results (KMeans, Birch).

3 CLUSTERS (FULL)					
CLUSTER A		CLUSTER B		CLUSTER C	
KM	BI	KM	BI	KM	BI
1	1	2	2	5	5
		3	3	7	7
		4	4	8	8
		6	6	9	9
		10	10		
		11	11		

3 CLUSTERS (UP)					
CLUSTER A		CLUSTER B		CLUSTER C	
KM	BI	KM	BI	KM	BI
1	1		2	2	
4	4		5	3	3
10	10	6	6	5	
11		7	7		11
		8	8		
		9	9		

3 CLUSTERS (DOWN)					
CLUSTER A		CLUSTER B		CLUSTER C	
KM	BI	KM	BI	KM	BI
1	1	5	5	4	4
2	2	7	7	6	6
3	3	8	8		10
10		9	9	11	11

Table 10: n=3, feature vector: [T1, T2, Velocity, CO2], clustering results (KMeans, Birch).

4-5 CLUSTERS (FULL)														
CLUSTER A			CLUSTER B			CLUSTER C			CLUSTER D			CLUSTER E		
KM	BI	MS	KM	BI	MS	KM	BI	MS	KM	BI	MS	KM	BI	MS
1	1	1	2	2	2	7	7	7	5	5	5	-	-	9
			3	3	3	9	9		6		6			
			4	4	4				8	8	8			
				6										
			10	10	10									
			11	11	11									

4-5 CLUSTERS (UP)														
CLUSTER A			CLUSTER B			CLUSTER C			CLUSTER D			CLUSTER E		
KM	BI	MS	KM	BI	MS	KM	BI	MS	KM	BI	MS	KM	BI	MS
1	1		2	2	2	7	7	7	3	3	3	-	-	1
4	4	4	5	5	5	9	9	9	11	11	11			
10	10	10	6	6	6									
			8	8	8									

4-5 CLUSTERS (DOWN)														
CLUSTER A			CLUSTER B			CLUSTER C			CLUSTER D			CLUSTER E		
KM	BI	MS	KM	BI	MS	KM	BI	MS	KM	BI	MS	KM	BI	MS
1	1	1	4	4	4	7	7	7	5	5	5	-	-	6
2	2	2		6					8	8	8			10
3	3	3		10					9	9	9			
6			11	11	11									
10														

Table 11: n=4 or 5, feature vector: [T1, T2, Velocity, CO2], clustering results (KMeans, Birch, Mean Shift).

4.3.2 Same “route” Category, Different Feature Vector

From the tables introduced at subsection 4.3.1 one could derive information on how similar the clustering results are for the same route category and different feature vectors as input.

<u>2D vs 3D</u>

For *n=2*, *Up* route results are identical while *Full* route and *Down* route results show a significant deviation for both KMeans and Birch. For *n=3*, all route categories had very different outcome for KMeans and Birch. For *n=4* each route category gave entirely different results.

As for Mean Shift, a completely alterative outcome were reached for all route categories. For *Up* route 2D input it created 3 clusters, while 3D input 5 clusters. As far as *Full* route and *Down* route are concerned, 2D input led to 4 clusters, while 3D input 5 clusters.

<u>2D vs 4D</u>

For *n=2* all route categories have identical results for both KMeans and Birch, while for *n=3* and *n=4* are completely different. The same observation as before, 2D vs 3D, applies for the Mean Shift algorithm.

<u>3D vs 4D</u>

For *n=2* all route categories gave identical results for both KMeans and Birch. For *n=3*, *Full* and *Up* routes gave almost the same results and *Down* routes had identical outcome. When *n=4* Birch gave the exact same results while KMeans produced identical results for *Full* and *Up* routes, and practically the same results for *Down* route. Mean Shift on the other hand, had a minor deviation for *Full* route while reached the exact same outcome for *Up* and *Down* routes.

4.3.3 Same Feature Vector, Same “route” Category

The clear majority of the algorithms’ results are identical, or very similar. Although for a few combinations, *[n=3, 2D, Full]*, *[n=3, 4D, Full]*, *[n=3, 4D, Up]*, and *[n=4, 2D, Up]* there are considerable divergences between the outcome of each algorithm. For the sake of visualization, all plots for 3D feature vector are presented below.

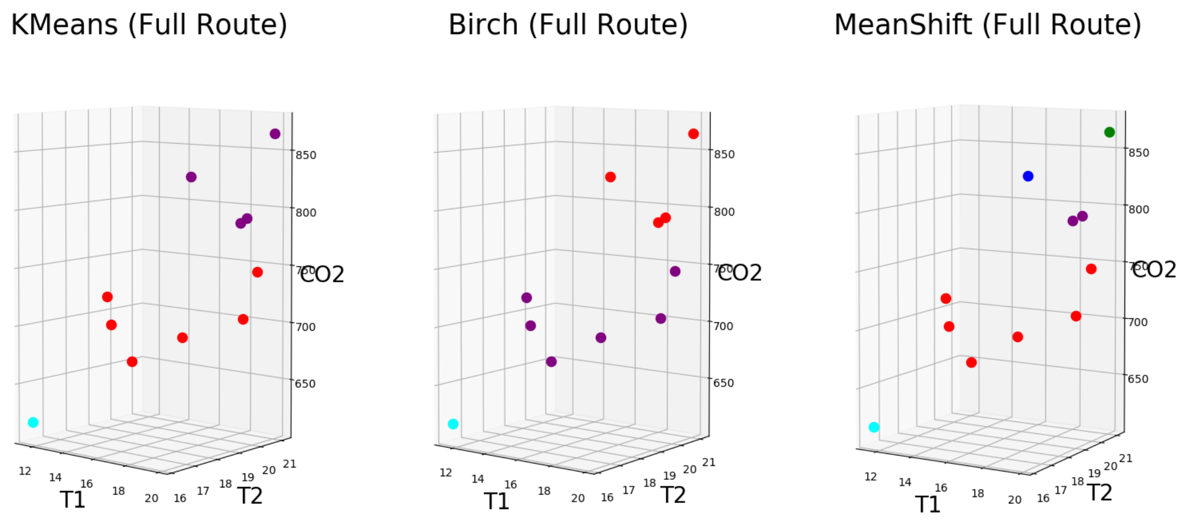


Figure 22: 3D feature vector, Full route, all algorithms.

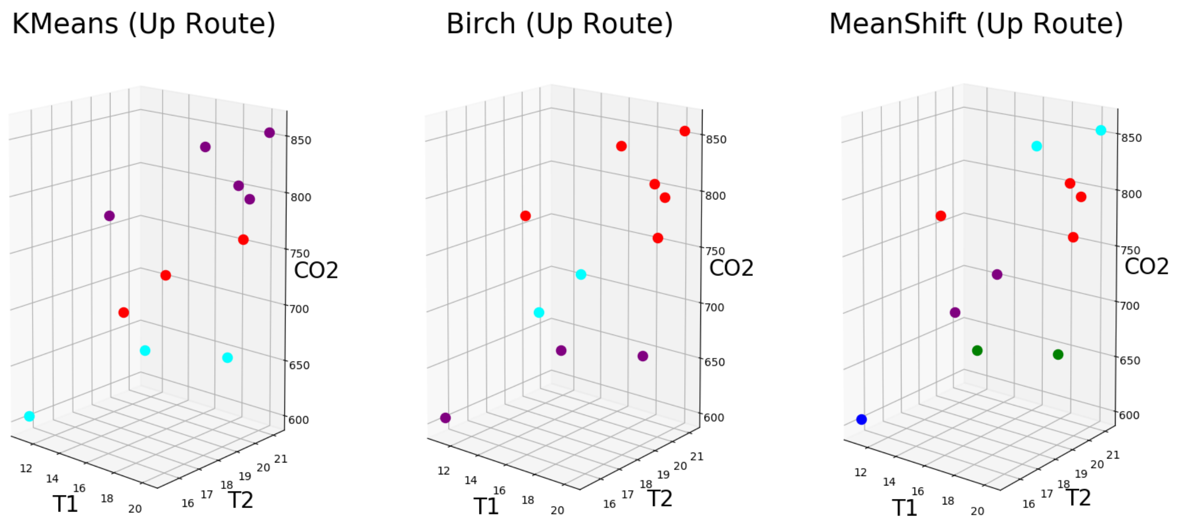


Figure 23: 3D feature vector, Up route, all algorithms.

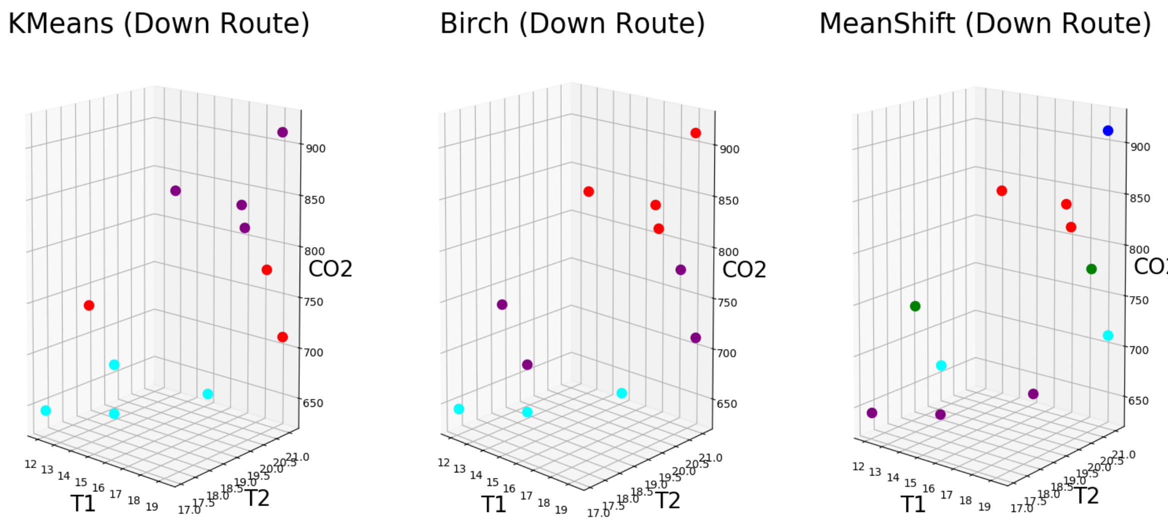


Figure 24: 3D feature vector, Down route, all algorithms.

4.4 Discussion

As the number of the routes that have to be grouped is very small, no generic conclusion, regarding the algorithm suitability for the task under consideration can be reached. The results of all algorithms are identical in 40% of the cases, almost the same in 51% of the cases, and display important deviation in 9% of the cases. However, these observations may be different if a larger number of the routes was available.

Another issue is the execution time of the algorithm. Even if an algorithm gives better results than an alternative one, if the time required to run is too long, it may be better to use the faster one with slighter worse results. Due to the small number of points that the algorithms had to group, this important aspect was not possible to be evaluated. For this feature to be evaluated a larger dataset has to be analyzed, in order to gather information that will assist in assessing the result’s quality versus execution time.

Chapter 5

Forecasting

A time series is a collection of data points collected, commonly, at time intervals of equal length. Time series modeling, and by extension time series forecasting, is more complex than the analysis of any sequence not depending on time. There are two main reasons for that. First of all, in the regression model, a main hypothesis is that the observations are independent. This hypothesis does not hold true on time series since the data are correlated. Secondly, time series usually tend to exhibit seasonality trends in addition to any other trend.

Traditional forecasting method uses statistical models like ARIMA (AutoRegressive Integrated Moving Average) for example. The approach of this master thesis will be the design and the implementation of two different Neural Networks for the forecasting. The first one will be an MLP (MultiLayer Perceptron), a Feedforward NN, and the second an LSTM (Long Short-Term Memory), a Recurrent NN. The main difference between an FNN and an RNN is that in the FNN, the output of a layer does not affect that same layer, in other words, there is no feedback. On the other hand, in the RNN, loops are showing up, thus allowing information to persist. That is the reason why it is told that an RNN has "memory".

5.1 Neural Networks Introduction

The human brain could be characterized as a highly complex information-processing system. The way that brain computes, presents many differences from the way a typical computer does. Brain, also has the ability to develop, and this development depends highly on the "experiences" each person has [57].

Artificial Neural Networks, usually referred as "Neural Networks", were inspired by human brain. They are composed from simple processing units, called artificial neurons, that are massively interconnected. Their goal is to model the way in which the brain would carry out a specific task. Through a learning process, a Neural Network can obtain knowledge. That gained knowledge is stored to the interneuron connection strengths, which usually called synaptic weights. The change of these weights is the conventional way for Neural Network design. Nevertheless, the creation of a Neural Network that can alter even its own topology is possible and that constitutes a second design method.

Neural Networks are not a new idea. Actually, they were introduced decades ago, but the first results were not encouraging. The reason though, they have become a trend in the last decade, is because they started to produce fine results. The main cause of this change was *Big Data*. Neural Networks "learn" through training. Training involves the feeding of a Neural Network with data, in order to acquire knowledge and achieve good performance. In the *Big Data era*, an enormous volume of data exists. That made possible the development of sufficiently trained Neural Networks, which can perform well.

5.2 Multilayer Perceptron Model

The standard linear perceptron could not distinguish data that are not linearly separable. For that reason, a modification of the linear perceptron were developed, the Multilayer Perceptron (MLP). MLPs are widely used and have three main features.

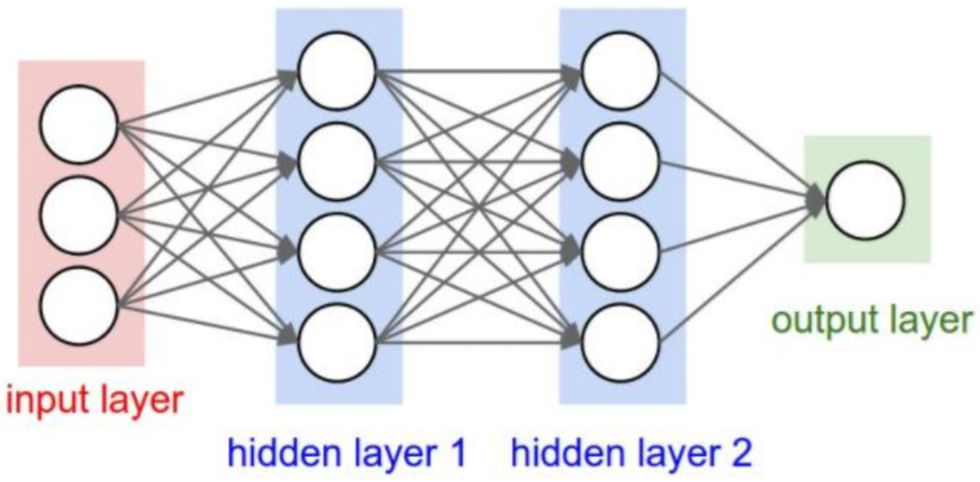


Figure 25: A regular MLP Neural Network with 2 hidden layers [58].

i. The network contains one or more layers between the input and the output nodes. Those layers are called hidden layers.

ii. The interconnectivity of network's neurons is high and its extension is decided by the network's weights.

iii. The activation function of each network's neuron has to be nonlinear and differentiable.

A sufficient number of experiments took place with the MLP model for forecasting the levels of CO_2 inside the cabins of the train. There are many different parameters, which can be changed, to tune a neural network. In each series of the experiments, one parameter was changed, while the others stayed constant, in order to achieve an evaluation of the performance of the neural network.

5.2.1 MLP Experiment Results

The epochs of the training was the first parameter that was tested. That choice was made for two reasons. The first one was, so that we could determine the range of the number of training epochs that would not lead to underfitting or overfitting. The second was to establish a baseline of the performance of the Neural Network.

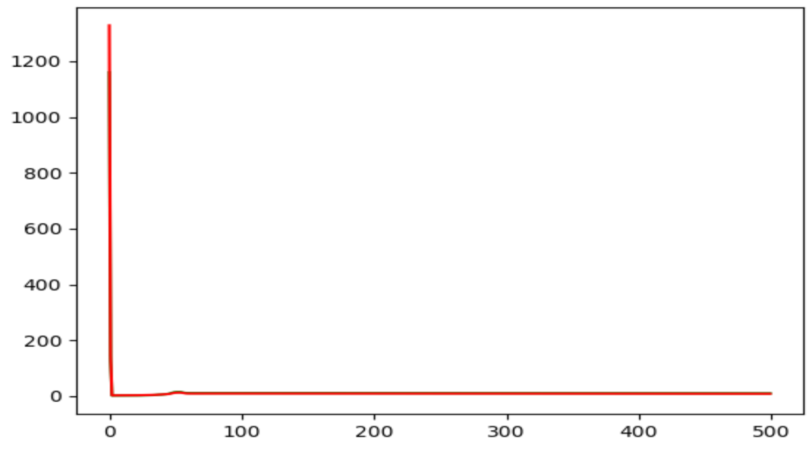


Figure 26: Plot of RMSE against Epochs of training for MLP Network (1 hidden layer, 7 neurons, window = 1, batch size = 100). The green line corresponds to the RMSE of the prediction on the training set while the red line corresponds to the RMSE of the prediction on the test set.

After studying the above curve, a decision to set the number of epochs to 100 for the rest of the experiments were made. The batch size was the second parameter that was studied. Usually, the whole train set is not propagated to the Neural Network to train with all at once, but instead, it is distributed in batches, which are used to train the Neural Network. The number of the data, that every batch is composed from, is called batch size. The results are presented at the following table. A batch size of 50 was selected for the rest of the experiments.

NEURONS	WINDOW	BATCH	EPOCHS	RMSE
7	1	**1**	100	**2.65**
7	1	**2**	100	**2.497**
7	1	**4**	100	**2.49**
7	1	**6**	100	**2.501**
7	1	**8**	100	**2.496**
7	1	**10**	100	**2.531**
7	1	**25**	100	**2.495**
7	1	**50**	100	**2.493**
7	1	**75**	100	**2.5**
7	1	**100**	100	**2.54**
7	1	**150**	100	**2.5**
7	1	**200**	100	**2.5**
7	1	**300**	100	**2.5**
7	1	**400**	100	**2.498**
7	1	**500**	100	**2.5**

Table 12: MLP results for different Batch Sizes (1 hidden layer).

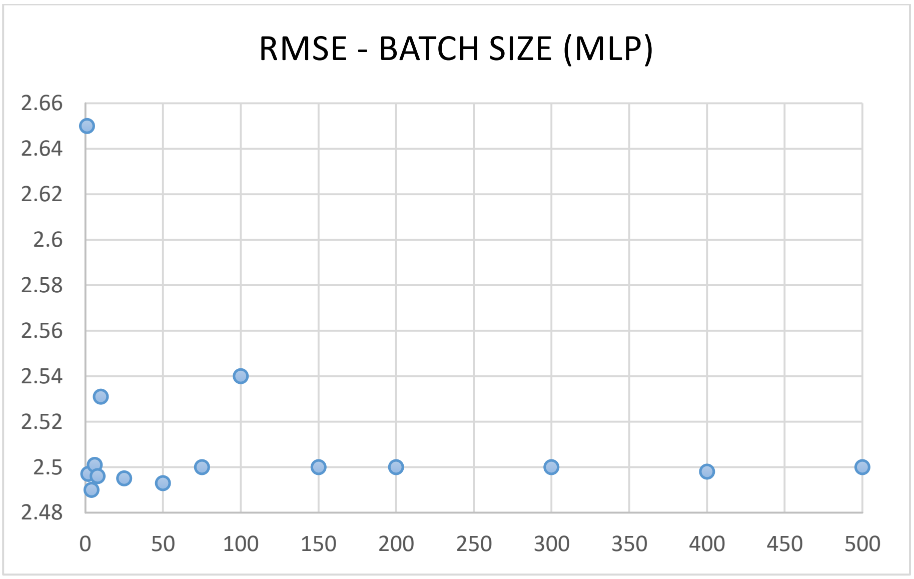


Figure 27: Plot of RMSE against Batch Size for MPL Network (1 hidden layer).

The third parameter that was investigated was the window. Since the problem is the forecasting of a time series, that term corresponds to the number of the previous data, from a time instant, that are available to the Neural Network in order to forecast the value of that particular time instant. It is observed that the better score is achieved when the MLP takes into consideration only the previous value. That means that the value of CO_2 Level depends mostly from the exact previous value and that the older measurements act as noise.

NEURONS	WINDOW	BATCH	EPOCHS	RMSE
7	**1**	50	100	**2.493**
7	**2**	50	100	**2.995**
7	**3**	50	100	**2.755**
7	**4**	50	100	**3.03**
7	**5**	50	100	**2.82**
7	**6**	50	100	**2.684**
7	**7**	50	100	**2.715**
7	**8**	50	100	**3.808**
7	**9**	50	100	**3.808**
7	**10**	50	100	**4.703**

Table 13: MLP results for different Window Sizes (1 hidden layer).

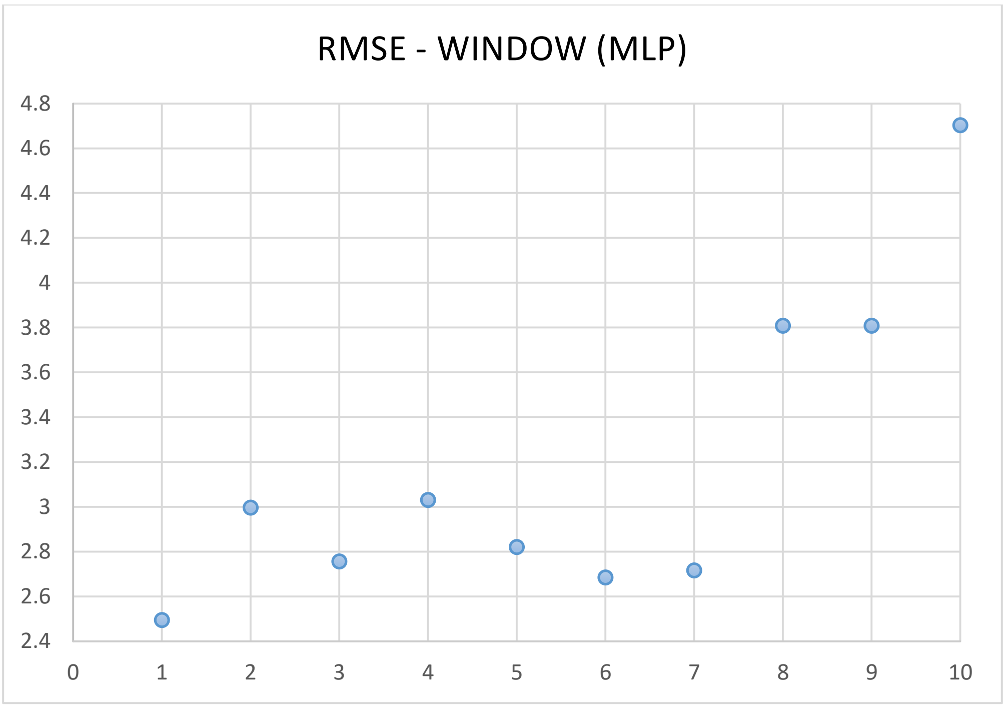


Figure 28: Plot of RMSE against Window Size for MLP Network (1 hidden layer).

The number of neurons, on an MLP with a single hidden layer, was the fourth parameter that was examined. The increase of the number of neurons, showed that there is no rule as “the more the better”, and that the best suited number of neurons depends on the nature of the quantity, which the Neural Network is trying to forecast.

NEURONS	WINDOW	BATCH	EPOCHS	RMSE
1	1	50	100	**734.733**
2	1	50	100	**2.489**
4	1	50	100	**2.489**
6	1	50	100	**2.494**
7	1	50	100	**2.493**
8	1	50	100	**2.497**
10	1	50	100	**2.509**
15	1	50	100	**2.509**
20	1	50	100	**2.504**
30	1	50	100	**2.489**
40	1	50	100	**2.49**
50	1	50	100	**2.538**
75	1	50	100	**2.518**
100	1	50	100	**2.595**

Table 14: MLP results for different number of Neurons (1 hidden layer).

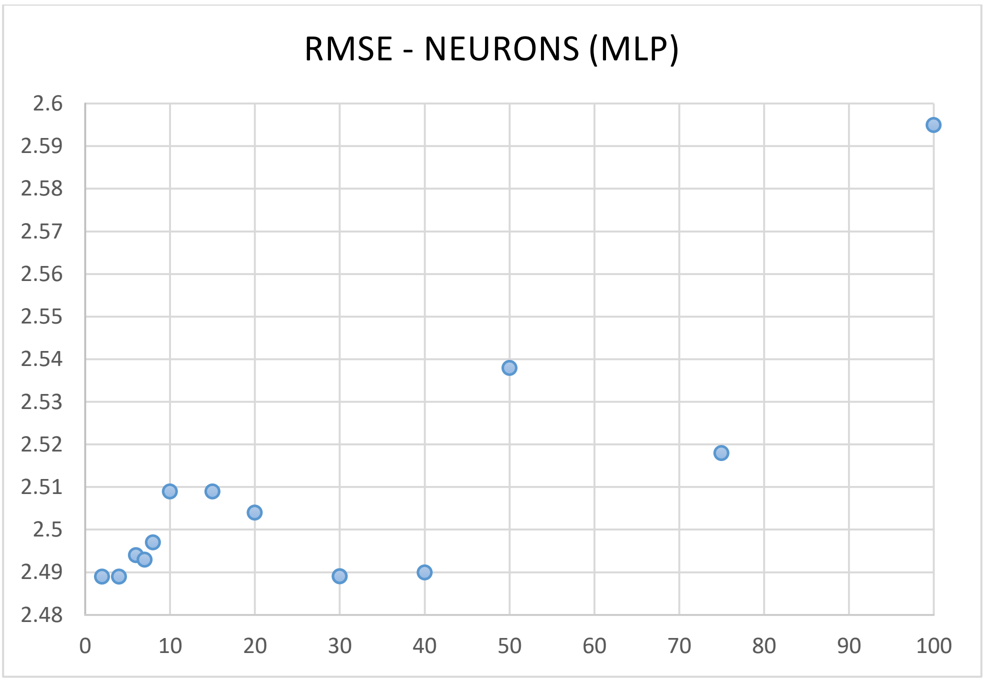


Figure 29: Plot of RMSE against Number of Neurons for MLP Network (1 hidden layer).

The last parameter that was chosen to investigate was the number of the layers of the MLP, and by extension the number of neurons for each layer. The multiple number of hidden layers did not increased the performance of the MLP. On the contrary, when the number of the hidden layers were more than 3, the performance presented a decrease, which for some setups was significant. Some representative cases are presented in the following table.

NEURONS	WINDOW	BATCH	EPOCHS	RMSE
4-4	1	50	100	**2.498**
4-6	1	50	100	**2.526**
4-7	1	50	100	**2.499**
4-8	1	50	100	**2.541**
7-7	1	50	100	**2.616**
7-14	1	50	100	**2.602**
30-30	1	50	100	**2.521**
30-60	1	50	100	**2.93**
4-4-4	1	50	100	**3.061**
7-14-28	1	50	100	**4.39**
7-7-14-14	1	50	100	**2.821**
7-14-14-28	1	50	100	**2.932**
7-14-28-56	1	50	100	**2.759**
10 Hidden Layers	1	50	100	**7.013**

Table 15: MLP results for different number of Hidden Layers and Neurons.

The performance of MLP, as far as the forecasting of CO_2 Level is concerned, is extremely good. The lowest value of RMSE (Root Mean Square Error) that MLP achieved was 2.489. Taking into account that the mean value of CO_2 Level for the entire day is 710, the error of forecasting values is about 0.35%. It should be mentioned that, despite our initial expectations, the deep Neural Network architectures achieved either almost equal score, as the 1 hidden layer MLP, or were outperformed.

Due to the good performance of the MLP to CO_2 Level forecasting it was decided to test that architecture for velocity forecasting too. The parameters were chosen intuitively. After 20 different runs, the lowest RMSE value the MLP achieved was 1.3. Since the mean velocity is 13.55 approximately, the lowest RMSE value is translated to 9.59% error on an estimation, which is very high.

5.3 Long Short-Term Memory Network

As mentioned before the main advantage of an RNN is the “memory”. If the distance of an important information from the place that it is required is relative small, an RNN, or even an MLP with a sufficient “window” can learn to use that information. In the case where the useful information is many steps behind an MLP with an enormous “window” would fail. Although, theoretically, an RNN would be capable of using this information, in practice, it does not [59].

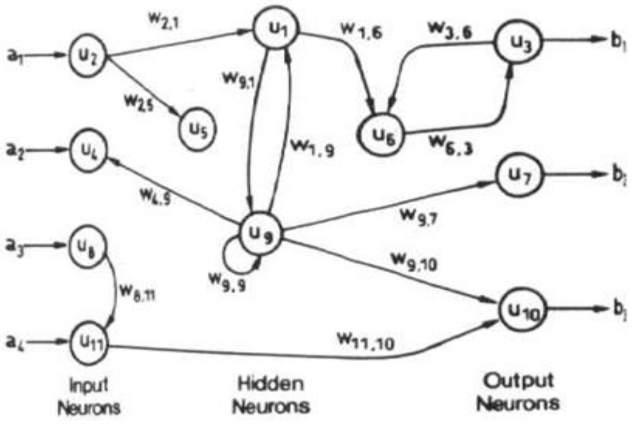


Figure 30: An example of a random Recurrent Neural Network [60].

Long short-term memory (LSTM) is a Recurrent Neural Network architecture, which can learn long-term dependencies. The design of the LSTM allows it to overcome the issue of the long-term dependencies that has emerged from the standard RNN. Each RNN could be considered as a chain of successively units of the same neural network. The standard RNN’s module has a very simple structure while the LSTM’s module consists of four neural network layers.

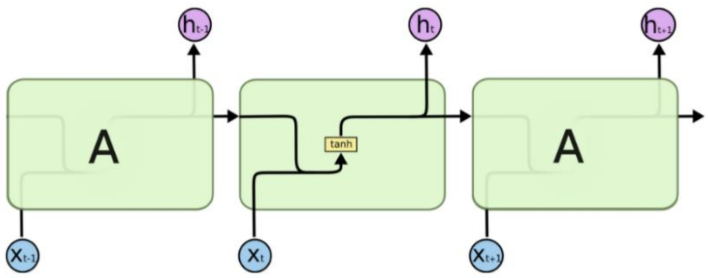


Figure 31: The repeating module in a standard RNN [61].

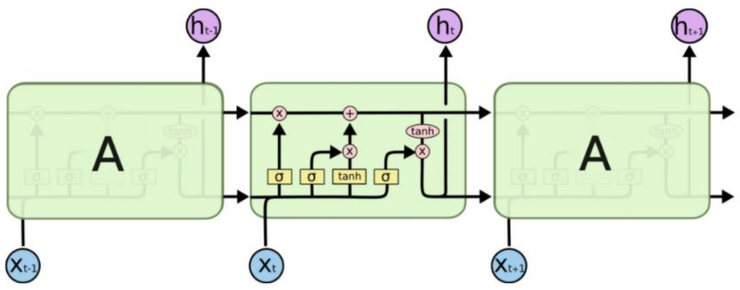


Figure 32: The repeating module in an LSTM [61].

5.3.1 LSTM Experiment Results

In this series of experiments, the results of an LSTM according to the number of neurons were studied. Moreover, an investigation on the impact of the input, stationary or non-stationary, and on the capability of the LSTM to save its state, to the NN's performance was conducted.

An LSTM network prediction is depending on the activation of the memory cell of the previous time step. The stateless LSTM initiates the cell states and hidden states, usually to zero, after each batch, while the stateful LSTM transfers the last cell and hidden states of a training batch to the next one.

Like the MLP experiments, the initial parameter for LSTM testing was the epochs of the training, for the same two reasons. To avoid the underfitting or the overfitting of the network, and to establish a baseline for the performance of the LSTM.

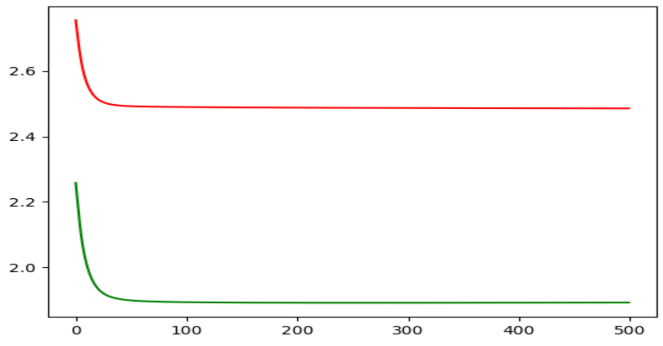


Figure 33: Plot of RMSE against Epochs of training for LSTM Network (1 hidden layer, 2 neurons, window = 1, batch size = 100). The green line corresponds to the RMSE of the prediction on the training set while the red line corresponds to the RMSE of the prediction on the test set.

The number of training epochs was decided to be 100 for each of the following experiments. Since the LSTM uses as activation function the $tanh$, the scaling of the training set between the values [-1, 1] was deemed necessary. All the experiments are going to investigate the performance of an LSTM network, with 1 hidden layer, against the number of neurons. The first network to be studied is a stateless LSTM, which receives as input the scaled dataset.

NEURONS	WINDOW	BATCH	EPOCHS	RMSE
2	1	100	100	**4.413**
4	1	100	100	**5.164**
6	1	100	100	**5.852**
8	1	100	100	**3.38**
10	1	100	100	**2.794**
15	1	100	100	**3.036**
20	1	100	100	**3.087**
30	1	100	100	**2.813**
40	1	100	100	**2.571**
50	1	100	100	**2.735**
75	1	100	100	**2.587**
100	1	100	100	**2.57**

Table 16: LSTM results for different number of Neurons (1 hidden layer, stateless, non-stationary input).

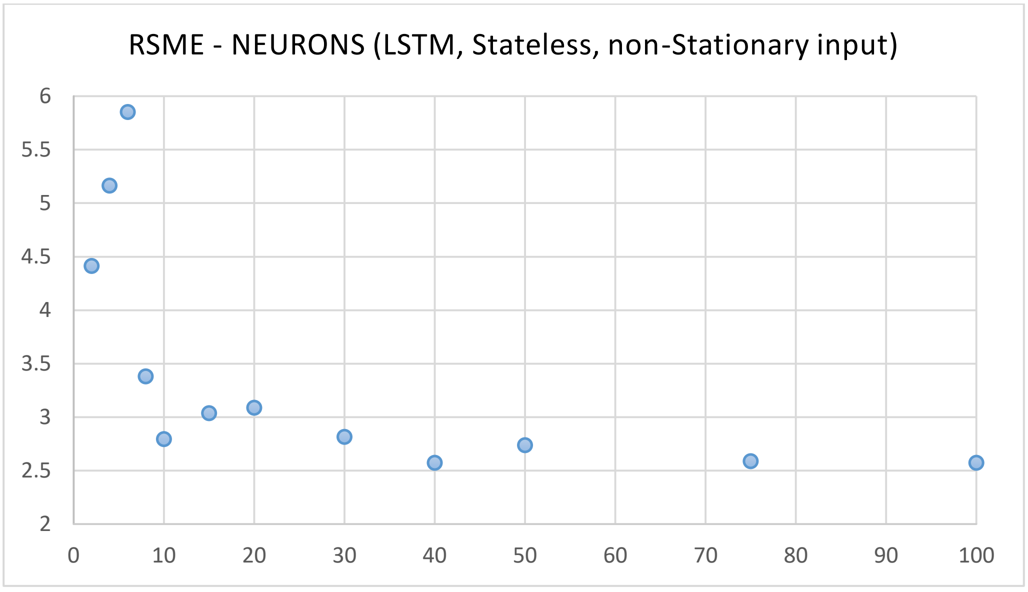


Figure 34: Plot of RMSE against Number of Neurons (LSTM, 1 hidden layer, stateless, non-stationary input).

The second one is a stateless LSTM also, but this time the input is a stationary dataset. In order to make the dataset stationary the differencing method was used at the original dataset. Then the scaling was applied to the differenced dataset.

NEURONS	WINDOW	BATCH	EPOCHS	RMSE
2	1	100	100	**2.549**
4	1	100	100	**2.55**
6	1	100	100	**2.552**
8	1	100	100	**2.553**
10	1	100	100	**2.548**
15	1	100	100	**2.553**
20	1	100	100	**2.56**
30	1	100	100	**2.549**
40	1	100	100	**2.552**
50	1	100	100	**2.54**
75	1	100	100	**2.532**
100	1	100	100	**2.536**

Table 17: LSTM results for different number of Neurons (1 hidden layer, stateless, stationary input).

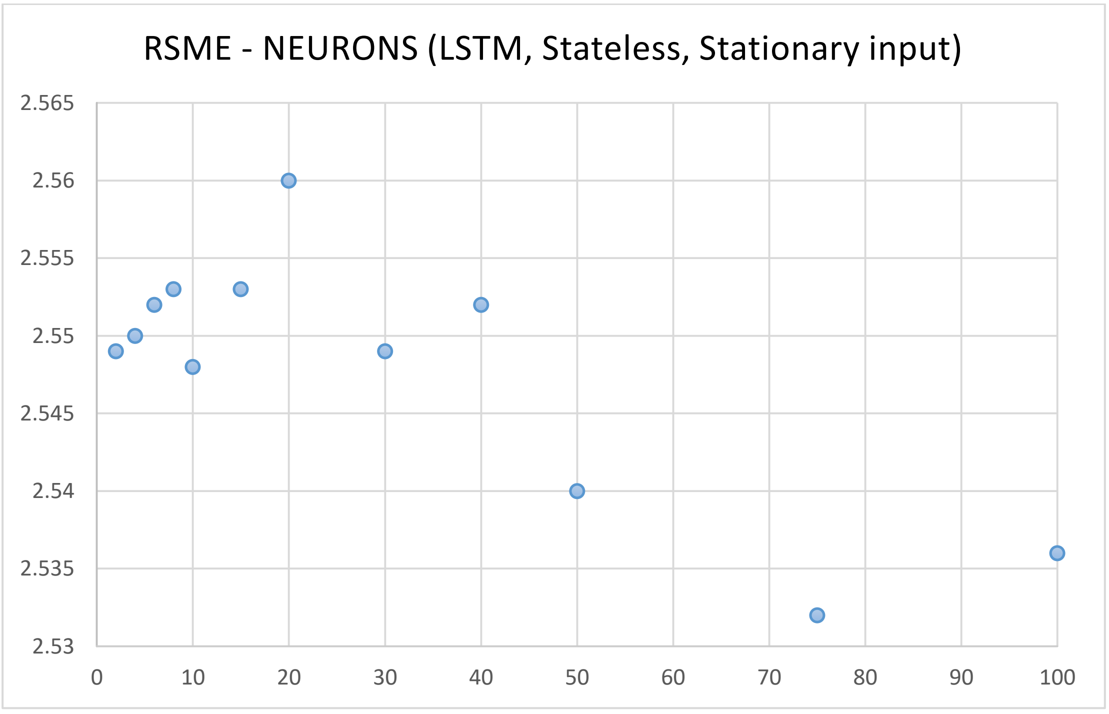


Figure 35: Plot of RMSE against Number of Neurons (LSTM, 1 hidden layer, stateless, stationary input).

In the last experiment, a stateful LSTM network is going to be investigated, which receives as input the stationary dataset.

NEURONS	WINDOW	BATCH	EPOCHS	RMSE
1	1	100	100	**2.491**
2	1	100	100	**2.491**
4	1	100	100	**2.488**
6	1	100	100	**2.49**
8	1	100	100	**2.489**
10	1	100	100	**2.488**
15	1	100	100	**2.49**
20	1	100	100	**2.489**
30	1	100	100	**2.501**
40	1	100	100	**2.489**
50	1	100	100	**2.489**
75	1	100	100	**2.488**
100	1	100	100	**2.488**

Table 18: LSTM results for different number of Neurons (1 hidden layer, stateful, stationary input).

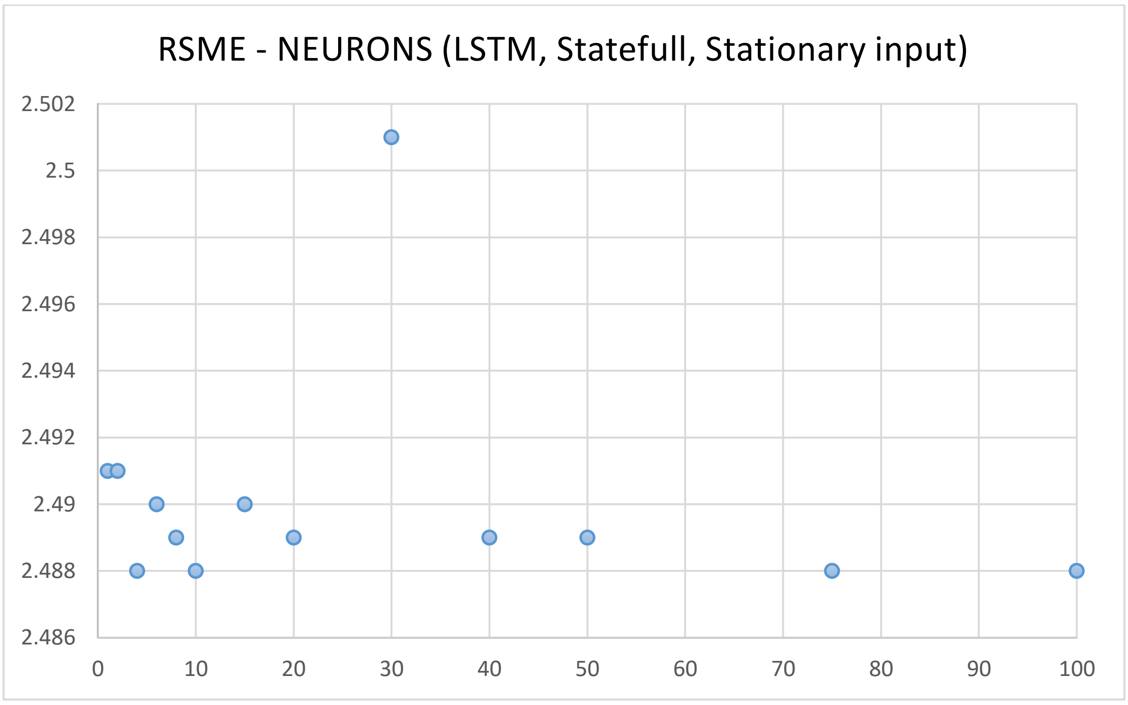


Figure 36: Plot of RMSE against Number of Neurons (LSTM, 1 hidden layer, stateful, stationary input).

From the results of the first two experiments the basic conclusion is that the input plays a significant role. The pre-processing of the data is of vital importance for the performance of the Neural Network. The stateful LSTM shows better results from the stateless LSTM for the same input. Although it has to be stressed out, that those differences are not worthwhile.

Since some experiments with the forecasting of velocity took place about the MLP, it was deemed proper to put into test LSTM too. Again the parameters were chosen intuitively. The best RMSE scores for the stateful LSTM and the stateless LSTM were 0.7 and 0.202 correspondingly. That means that their performance on forecasting velocity is worse than forecasting CO_2 Level. The stateful LSTM score is translated to 5.17% error, which is not good at all. The stateless LSTM error is corresponding to 1.49% error, which is nearly acceptable.

5.4 MLP and LSTM Results Comparison

The following diagrams illustrate the best scores achieved by each Neural Network architecture that was evaluated in the framework of this master thesis. From the comparison of these results, the following conclusions can be derived.

- The performance of a Neural Network is depending on the tuning of its parameters, the pre-processing of the training set, and the nature of the problem.
- As far as the forecasting of CO_2 Level is concerned, all the architectures that were investigated had excellent performance.
- At the forecasting of Velocity, the stateless LSTM had a good performance while the other architectures exhibit a poor performance.
- The stateful LSTM is worse than the stateless LSTM when each batch of training has data that are uncorrelated between them.

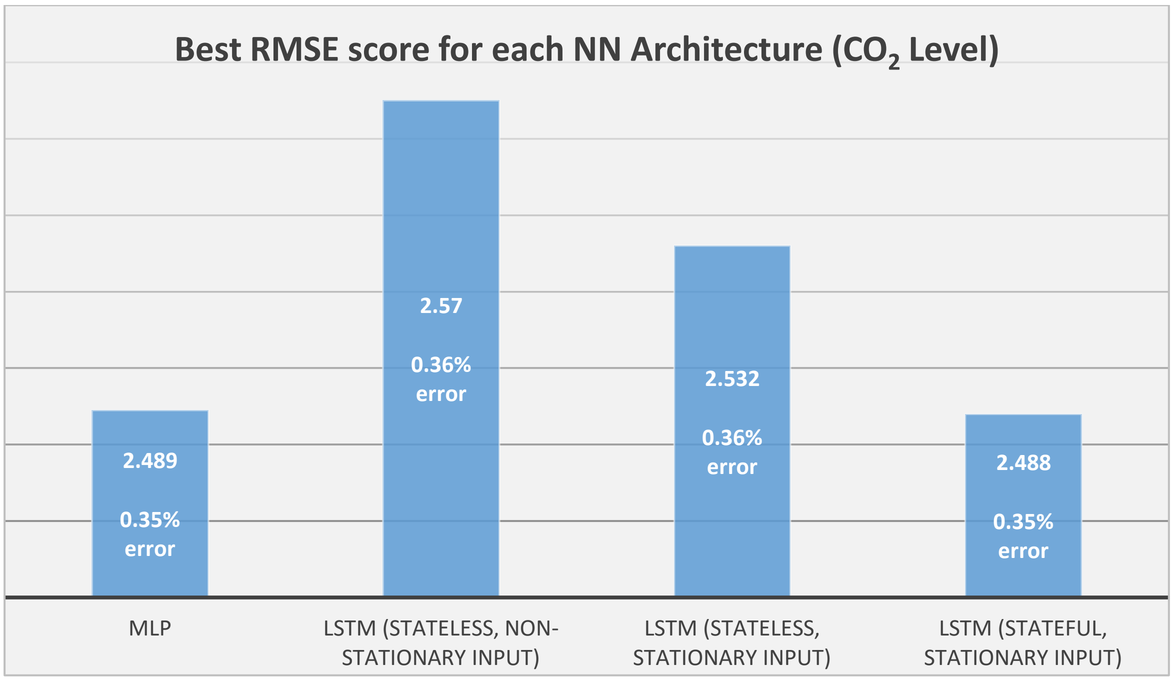


Figure 37: Lowest RMSE value for each NN Architecture (Forecasting CO_2 Level).

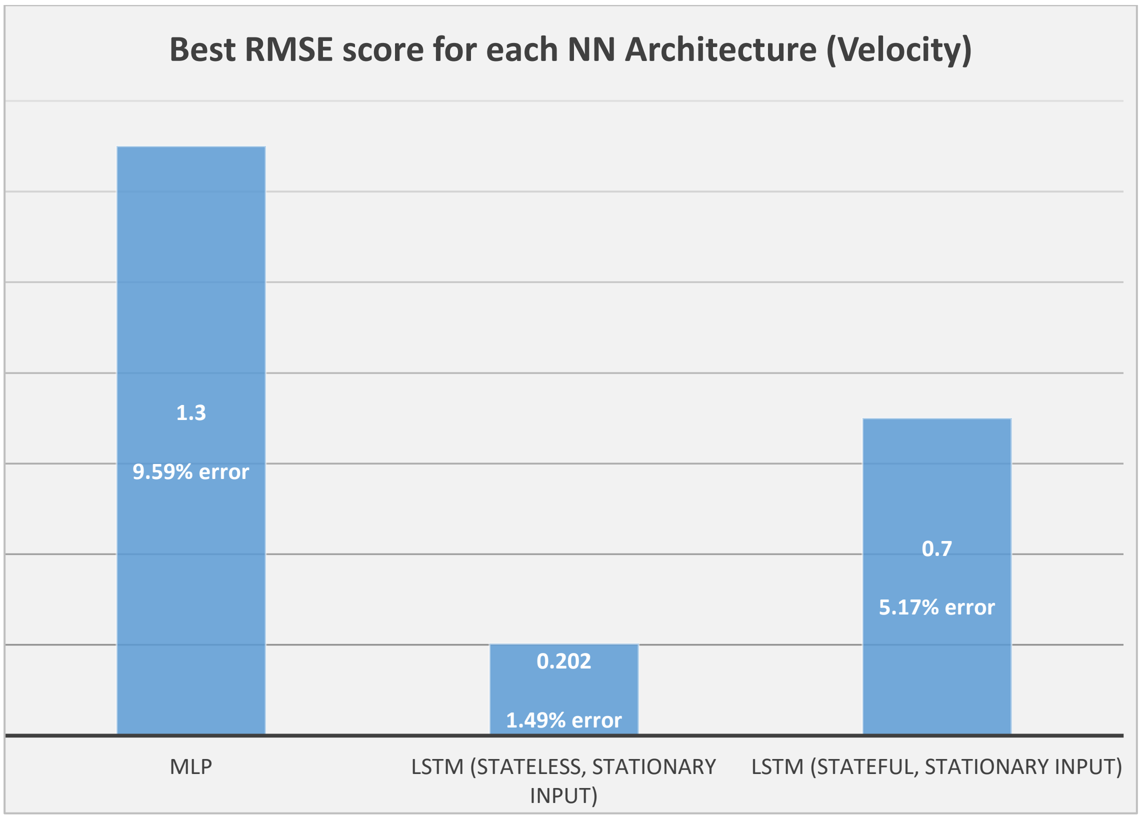


Figure 38: Lowest RMSE value for each NN Architecture (Forecasting Velocity).

The available dataset comprises 54758 measurements for each quantity and was split into two parts. The first 40% of the measurements (21903) were the training set while the other 60% (32855) were the test set.

In order to achieve reproducibility, since the initialization of the weights of a Neural Network are randomly generated and that causes slights deviations on the results, the same constant number where provided as seed on every experiment.

Chapter 6

SiteWhere

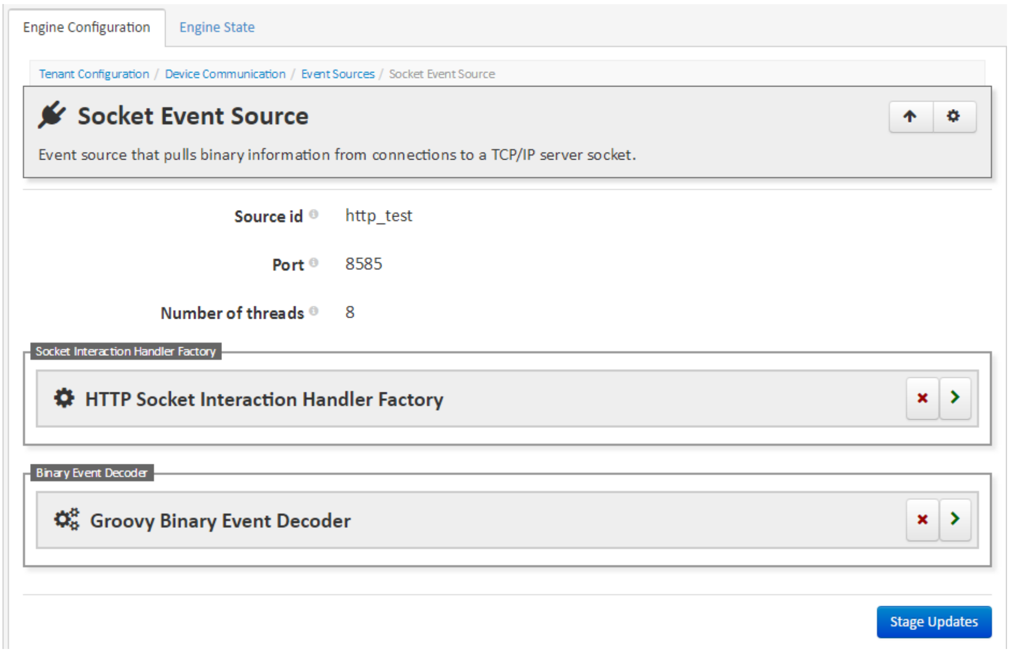
As it was already mentioned, *SiteWhere* is an open source IoT platform that has been used for the completion of this work. This platform is open source and offers a wide spectrum of capabilities. However, this work is focused on the collection and storage of data and their processing, and not the implementation of the large variety of protocols available in the platform.

6.1 Collection and Storage

The first step is the collection and storage of the data. A fundamental notion of the platform is the event. Every inbound message is considered to be *SiteWhere* event. One type of data that can be received by the platform and generate an event, after the necessary processing of the payload, is the data of an HTTP request. In order, for *SiteWhere*, to be able to accept HTTP requests, a socket event source has to be configured.

The whole idea is based on the principle of the connection between server and client via a particular port. The socket event source creates a server socket that monitors a specific port. The communication between server and client is managed by a socket interaction handler, which returns a payload. That payload is forwarded to a specific decoder that is defined by the user, in order to produce the *SiteWhere* event.

By using the Administrative Application of *Sitewhere* a socket event source were created with Source Id = “http_test”, Port = 8585, and Number of Threads = 8. After the configuration of the socket, the socket interaction handler had to be chosen. *SiteWhere* provides the “HTTP Socket Interaction Handler Factory” for HTTP interactions. The last step was to determine the decoder that will be used to parse the particular inbound HTTP payload into a *SiteWhere* event. A Groovy script was created to play the role of the decoder.


Engine Configuration | Engine State

Tenant Configuration / Device Communication / Event Sources / Socket Event Source

Socket Event Source

Event source that pulls binary information from connections to a TCP/IP server socket.

Source id	http_test
Port	8585
Number of threads	8

Socket Interaction Handler Factory

HTTP Socket Interaction Handler Factory

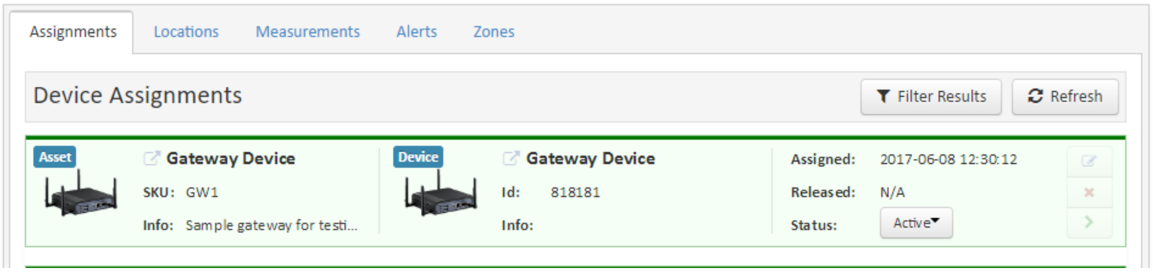
Binary Event Decoder

Groovy Binary Event Decoder

Stage Updates

Figure 39: Configuration of a Socket Event Source (from the Administrative Application).

SiteWhere can receive and record data, but in order to be able to do that a device, along with an assignment should be generated. A gateway device was generated, with a specific hardware id. Then a creation of an assignment for that gateway device followed, to enable it to accept requests.


Assignments | Locations | Measurements | Alerts | Zones

Device Assignments

Filter Results | Refresh

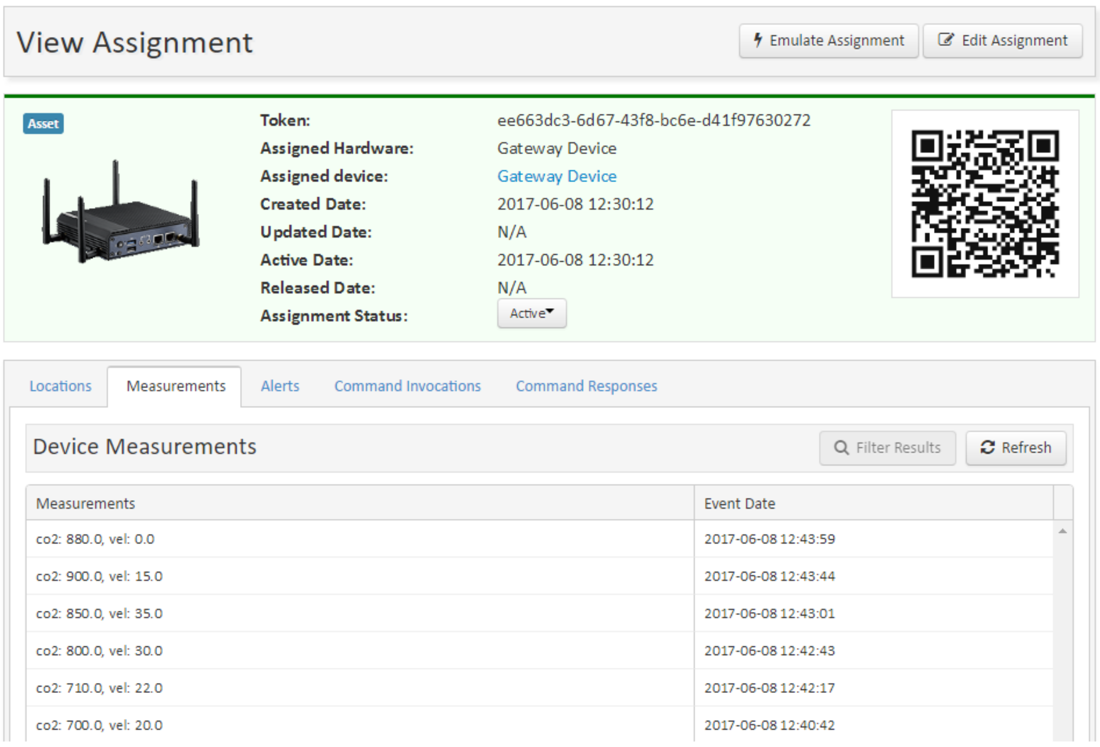
Asset — Gateway Device
SKU: GW1
Info: Sample gateway for testi...

Device — Gateway Device
Id: 818181
Info:

Assigned: 2017-06-08 12:30:12
Released: N/A
Status: Active

Figure 40: Device assignment (from the Administrative Application).

Testing was required to verify that the socket was operating as intended. The construction of a Junit test was preferred, for the convenience it provides, to change it and execute it dynamically from an IDE (IntelliJ IDEA was used). A sample JSON payload was generated and was sent using the HTTP protocol. The payload of the HTTP request was parsed from the Groovy script, which had the role of the decoder, and the wanted fields of the JSON payload were extracted and recorded as *SiteWhere* event.



Token:	ee663dc3-6d67-43f8-bc6e-d41f97630272
Assigned Hardware:	Gateway Device
Assigned device:	Gateway Device
Created Date:	2017-06-08 12:30:12
Updated Date:	N/A
Active Date:	2017-06-08 12:30:12
Released Date:	N/A
Assignment Status:	Active

Measurements	Event Date
co2: 880.0, vel: 0.0	2017-06-08 12:43:59
co2: 900.0, vel: 15.0	2017-06-08 12:43:44
co2: 850.0, vel: 35.0	2017-06-08 12:43:01
co2: 800.0, vel: 30.0	2017-06-08 12:42:43
co2: 710.0, vel: 22.0	2017-06-08 12:42:17
co2: 700.0, vel: 20.0	2017-06-08 12:40:42

Figure 41: The measurements that were sent as HTTP request, have been successfully parsed and stored.

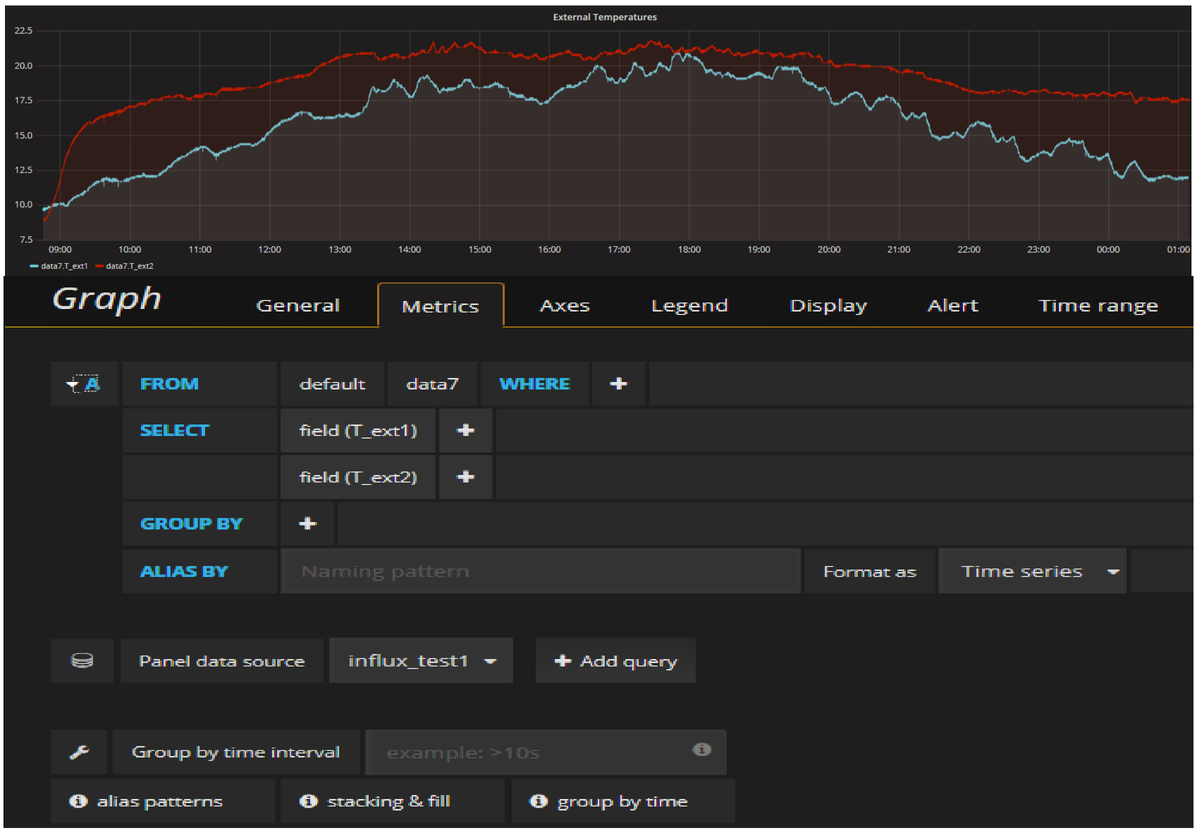
6.2 Data Processing

Since the collection and the data storage were successful, the next step was the processing of the collected data. SiteWhere, by default stores the event data and the device management data into MongoDB. Although, the option of using InlfuxDB to store the event data is available. Those databases are both NoSQL databases. Queries in MongoDB are entirely different from the standard SQL queries, while in InfluxDB, they have some differences but they also present some similarities to SQL queries.

The processing of the data was divided into three parts. The first involves visualization of the stored data with the help of Grafana, which has a build-in connectivity with InfluxDB. The second part includes the extraction of the data from both databases and the appliance of a clustering algorithm, while the third includes the forecasting with an MLP Neural Network. The processing of the data from both second and third part is going to take place on a Spark instance.

6.2.1 Visualization

Grafana is a data visualization tool. A local instance was installed that could be accessed from a browser. A data source was created that connected Grafana with the InfluxDB database that had the measurements. A new dashboard was created to represent the stored data in a graphical way.



FROM default data7 WHERE +
SELECT field (T_ext1) +
field (T_ext2) +
GROUP BY +
ALIAS BY Naming pattern Format as Time series
Panel data source influx_test1
Group by time interval example: >10s

Figure 42: Grafana's visual query editor for InfluxDB, and visualization of query's results. (query: SELECT T_ext1, T_ext2 FROM data7)

Since an InfluxDB database was chosen as data source, Grafana allows the user to query for data, using the InfluxQL, by a visual query editor. The results that are returned from the query are visualized. A graphic plot for every quantity of the dataset was decided to be made. It should be noticed that Grafana has a great number of options to customize the visualization of the data.

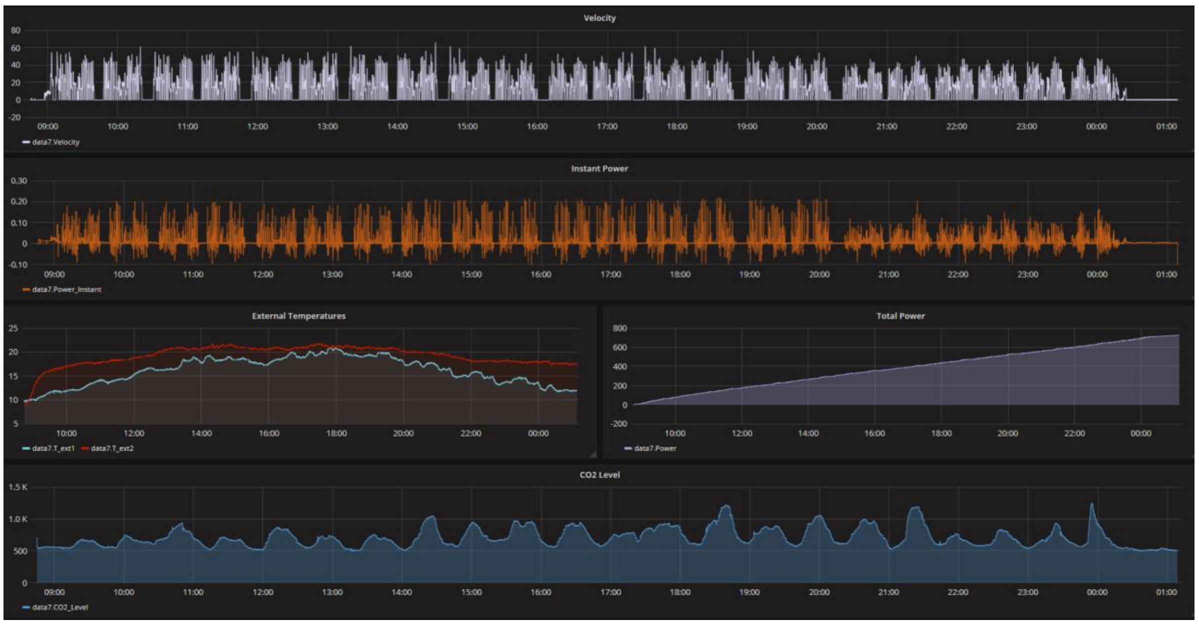


Figure 43: Plots for Velocity, Instant Power, External Temperature 1, External Temperature 2, Total Power, and CO2 Level for the duration of a single day.

6.2.2 Clustering

Since Spark is equipped with a library [62] that implements the KMeans algorithm, it was decided to use that implementation for the clustering of the data. The KMeans that used on the "local" process was an implementation from scikit-learn [56]. The scikit Kmeans runs the clustering algorithm 10 times by default and chooses the best output in terms of inertia, also known as the "within-cluster sum of squares criterion". On the contrary, the Spark Kmeans only runs the algorithm once.

Due to the different implementation of the algorithm, there were different results for some experiments. Another issue is that Spark Kmeans takes as an argument a “seed” which is equals “None” by default. But changing this “seed” alters the results of the algorithm. In the table below a scenario where depending on the “seed” is presented, the results of scikit Kmeans compared with the ones of Spark KMeans were identical (for seed=0), very similar (for seed=None) or quite different (for seed=8).

ROUTES	SPARK	SPARK (0)	SPARK (8)	SCIKIT
1	2	2	1	2
2	2	2	2	2
3	1	1	0	1
4	0	0	0	0
5	0	0	0	0
6	0	0	0	0
7	0	0	0	0
8	0	0	0	0
9	1	1	0	1
10	2	1	2	1
11	2	2	2	2

Table 19: Clustering results from KMeans algorithm for 3 clusters on "Full Route", Feature Vector: [T1, T2]. Spark implementation with seed=None, 0, 8 and Scikit implementation.

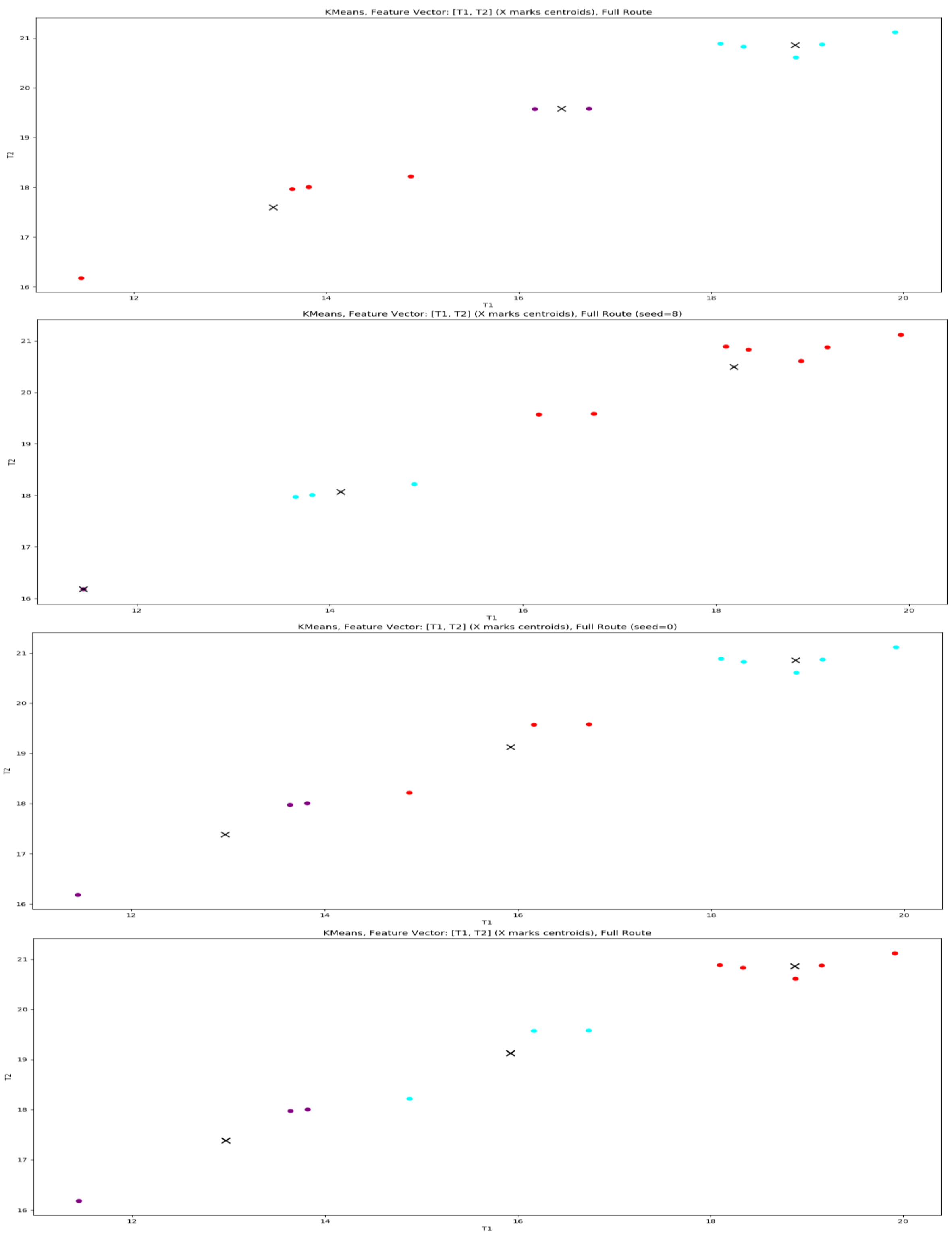
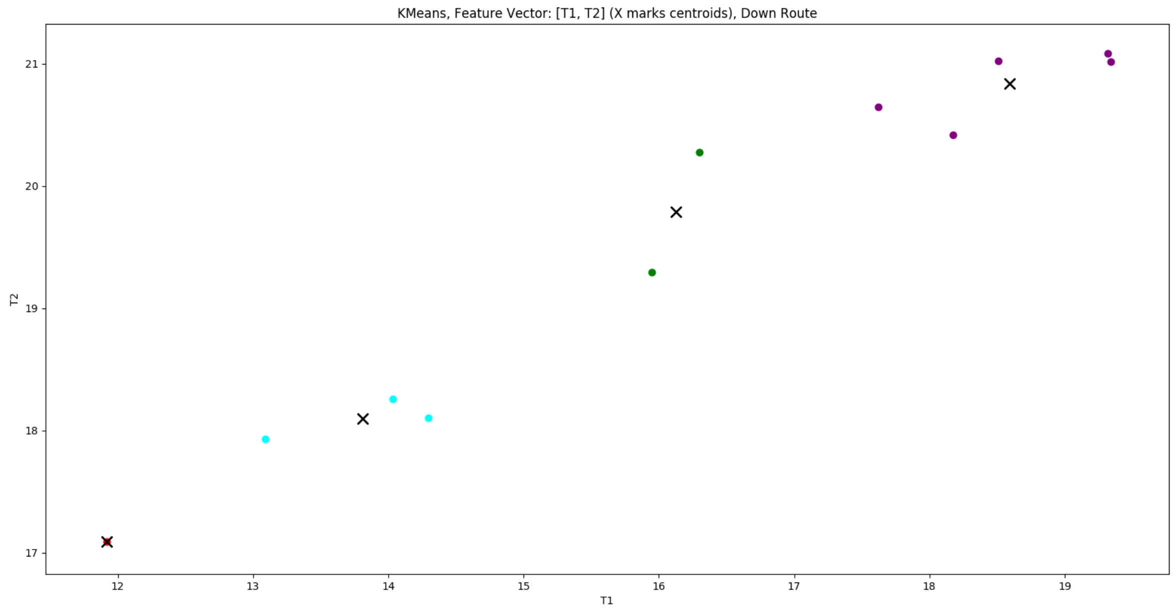


Figure 44: Plot of KMeans results for 3 clusters on "Full Route". Feature Vector: [T1, T2]. 1st (Top): Spark KMeans (seed=None). 2nd: Spark KMeans (seed=8). 3rd: Spark KMeans (seed=0). 4th (Bottom): Scikit KMeans.

All the experiments that were conducted with the scikit KMeans for feature vector: [T1, T2], were replicated with Spark KMeans with seed=0. The results for the “Up Route” were identical for each number n of clusters (n=2, 3, 4). The “Full Route” had only 1 difference when n=2, was identical for n=3, but presented an important variation for n=4. As far as the “Down Route” is concerned, it exhibited identical results for n=2 but serious differences for n=3 and n=4.



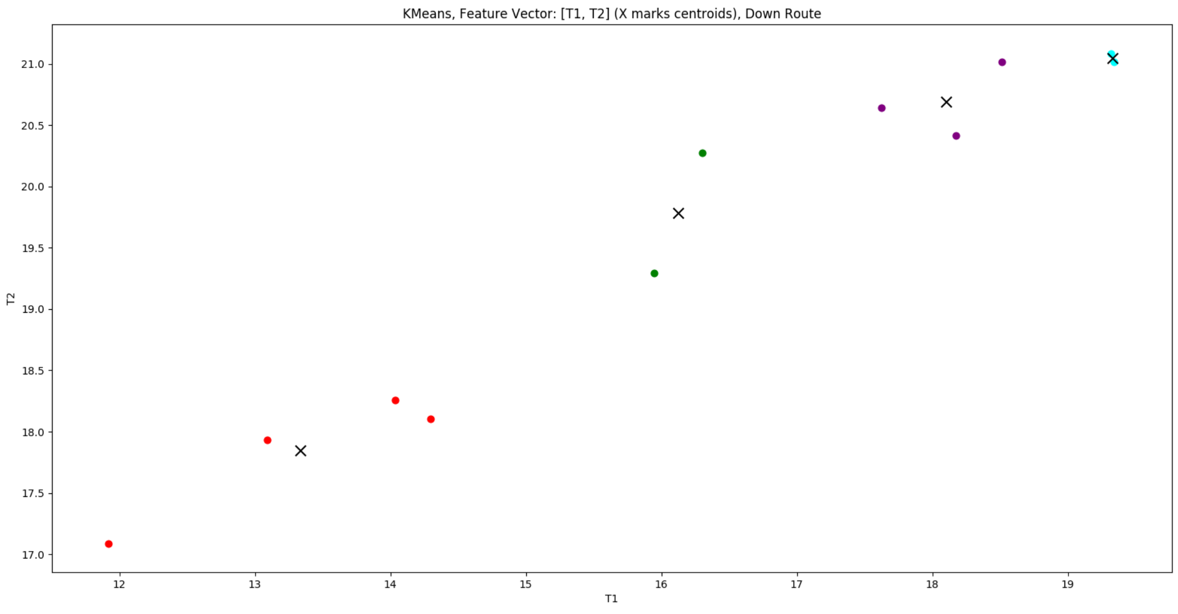


Figure 45: Feature Vector:[T1,T2], n=4, "Down Route". Up: Scikit KMeans. Down: Spark KMeans (seed=0).

6.2.3 Forecasting

Since Spark does not have available a library for Forecasting the same code that was used "locally" was loaded on a Spark instance and was executed there. The results were identical, as expected, since the code was the same, and the random seed for the initialization of the weights was the same as well.

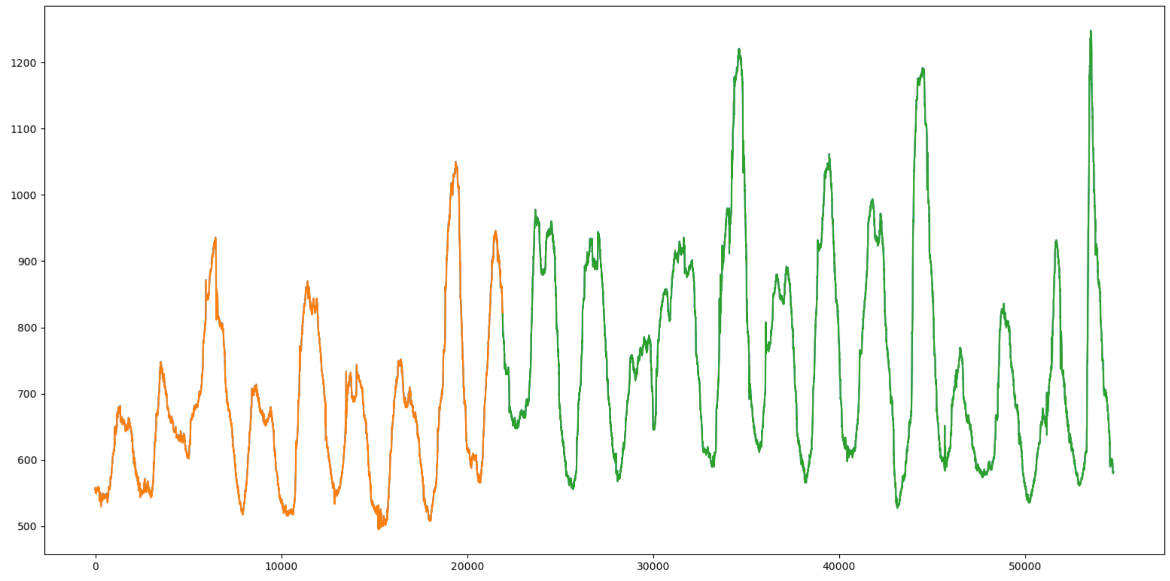
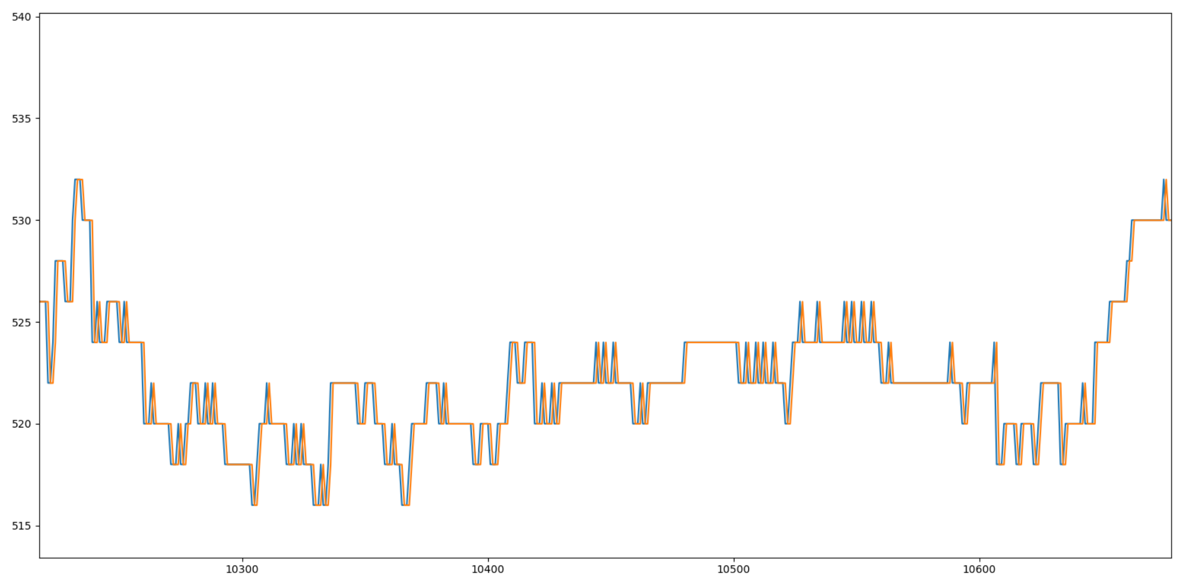


Figure 46: Forecasting CO_2 Level with an MLP NN (1 layer, 4 Neurons). The blue line is the actual measurements, while the orange line is the prediction on the training dataset, and the green one on the testing dataset. The measurements are from the duration of a day.

Due to the small error there is no distinction between the real values and the forecasted ones when the plot has all the measurements from the duration of a day. For that reason the figures below are plots within a much smaller window of time. It is observed that despite the small error between the actual and the forecasted value, the trend of both real measurements and forecasted ones is identical.



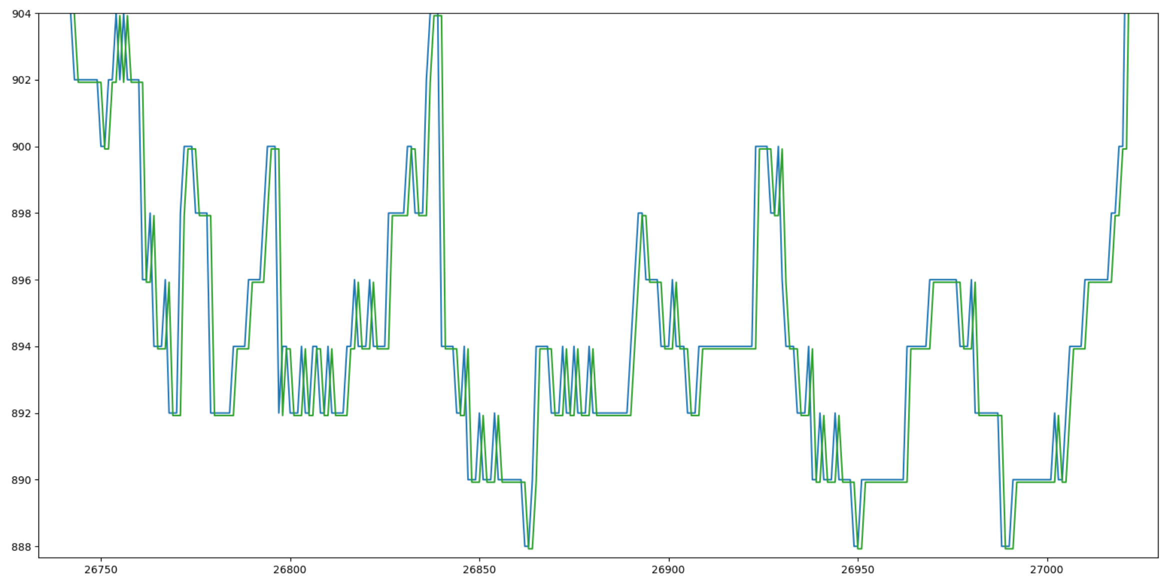


Figure 47: Two different, zoomed in parts from the above figure, in order to distinguish visually the real values from the predictions. The blue line is the actual measurements, while the orange line is the prediction on the training dataset, and the green one on the testing dataset.

Chapter 7

Conclusions

In this master thesis, a general solution focusing on the IoT management was described along with the presentation of *SiteWhere*, an open-source IoT platform. This report provided an introduction to the *Cloud*, including a short survey on platforms and frameworks that are capable to address the *Big Data Challenge*. Following this the processing of a dataset was performed, which was composed by measurements, of various quantities, from sensors that were installed on a train.

The first part of the processing was the clustering with 4 different algorithms in order to find the routes of the train that share similar conditions. Three different interpretations of the "route" were considered (Up Route, Down Route, Full Route). One of the algorithms (DBSCAN) did not work because of the small number of routes (11 routes). The other three algorithms (KMeans, Birch, Mean Shift) had identical or very similar results on the 91% of the cases. An experiment with a high number of routes should take place in order to derive safer conclusions.

The second part was forecasting of 2 different quantities (CO_2 Level, Velocity) with the help of an MLP Neural Network and an LSTM Neural Network. Both NNs achieved the same score when they had to forecast the CO_2 Level. LSTM has been proven to be superior to MLP (1.49% error against 9.59% error), when the quantity forecasted was Velocity. It was also shown that the pre-processing of the data affects considerably the training of a Neural Network and consequently its performance on forecasting.

Finally, the main functionality of SiteWhere, collection and storage of data as well as retrieval and processing, was investigated. A case study of using http requests to "feed" the server with data, of retrieving the data from the database and visualize them through Grafana, as well as of processing those data on a Spark instance, was provided.

Spark provided a library with an implementation of KMeans. Since that implementation was different from the one that was used locally (scikit library), the results of the clustering were not always the same. On the other hand, since the code of the MLP Neural Network was the same, the results on forecasting were identical.

References

[1] [Online]. Available: http://www.sitewhere.org/.

[2] *NIST SP 800-145, "A NIST definition of cloud computing".*

[3] [Online]. Available: https://ornot.ca/2009/08/04/cloud-computing-paradigm-chart/.

[4] [Online]. Available: https://en.wikipedia.org/wiki/Big_data.

[5] [Online]. Available: http://www.nosql-database.org/.

[6] A. H. Al Hinai, "A Performance Comparison of SQL and NoSQL Databases for Large Scale Analysis of Persistent Logs," Uppsala, 2016.

[7] M. Kanehisa, S. Goto, Y. Sato, M. Kawashima, M. Furumichi and M. Tanabe, "Data, information, knowledge and principle: back to metabolism in KEGG," *Nucleic Acids Research,* vol. 42, no. D1, p. D199, 2014.

[8] S. Sharma, "An Extended Classification and Comparison of NoSQL Big Data Models," *CoRR,* vol. abs/1509.08035, 2015.

[9] J. ZHENG, Y. YE, T. LIU, C. DAI and H. WANG, "Design of live video streaming,recording and storage system based on Flex,Red5 and MongoDB," *Journal of Computer Applications,* 2014.

[10] F. K. Putri, S. An, M. T. Purnaningtyas, H.-Y. Jeong and J. Kwon, "Finding Frequent Route of Taxi Trip Events Based on MapReduce and MongoDB," *KIPS Transactions on Software and Data Engineering,* vol. 4, no. 9, pp. 347-356, 2015.

[11] Y.-S. Kang, I.-H. Prak, J. Rhee and Y.-H. Lee, "MongoDB-Based Repository Design for IoT-Generated RFID/Sensor Big Data," *IEEE Sensors Journal,* vol. 16, no. 2, pp. 485-597, 2016.

[12] M. Mesiti, M. Re and G. Valentini, "Scalable Network-based Learning Methods for Automated Function Prediction based on the Neo4j Graph-database," Milano, 2013.

[13] S. Beis, S. Papadopoulos and Y. Kompatsiaris, "Benchmarking graph databases on the problem of Community Detection," in *New Trends in Database and Information Systems II: Selected papers of the 18th East European Conference on Advances in Databases and Information Systems and Associated Satellite Events*, Springer International Publishing, 2015, pp. 3-14.

[14] J. Z. Pan, Y. Ren, G. Montiel and Z. Wu, "Fast In-Memory Reasoner for Oracle NoSQL Database EE: Uncover Hidden Relationships that Exist in Your Enterprise Data," in *International Semantic Web Conference*, 2014.

[15] H. Kondylakis, A. Fountouris and D. Plexousakis, "Efficient Implementation of Joins over Cassandra DBs," in *International Conference on Extending Database Technology (EDBT/ICDT)*, Bordeaux, France, 2016.

[16] M. Ben Brahim, W. Drira, F. Filali and N. Hamdi, "Spatial data extension for Cassandra NoSQL database," *Journal of Big Data,* vol. 3, no. 1, p. 11, 2016.

[17] Du N, Z. J., M. Zhao, X. D. and X. Y., "Spatio-Temporal Data Index Model of Moving Objects on Fixed Networks Using HBase," in *2015 IEEE International Conference on Computational Intelligence Communication Technology*, 2015.

[18] L. Antova, R. Baldwin, D. Bryant, T. Cao, M. Duller, J. Eshleman, Z. Gu, E. Shen, M. A. Soliman and F. M. Waas, "Datometry Hyper-Q: Bridging the Gap Between Real-Time and Historical Analytics," in *Proceedings of the 2016 International Conference on Management of Data*, San Francisco, California, USA, 2016.

[19] [Online]. Available: http://www.datastax.com/wp-content/themes/datastax-2014-08/files/NoSQL_Benchmarks_EndPoint.pdf.

[20] [Online]. Available: https://docs.microsoft.com/en-us/azure/documentdb/documentdb-nosql-vs-sql.

[21] E. Begoli, "A short survey on the state of the art in architectures and platforms for large scale data analysis and knowledge discovery from data," in *Proceedings of the WICSA/ECSA 2012 Companion Volume*, New York, NY, USA, 2012.

[22] [Online]. Available: https://en.wikipedia.org/wiki/Apache_Hadoop.

[23] [Online]. Available: https://en.wikipedia.org/wiki/Apache_Spark.

[24] [Online]. Available: https://spark.apache.org/docs/latest/cluster-overview.html.

[25] [Online]. Available: https://hortonworks.com/apache/storm/#section_2.

[26] [Online]. Available: https://en.wikipedia.org/wiki/Storm_(event_processor).

[27] [Online]. Available: https://www.linkedin.com/pulse/apache-storm-architecture-overview-chandan-prakash.

[28] [Online]. Available: https://www.tutorialspoint.com/apache_storm/apache_storm_cluster_architecture.htm.

[29] [Online]. Available: https://en.wikipedia.org/wiki/Esper_(software).

[30] [Online]. Available: http://www.espertech.com/esper/.

[31] Y. Guo, J. Rao, D. Cheng and X. Zhou, "iShuffle: Improving Hadoop Performance with Shuffle-on-Write," *IEEE Transactions on Parallel and Distributed Systems,* vol. PP, pp. 1-1, 2016.

[32] P. Qin, B. Dai, B. Huang and G. Xu, "Bandwidth-Aware Scheduling With SDN in Hadoop: A New Trend for Big Data," *IEEE Systems Journal,* vol. PP, no. 99, pp. 1-8, 2015.

[33] D. Pirozzi, V. Scarano, S. Begg, G. D. Sercey, A. Fish and A. Harvey, "Filter large-scale engine data using apache spark," in *2016 IEEE 14th International Conference on Industrial Informatics (INDIN)*, 2016.

[34] M. A. Alsheikh, D. Niyato, S. Lin, H. p. Tan and Z. Han, "Mobile big data analytics using deep learning and apache spark," *IEEE Network,* vol. 30, no. 3, pp. 22-29, 2016.

[35] [Online]. Available: http://www.cakesolutions.net/teamblogs/comparison-of-apache-stream-processing-frameworks-part-1.

[36] [Online]. Available: https://www.dezyre.com/article/spark-vs-hadoop-vs-storm/145.

[37] [Online]. Available: https://tech.zalando.com/blog/apache-showdown-flink-vs.-spark/.

[38] C.-K. Shieh, S.-W. Huang, L.-D. Sun, M.-F. Tsai and N. Chilamkurti, "A topology-based scaling mechanism for Apache Storm," *International Journal of Network Management,* 2016.

[39] J. S. v. d. Veen, B. v. d. Waaij, E. Lazovik, W. Wijbrandi and R. J. Meijer, "Dynamically Scaling Apache Storm for the Analysis of Streaming Data," in *2015 IEEE First International Conference on Big Data Computing Service and Applications*, 2015.

[40] G. Mylavarapu, J. Thomas and A. K. TK, "Real-Time Hybrid Intrusion Detection System Using Apache Storm," in *2015 IEEE 17th International Conference on High Performance Computing and Communications, 2015 IEEE 7th International Symposium on Cyberspace Safety and Security, and 2015 IEEE 12th International Conference on Embedded Software and Systems*, 2015.

[41] N. Zacheilas, V. Kalogeraki, N. Zygouras, N. Panagiotou and D. Gunopulos, "Elastic complex event processing exploiting prediction," in *2015 IEEE International Conference on Big Data (Big Data)*, 2015.

[42] Z. Zhuang, T. Feng, Y. Pan, H. Ramachandra and B. Sridharan, "Effective Multi-stream Joining in Apache Samza Framework," in *2016 IEEE International Congress on Big Data (BigData Congress)*, 2016.

[43] T. Feng, Z. Zhuang, Y. Pan and H. Ramachandra, "A memory capacity model for high performing data-filtering applications in Samza framework," in *2015 IEEE International Conference on Big Data (Big Data)*, 2015.

[44] M. Junghanns, A. Petermann, N. Teichmann, K. Gomez and E. Rahm, "Analyzing Extended Property Graphs with Apache Flink," in *Proceedings of the 1st ACM SIGMOD Workshop on Network Data Analytics*, 2016.

[45] M. Falkenthal, U. Breitenbucher, K. Kepes, F. Leymann, M. Zimmermann, M. Christ, J. Neuffer, N. Braun and A. W. Kempa-Liehr, "OpenTOSCA for the 4th Industrial Revolution: Automating the Provisioning of Analytics Tools Based on Apache Flink," in *Proceedings of the 6th International Conference on the Internet of Things*, 2016.

[46] W. F. Domoney, N. Ramli, S. Alarefi and S. D. Walker, "Smart city solutions to water management using self-powered, low-cost, water sensors and apache spark data aggregation," in *2015 3rd International Renewable and Sustainable Energy Conference (IRSEC)*, 2015.

[47] [Online]. Available: https://yahooeng.tumblr.com/post/135321837876/benchmarking-streaming-computation-engines-at.

[48] Y. Wang, "Stream Processing Systems Benchmark: StreamBench," Espoo, 2016.

[49] S. Theodoridis, Machine Learning A Bayesian and Optimization Perspective, Elsevier, 2015.

[50] S. García, J. Luengo and F. Herrera, "Tutorial on Practical Tips of the Most Influential Data Preprocessing Algorithms in Data Mining," *Knowledge-Based Systems,* vol. 98, no. C, pp. 1-29, 2016.

[51] S. García, S. Ramírez-Gallego, J. Luengo, J. M. Benítez and F. Herrera, "Big data preprocessing: methods and prospects," *Big Data Analytics,* vol. 1, no. 1, 2016.

[52] S. Papadimitriou, J. Sun, C. Faloutos and P. S. Yu, "Dimensionality Reduction and Filtering on Time Series Sensor Streams," in *Managing and Mining Sensor Data*, C. C. Aggarwal, Ed., Boston, MA, Springer US, 2013, pp. 103-141.

[53] [Online]. Available: http://machinelearningmastery.com/time-series-forecasting/.

[54] [Online]. Available: http://www.cardinalpath.com/forecasting-with-machine-learning-techniques/.

[55] [Online]. Available: http://www.citura.fr/ftp/document/backlight-plan-tramway-armoire-technique-500x1025-web.pdf.

[56] [Online]. Available: http://scikit-learn.org/stable/modules/clustering.html.

[57] S. Haykin, Neural Networks and Learning Machines, New York: Prentice Hall/Pearson, 2009.

[58] [Online]. Available: https://cs231n.github.io/convolutional-networks/.

[59] Y. Bengio, P. Simard and P. Frasconi, "Learning Long-term Dependencies with Gradient Descent is Difficult," *Trans. Neur. Netw.,* vol. 5, no. 2, pp. 157-166, March 1994.

[60] [Online]. Available: http://www.cs.iusb.edu/~danav/teach/c463/12_nn.html.

[61] [Online]. Available: https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[62] [Online]. Available: https://spark.apache.org/docs/latest/api/python/pyspark.ml.html#.

[63] [Online]. Available: https://www.gartner.com/newsroom/id/1731916.

Appendix

A part of the total code that was used for this master thesis, but not every single line that was written, is introduced below. It was decided to present the clustering procedure with the usage of KMeans algorithm, the forecasting with both MLP and stateless LSTM Neural Networks, and the code for running KMeans on a Spark instance.

Code 1: Program, which loads the info for each route from a csv file, performs a clustering with KMeans algorithm for 3 different feature vectors as input, and writes the results in a csv file.

```python
import pandas as pd
import numpy as np
import csv

from sklearn.cluster import KMeans

import logging
logging.basicConfig(format='%(asctime)s : %(levelname)s : %(message)s', le
vel=logging.INFO)


DELIMITER = '\t'
NUMBER_OF_CLUSTERS = 3
OUTPUT_FILE = 'Kmeans_4Clusters_Information_Down.csv'
### UP AND DOWN TOGETHER
INPUT_FILE_2 = 'MeanValuesPerRouteFull.csv'
### UP
INPUT_FILE_3 = 'MeanValuesPerRouteUp.csv'
### DOWN
INPUT_FILE_4 = 'MeanValuesPerRouteDown.csv'


# (ROUTES) UNCOMMENT THE ONE YOU NEED
#df = pd.read_csv(INPUT_FILE_2, delimiter = DELIMITER)
#df = pd.read_csv(INPUT_FILE_3, delimiter = DELIMITER)
df = pd.read_csv(INPUT_FILE_4, delimiter = DELIMITER)

NUMBER_OF_ROUTES = len(df.T1_mean)

##############     MEAN VALUES CSV     ##############
T1 = np.asarray(df.T1_mean)
T2 = np.asarray(df.T2_mean)
Velocity = np.asarray(df.Velocity_mean)
CO2 = np.asarray(df.CO2_mean)
Power_Sum = np.asarray(df.Power)

```

```

############    KMEANS CLUSTERING    ############
# 1) T1, T2 Clustering
Routes = []  # 2D array, row=route, col=features

for i in range(NUMBER_OF_ROUTES):
    feature_vec = []
    feature_vec.append(T1[i])
    feature_vec.append(T2[i])
    Routes.append(feature_vec)

data = np.asarray(Routes)
kmeans = KMeans(n_clusters=NUMBER_OF_CLUSTERS)
kmeans.fit(data)
predictions1 = kmeans.predict(data)

#2) T1, T2, CO2 Clustering
Routes2 = []  # 2D array, row=route, col=features

for i in range(NUMBER_OF_ROUTES):
    feature_vec = []
    feature_vec.append(T1[i])
    feature_vec.append(T2[i])
    feature_vec.append(CO2[i])
    Routes2.append(feature_vec)
data2 = np.asarray(Routes2)
kmeans2 = KMeans(n_clusters=NUMBER_OF_CLUSTERS)
kmeans2.fit(data2)
predictions2 = kmeans2.predict(data2)

#3) T1, T2, Velocity, CO2 Clustering
Routes3 = []  # 2D array, row=route, col=features

for i in range(NUMBER_OF_ROUTES):
    feature_vec = []
    feature_vec.append(T1[i])
    feature_vec.append(T2[i])
    feature_vec.append(Velocity[i])
    feature_vec.append(CO2[i])
    Routes3.append(feature_vec)

data3 = np.asarray(Routes3)
kmeans3 = KMeans(n_clusters=NUMBER_OF_CLUSTERS)
kmeans3.fit(data3)
predictions3 = kmeans3.predict(data3)


##############     CSV CREATION     ##############
with open(OUTPUT_FILE, 'w', newline='') as csv_f:
    writer=csv.writer(csv_f, delimiter='\t')
    writer.writerow(['Route', 'Cluster', 'Cluster2', 'Cluster3'])
    for i in range(NUMBER_OF_ROUTES):
        writer.writerow([i, predictions1[i], predictions2[i], predictions3[i]])
```

Code 2: Program, which loads the data from a csv file, creates an MLP Neural Network, trains it, and measures its performance on forecasting.

```python
import numpy as np
import pandas as pd
import math

from keras.models import Sequential
from keras.layers import Dense

import logging
logging.basicConfig(format='%(asctime)s : %(levelname)s : %(message)s', le
vel=logging.INFO)


### FUNCTION -> Create the NN input ###
def create_NN_input(dataset, steps_back=1):
    dataX, dataY = [], []
    for i in range(len(dataset)-steps_back-1):
        temp = dataset[i:(i+steps_back),0]
        dataX.append(temp)
        dataY.append(dataset[i+steps_back, 0])
    return np.array(dataX), np.array(dataY)
###############################################


# Fix random seed for reproducibility
np.random.seed(8)

# Configurations
TRAIN_SET_SIZE = 0.4
NEURONS = 4
NEURONS_2 = 6
BATCH_SIZE = 100
EPOCHS = 100
CSV_START = 1050
CSV_END = 55810
STEPS_BACK = 1
DELIMITER = ','
INPUT_FILE = 'nn_csv_input.csv'

# usecols -> the column with the data I want
# 8-> Velocity, 9-> CO2, 11-> Instant_Power
df = pd.read_csv(INPUT_FILE, delimiter = DELIMITER, usecols=[9])
temp = df.values
temp = temp.astype('float32')
dataset = temp[CSV_START:CSV_END]

# Split into TRAIN set and TEST set
train_set_size = int(len(dataset)*TRAIN_SET_SIZE)
test_set_size = len(dataset) - train_set_size
train_set = dataset[0:train_set_size,:]
test_set = dataset[train_set_size:len(dataset),:]

# Create the Training and Testing Dataset
trainX, trainY = create_NN_input(train_set, STEPS_BACK)
testX, testY = create_NN_input(test_set, STEPS_BACK)
```

```

# Create and Fit the Multilayer Perceptron model
model = Sequential()
model.add(Dense(NEURONS, input_dim=STEPS_BACK, activation='relu'))
#model.add(Dense(NEURONS_2, input_dim=STEPS_BACK, activation='relu'))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(trainX, trainY, epochs=EPOCHS, batch_size=BATCH_SIZE, verbose=0)

# Estimate model performance
trainScore = model.evaluate(trainX, trainY, verbose=0)
print('Train Score: %.5f MSE (%.5f RMSE)' % (trainScore, math.sqrt(trainScore)))
testScore = model.evaluate(testX, testY, verbose=0)
print('Test Score: %.5f MSE (%.5f RMSE)' % (testScore, math.sqrt(testScore)))

# Predictions
trainPredict = model.predict(trainX)
testPredict = model.predict(testX)

# Shift train predictions for plotting
trainPredictPlot = np.empty_like(dataset)
trainPredictPlot[:, :] = np.nan
trainPredictPlot[STEPS_BACK:len(trainPredict)+STEPS_BACK, :] = trainPredict

# Shift test predictions for plotting
testPredictPlot = np.empty_like(dataset)
testPredictPlot[:, :] = np.nan
testPredictPlot[len(trainPredict)+(STEPS_BACK*2)+1:len(dataset)-1, :] = testPredict

# plot real data and predictions
plt.plot(dataset)
plt.plot(trainPredictPlot)
plt.plot(testPredictPlot)
plt.show()
```

Code 3: Program, which loads the data from a csv file, creates an LSTM Neural Network, trains it, and measures its performance on forecasting.

```python
import numpy as np
import pandas as pd
import math

from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error

import logging
logging.basicConfig(format='%(asctime)s : %(levelname)s : %(message)s', level=logging.INFO)


### FUNCTION -> Create the NN input ###
def create_NN_input(dataset, steps_back=1):
    dataX, dataY = [], []
    for i in range(len(dataset)-steps_back-1):
        temp = dataset[i:(i+steps_back),0]
        dataX.append(temp)
        dataY.append(dataset[i+steps_back, 0])
    return np.array(dataX), np.array(dataY)


### FUNCTION -> Create the differenced series ###
def create_differenced_series(dataset, interval=1):
    diff_series = []
    for i in range(interval, len(dataset)):
        value = dataset[i] - dataset[i - interval]
        diff_series.append(value)
    return diff_series


### FUNCTION -> Invert the differenced series ###
def invert_differenced_series(dataset, diff_series):
    inverted_diff_series = []
    for i in range(len(diff_series)):
        value = dataset[i] + diff_series[i]
        inverted_diff_series.append(value)
    return np.array(inverted_diff_series)


### FUNCTION -> Transform trainY
def transform_Y_for_plot(dataset):
    tranformed_dataset = []
    tranformed_dataset.append(dataset)
    return np.array(tranformed_dataset)
###############################################


# Fix random seed for reproducibility
np.random.seed(8)

```

```python
INTERVAL = 1
TRAIN_SET_SIZE = 0.4
NEURONS = 10
BATCH_SIZE = 100
EPOCHS = 100
CSV_START = 1053
CSV_END = 55810
STEPS_BACK = 1
DELIMITER = ','
INPUT_FILE = 'nn_csv_input.csv'

# usecols -> the column with the data I want
# 8-> Velocity, 9-> CO2, 11-> Instant_Power
df = pd.read_csv(INPUT_FILE, delimiter = DELIMITER, usecols=[8])
temp = df.values
temp = temp.astype('float32')
dataset = temp[CSV_START:CSV_END]

# Transform the data to stationary
dataset_diff = create_differenced_series(dataset, INTERVAL)

# Scale the dataset
normalizer = MinMaxScaler(feature_range=(-1,1))
dataset_scaled = normalizer.fit_transform(dataset_diff)

# Split into TRAIN set and TEST set
train_set_size = int(len(dataset_scaled)*TRAIN_SET_SIZE)
test_set_size = len(dataset_scaled) - train_set_size
train_set = dataset_scaled[0:train_set_size,:]
test_set = dataset_scaled[train_set_size:len(dataset_scaled),:]

train_set_initial = dataset[0:train_set_size,:]
test_set_initial = dataset[train_set_size:len(dataset),:]


# Create the Training and Testing Dataset
trainX, trainY = create_NN_input(train_set, STEPS_BACK)
testX, testY = create_NN_input(test_set, STEPS_BACK)

trainX_initial, trainY_initial = create_NN_input(train_set_initial, STEPS_BACK)
testX_initial, testY_initial = create_NN_input(test_set_initial, STEPS_BACK)

# Reshape input to [samples, time steps, features]
trainX = np.reshape(trainX, (trainX.shape[0], 1, trainX.shape[1]))
testX = np.reshape(testX, (testX.shape[0], 1, testX.shape[1]))

# Create and Fit the LSTM Neural Network
model = Sequential()
model.add(LSTM(NEURONS, input_shape=(1, STEPS_BACK)))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(trainX, trainY, epochs=EPOCHS, batch_size=BATCH_SIZE, verbose=0)


```

```python
    # Predictions
    trainPredict = model.predict(trainX, batch_size=BATCH_SIZE)
    testPredict = model.predict(testX, batch_size=BATCH_SIZE)

    # Invert scaling for the predictions
    trainPredict = normalizer.inverse_transform(trainPredict)
    testPredict = normalizer.inverse_transform(testPredict)

    # Invert the initial transformation (revert the difference)
    trainPredict = invert_differenced_series(trainX_initial, trainPredict)
    testPredict = invert_differenced_series(testX_initial, testPredict)

    # Prepare trainY and testY fot testing
    trainY_initial = transform_Y_for_plot(trainY_initial)
    testY_initial = transform_Y_for_plot(testY_initial)


    # Estimate model performance
    trainScore = mean_squared_error(trainY_initial[0], trainPredict[:,0])
    print('Train Score: %.5f MSE (%.5f RMSE)' % (trainScore, math.sqrt(trainScore)))
    testScore = mean_squared_error(testY_initial[0,:-1], testPredict[:,0])
    print('Test Score: %.5f MSE (%.5f RMSE)' % (testScore, math.sqrt(testScore)))
```

Code 4: Program, which connects to an InfluxDB database, sends a query and collects the data. Loads extra information about the routes from a csv file and runs KMeans algorithm on a Spark instance. At the end, it exports the results in a csv file and plots the clusters.

```
from pyspark import SparkConf, SparkContext
from pyspark.sql import SparkSession

from pyspark.ml.linalg import Vectors
from pyspark.ml.clustering import KMeans

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import csv
from influxdb import InfluxDBClient

MASTER = "local[8]"
APP_NAME = "pyspark_test"

NUMBER_OF_CLUSTERS = 4
KMEANS_ITERATIONS = 100

DELIMITER = '\t'
NUMBER_OF_ROUTES = 11
INPUT_FILE = 'half_routes_down.xlsx'
INPUT_FILE_2 = 'half_routes_up.xlsx'
INPUT_FILE_3 = 'routes.xlsx'
OUTPUT_FILE = 'Kmeans_S_4_D.csv'

HOST = "localhost"
PORT = 8086
USER = "root"
PASSWORD = "root"
DBNAME = "test1"


def main(sc):

    # Load data from database
    client = InfluxDBClient(HOST, PORT, USER, PASSWORD, DBNAME)

    query = 'SELECT T_ext1, T_ext2 FROM data7'
    print("QUERY:", query)
    result = client.query(query)

    T_ext1 = []
    T_ext2 = []
    points = list(result.get_points(measurement='data7'))
    for point in points:
        T_ext1.append(point['T_ext1'])
        T_ext2.append(point['T_ext2'])

    # Load routes starting-ending time
    df = pd.read_excel(io=INPUT_FILE)
    Route_Start = np.asarray(df.Start)
    Route_Duration = np.asarray(df.Duration)

```

```python
    # Calculate Mean Values
    T1_mean = []
    T2_mean = []
    for i in range(NUMBER_OF_ROUTES): # for each route
        T1 = 0
        T2 = 0
        for j in range(Route_Duration[i]): # for every measurement in each route
            T1 = T1 + T_ext1[Route_Start[i]+j]
            T2 = T2 + T_ext2[Route_Start[i]+j]
        T1_mean.append(T1/Route_Duration[i])
        T2_mean.append(T2/Route_Duration[i])

    # SPARK KMeans
    data = []
    for i in range(NUMBER_OF_ROUTES):
        data.append((Vectors.dense(T1_mean[i], T2_mean[i]),))

    spark = SparkSession(sc)
    df_s = spark.createDataFrame(data, ["features"])
    kmeans = KMeans(k=NUMBER_OF_CLUSTERS, maxIter=KMEANS_ITERATIONS, seed=0)
    model = kmeans.fit(df_s)
    centroids = model.clusterCenters()
    transformed = model.transform(df_s).select("features", "prediction")
    rows = transformed.collect()


    ###############     CSV CREATION     ###############
    with open(OUTPUT_FILE, 'w', newline='') as csv_f:
        writer=csv.writer(csv_f, delimiter=DELIMITER)
        writer.writerow(['Route', 'Cluster'])
        for i in range(NUMBER_OF_ROUTES):
            writer.writerow([i, rows[i].prediction])


    ###############     PLOT     ###############
    fig, ax = plt.subplots()
    ax.set_title("KMeans, Feature Vector: [T1, T2] (X marks centroids), Down Route")
    ax.set_xlabel('T1')
    ax.set_ylabel('T2')

    for i in range(NUMBER_OF_CLUSTERS):
        ax.scatter(centroids[i][0], centroids[i][1], marker='x', s=100, linewidths=15, color='black', zorder=10)

    for i in range(NUMBER_OF_ROUTES):
        if rows[i].prediction == 0:
            ax.scatter(T1_mean[i], T2_mean[i], s=40, c='red')
        elif rows[i].prediction == 1:
            ax.scatter(T1_mean[i], T2_mean[i], s=40, c='purple')
        elif rows[i].prediction == 2:
            ax.scatter(T1_mean[i], T2_mean[i], s=40, c='cyan')
        else:
            ax.scatter(T1_mean[i], T2_mean[i], s=40, c='green')

    plt.show()


   ################################

if __name__ == "__main__":
    # Configure Spark
    conf = SparkConf().setAppName(APP_NAME).setMaster(MASTER)
    sc = SparkContext(conf=conf)

    main(sc)
```