Master Thesis

Framework for Automatic Selection of Analytic Platforms for Data mining Tasks

Osama Muazzen

Study Program: INFOTECH

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

Advisors: M.Sc. Manuel Fritz

Start Date: 06.05.2018

End date: 06.11.2018
Abstract

Data mining is becoming more important and can be applied to several domains. The importance of data mining refers to its ability to derive valuable knowledge from voluminous datasets. Nowadays, numerous analytic platforms are developed which execute different data mining algorithms such as Spark, Mahout, WEKA, etc. These analytic platforms differ by their characteristics, purposes, performance, and the manner of processing the data. The plethora of analytic platforms escalates the difficulty of selecting the most appropriate analytic platform that fit the needed data mining algorithm, the submitted dataset, and additional user-defined criteria. Several works were introduced in order to help users in selecting the appropriate analytic platform. These works are mainly benchmarks for evaluating the performance of these analytic platforms. However, these benchmarks have several issues regarding their objectivity and the considered measures. For example, several benchmarks focus only on the execution runtime as a performance indicator with completely ignoring other measures such as the consumption of resources.

This thesis introduces a novel approach to solve the problem of selecting the most appropriate analytic platform automatically for executing the needed data mining algorithm. The selection process depends on a hypothesis which is executing a data mining algorithm on similar datasets, using the same analytic platform, yields comparable performance. Depending on a previously conducted benchmark, the selection process compares the performance of different analytic platform and selects the one with the best performance according to a user-defined criterion, e.g., runtime of the data mining algorithm.

The introduced approach in this thesis aggregates several analytic platforms with various data mining algorithms in order to offer a single, abstract interface to several analytic platforms. Hence, it enables the user to execute data mining algorithms on different analytic platforms according to the user need. Moreover, this approach allows novice users without profound programming background to execute data mining algorithms easily by solely calling the algorithm with the required parameters via a REST API. Therefore, the proposed approach can be used with several programming languages. Furthermore, this approach can perform an automatic benchmark for several execution engines using different data mining algorithms and different datasets. The benchmark result is stored so it can be used for the automatic selection later on. The prototypical implementation of the introduced approach has a loosely coupled and extensible architecture that gives it the flexibility to add more execution engines and more data mining algorithms later on.

The introduced approach is evaluated on several analytic platforms such as Spark, Mahout, and WEKA along with several data mining algorithms from classification, clustering, and association rule discovery. The experimental results unveil that automatic selection of the analytic platform can save up to 98.45% of the execution time of the data mining algorithm in some cases. The proposed selection process achieves accuracy of 87.00% in selecting the analytic platform with best performance according to the runtime criterion. Selecting the most appropriate analytic platform causes a runtime overhead. However, this overhead is negligible compared to the actual execution time of the data mining algorithm.
# Table of Contents

1. Introduction .................................................................................................................. 1
2. Preliminaries .................................................................................................................. 3
   2.1 Data mining Techniques .......................................................................................... 3
       2.1.1 Classification/Regression .............................................................................. 4
       2.1.2 Clustering ..................................................................................................... 4
       2.1.3 Association Rule Discovery (ARD) ............................................................... 4
   2.2 Data mining Algorithms ............................................................................................ 5
       2.2.1 Decision Tree ............................................................................................... 5
       2.2.2 k-Means ....................................................................................................... 6
       2.2.3 FP-Growth .................................................................................................... 7
   2.3 Execution Engines .................................................................................................... 8
       2.3.1 Spark ........................................................................................................... 8
       2.3.2 Mahout ........................................................................................................ 9
       2.3.3 WEKA ....................................................................................................... 9
3. Related Work .................................................................................................................. 10
   3.1 Execution Engines and Data mining Algorithms Benchmarks ............................... 10
   3.2 Single Platform for Multiple Execution Engines .................................................. 12
   3.3 Executing Data mining Algorithms Without Programming .................................. 12
4. Concept .......................................................................................................................... 15
   4.1 Overview ............................................................................................................... 15
   4.2 Benchmark ............................................................................................................ 16
   4.3 Most Appropriate Execution Engine Selection ..................................................... 17
   4.4 Model Interface ...................................................................................................... 18
5. Implementation .............................................................................................................. 20
   5.1 Overview ............................................................................................................... 20
   5.2 Class Structure ...................................................................................................... 21
       5.2.1 The executionEngines Package .................................................................. 22
       5.2.2 The benchmark Package ............................................................................ 25
       5.2.3 The automaticSelection Package ............................................................... 27
       5.2.4 The dataTransformation Package ............................................................. 28
   5.3 Service Architecture ............................................................................................... 30
   5.4 Database ................................................................................................................ 32
   5.5 Libraries ................................................................................................................. 33
6. Evaluation ....................................................................................................................... 35
   6.1 Experimental Setup ................................................................................................. 35
   6.2 Benchmark ............................................................................................................. 38
6.3 Selection Process ................................................................................................. 43
  6.3.1 Distance Metrics............................................................................................. 44
  6.3.2 Runtime Overhead.......................................................................................... 48
  6.3.3 Quantifying Runtime Savings ........................................................................ 50
7. Conclusion............................................................................................................. 53
8. Outlook.................................................................................................................. 56
9. References.............................................................................................................. 58
10. Appendix ............................................................................................................. 62
  10.1 Appendix 1 .................................................................................................... 62
  10.2 Appendix 2 .................................................................................................... 64
1. Introduction

Nowadays, a huge amount of data is generated, and it is still growing steadily. That motivates several researches to be conducted in order to make use of these voluminous datasets. These researches aim to define processes for discovering valuable knowledge by analyzing this huge amount of data. One important process from this area is the Knowledge Discovery from Data (KDD) [2]. This process describes extracting useful knowledge from data. Data mining is considered as a single step in the KDD process that aims to detect knowledge and patterns out of the existing dataset.

The continues evolving of paradigms of the data processing influences the expert in the data field to develop more analytic platforms for performing mining tasks. These platforms were developed for several purposes, and they differ in characteristics, performance, and the data processing paradigm. Performing a data mining task needs prerequisites steps such as choosing an appropriate analytic platform, and refining the dataset to fit that analytic platform and the needed mining algorithm. There is no such a straightforward approach to perform such steps, and they need some efforts and expectations to be made. The plethora of the analytic platforms make the decision of choosing the most appropriate analytic platform harder for novice users.

Several works were performed for the sake of making the users’ decision easier for selecting the most appropriate analytic platforms. These works are principally benchmarks to evaluate the performance of these analytic platforms. These benchmarks are split into two types. The first type consists of benchmarks that evaluate and compare the performance of different analytic platforms [21][22][23][24][25]. Whereas, the second type consists of benchmarks of the performance of single analytic platform [26][27]. The previously mentioned works have several issues concerning their objectivity and the considered measures. For example, several previous works ignore the impact of varying the size of datasets on the performance and ignore other measurements such as consumption of resources or the quality of the mining algorithm. However, no such a work offers the user the automatic selection of the analytic platform depending on the user demand.

The approach proposed in this thesis aims to conduct benchmarks on different analytical platforms with different mining algorithms and using different datasets. That approach helps the user to select the most appropriate analytical platform according to user-specified criteria, such as the performance. The implementation of the proposed approach should be extensible and loosely coupled in order to add different execution engines and different mining algorithms for benchmarking.

Various analytical platforms are used for different purposes and use cases since they differ in execution performance. Hence, it is essential to have a concept that aggregates several execution engines in a single model to give the user the flexibility to choose between these engines according to the use case. Several works are performed in this area [28]. However, the user still needs to specify the analytic platform explicitly. Hence, there is a need for a concept that aggregates several execution engines in a single platform and furthermore helps the user to select the most appropriate execution engine automatically.

Many users from different domains require to execute data mining algorithms. However, executing data mining algorithms requires a certain level of programming skills. That motivates several works to be conducted in order to fill the gap between the execution of data mining algorithms and novice users without programming experiences [29][30][31]. Providing such solutions allows for efficient execution of mining algorithms for novice users who do not have
a profound programming background. Hence, the goal is to enable the user to concentrate more on designing the steps of processing the data and defining which mining algorithm to execute rather than concentrating on the programming level.

The goal of this thesis is to solve all of the previously mentioned issues when executing data mining algorithms. That can be achieved by designing an approach that solves the problem of selecting the most appropriate analytic platform automatically for executing the needed data mining algorithm. Depending on a previously conducted benchmark, the selection process compares the performance of different analytic platform and selects the one with the best performance according to the user-defined criterion. These criteria can be runtime, RAM usage, CPU usage, and the quality of the data mining algorithm. This approach provides an abstraction for several analytic platforms. Hence, it allows the user to execute mining algorithms on different analytic platforms according to the user need. Finally, this approach allows novice users without a profound programming background to execute the mining algorithms easily.

This thesis is structured into the following sections: In Section 2, the preliminaries of the used algorithms and the used technologies in this thesis are discussed. In Section 3, the related works that address the issues which are covered in this thesis are discussed. In Section 4, the concept of the proposed approach is discussed. In Section 5, the prototypical implementation of the introduced concept of the thesis is discussed. In Section 6, the evaluation of the proposed approach is performed via a conducted experiment using various dataset. In Section 7, the result of the thesis is the concluded. Finally, in Section 8, the outlook and future work are discussed.
2. Preliminaries

This chapter covers basic concepts about the used algorithms and used technologies in this thesis. In Section 2.1, an overview of the most commonly used data mining techniques along with some examples of each technique will be discussed. Whereas, Section 2.2 discusses some typical data mining algorithms and how they are working. Finally, Section 2.3 the most commonly used execution engines for data mining tasks will be covered.

2.1 Data mining Techniques

The recent smart technologies that have been used widely and frequently nowadays created the so-called “information era” [1]. That fact causes a huge amount of data to be generated and to be growing steadily in recent years. For instance, the World Wide Web nowadays is carrying large daily updated data from various fields such as business, medicine, education, society, and so forth.

In order to make use of these voluminous datasets, a methodology is required, which enables analysts to extract knowledge from data. An example of such a methodology is called Knowledge Discovery from Data (KDD), which was described by U. Fayyad et al. in [2].

![Fig. 1: KDD Process](image)

KDD is a process that contains several steps such as data selection, data pre-processing, data transformation, data mining, pattern evaluation, and knowledge presentation (see Fig. 1). Hence, data mining is considered as a single step in the KDD process that aims to detect knowledge and patterns out of the existing dataset.

Data mining relies on different techniques that are divided into two main categories: supervised learning and unsupervised learning. Supervised learning techniques refer to those techniques that work on datasets with already given correct outputs which are called labels. Labels are merely the categories that items in a dataset should belong to. For that reason, these techniques are known as predictive techniques as well. Classification and regression are two data mining techniques that belong to supervised learning [3].

In unsupervised learning, the data has no labels. However, the unsupervised learning can be used to describe the internal structure of the given dataset. Hence, these techniques are said to be descriptive data mining techniques. Clustering and Association Rule Discovery (ARD) are two techniques of unsupervised learning.
2.1.1 Classification/Regression

Classification is a data mining technique that aims to build a predictive model out of an existing dataset [1]. This model is used later on for predicting classes. Generally, classification is divided into two phases which are training phase and testing phase. The training phase aims to build up a model out of the so-called training dataset. The training dataset consists of tuples, and each tuple is merely a set of attributes associated with a corresponding class label. These attributes might be numerical values or nominal values as well. The testing phase of classification is to use the generated model for predicting the class label of the submitted testing dataset.

Classification can be applied when the predictive class label originates from a categorical domain. However, the regression technique can be used in case of that the expected value is from a continuous “numerical” domain.

As a simple example of classification technique from the business field is a bank loan classifier. A model is built depending on a training data that is retrieved from the information of previous loan customers. This model can predict whether a future loan is “risky” or “safe” as class labels.

Another example from a sales store is a price predictor. A regression model is built from data of previous items that have some information as attributes and a price as an expected value. This model is used for predicting the price as a continues value.

2.1.2 Clustering

Clustering is one of the descriptive data mining techniques [1]. Clustering is one approach to describe a dataset by segmenting its items into different groups according to the similarity between each item within the chunk itself. This chunk is to be called a cluster. Assigning the items within clusters is accomplished in such a way that items from the same cluster are too similar, and items from different clusters are dissimilar.

Furthermore, each cluster must have at least one item. These clusters are generated without previous influencing factors like the case of class labels in classification. Hence, clustering helps to discover unknown groups of the existing dataset.

Clustering has been deployed widely for diverse applications in various fields such as image pattern recognition and web search. The dataset from the bank which used as an example in the previous section can also be an application of clustering but without any class labels. Clustering can be used here for segmenting the customers into different groups. These clusters can be beneficial to the bank for identifying new groups of customers and targeting them with new services in the future.

2.1.3 Association Rule Discovery (ARD)

ARD is as a descriptive data mining technique to discover possible patterns out of a dataset [1]. ARD produces rules that describe how items in a dataset are related to each other and how they are correlated. These rules are discovered according to user-provided parameters that describe the interesting occurrence of these rules from the user point of view. These parameters are called minimum support and minimum confidence.
Generating such rules requires two steps, i.e., (1) discovering the frequent itemsets from the dataset and (2) generating the association rules out of these frequent itemsets. The frequent itemsets are a group of items that exist in transactions frequently within the dataset.

Let $A \rightarrow B$ be a rule where $A$ and $B$ are items from a dataset. This rule indicates that if $A$ is existing in one transaction, then $B$ also exists in the same transaction too. The support is a percentage that determines the occurrence percentage of $A$ and $B$ being together in one transaction in the corresponding dataset. Whereas the confidence represents accuracy that is, when $A$ exists in a transaction, then $B$ will be in the same transaction too.

The most famous application of ARD is to define most frequently bought-together items in a specific store. Therefore, association rules are generated out of previous purchase transactions that contain these items.

### 2.2 Data mining Algorithms

In this section, three commonly used data mining algorithms [4] of the previously described data mining techniques are introduced. These algorithms are Decision Tree, k-Means, and FP-Growth. These algorithms are executed on a dataset $D$ that has $f$ features.

#### 2.2.1 Decision Tree

The decision tree is categorized as a classification data mining technique. The decision tree is a recursive process that aims to create a tree classifier out of a dataset. The tree is composed of a root node, internal nodes, and leaf nodes (see Fig. 2) [1][5].

Each node in the tree represents a test on an attribute from the training dataset. The branches that connect the nodes represent the consequence of the test done on each node. Finally, leaf nodes represent the class labels.

Fig. 2 is an example of a decision tree application. This tree represents the willingness of a person to buy a computer concerning some attributes like age, being a student, and the credit rating. For instance, if the person is young and a student then this person has the willingness of buying a new computer. However, if the person is young but not a student, then this person is not interested in buying a new computer.

The decision tree model is produced by a recursive algorithm. The algorithm receives the following parameters as input: Dataset $D$ along with corresponding class labels, attributes to be considered while splitting $f$, and a criterion, upon which the split is performed.
The splitting criterion indicates the way of splitting the dataset by determining two things. (1) Which attribute to be tested and (2) how this attribute will be split for a specific node N. The splitting criterion depends on the so-called attribute selection measure. There are three popular attribute selection measures [1]: information gain, gain ratio, and Gini index. The previously mentioned selection measures differ in the produced decision trees. In contrast to other selection measures, Gini index enforces the generated decision tree to be a binary tree. However, information gain and gain ratio allow for the generated decision tree to be a non-binary tree.

2.2.2 k-Means

The k-Means algorithm is one of the clustering algorithms that separates the items over clusters depending on the distance between the item and the centroid of the cluster.

The items in the dataset are represented in Euclidean space. Let $k$ be the number of clusters to be created, then $c_1, \ldots, c_k$ represent the centroid of each cluster. k-Means considers the centroid as the mean value of all items in the corresponding cluster according to the Euclidean distance. The less the Euclidean distance, the more similar are the item and the centroid.

The k-Means algorithm requires the dataset $D$, and the number of clusters to be created $k$ as input parameters. Initially, k-Means creates $k$ centroids and assigns items to these clusters according to the closest centroid. In a repeatable manner, k-Means reassigns items to clusters according to the Euclidian distance between each item and the centroid of the cluster. k-Means stops when no items change their associations anymore.

Fig. 3 shows the iterations while performing k-Means with 3 clusters. In Fig. 3 (a), the dashed lines show the initial clusters. However, the dotted lines in Fig. 3 (b) and Fig. 3 (c) show how the clusters change during the iterations. Finally, the solid lines represent the final clusters after the algorithm completion.
2.2.3 FP-Growth

Frequent Pattern Growth or shortly FP-Growth is an algorithm to discover association rules [1]. Generating the association rules from transactions in a dataset requires generating the frequent itemsets. In order to generate the frequent itemset, there is a naïve method used in the Apriori algorithm that depends on two iterative steps: (1) iterating over the database multiple times for generating the candidate frequent itemsets, and (2) excluding the itemsets that have less support than the minimum provided support. The disadvantage of this method is that an iteration through the dataset is required each time for calculating the support of each itemset. This method is inefficient and costly in case of massive databases. However, FP-Growth draws on another approach which generates FP-tree holding the information of the itemsets. Hence, generating the frequent itemset can be done by data mining the FP-tree rather than iterating the whole dataset repeatedly.

In order to generate the FP-tree, multiple steps are required. (1) iterating over the dataset for generating the frequent itemsets of size one along with their support. (2) generating a list that holds these itemsets and sorting them in descending order. (3) creating the root of the FP-tree and label it as “null”. (4) sorting the items in each transaction in the dataset according to the order of the previously generated list. (5) Adding the sorted item along with a counter 1 as a chain connected to the root of FP-Tree. The chain is created in such a way that the first item will be a parent for the second and so on. If the item in a transaction is already inserted, then the counter of that item will be increased.

<table>
<thead>
<tr>
<th>Transaction ID</th>
<th>Itemset</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>I2, I1, I5</td>
</tr>
<tr>
<td>2</td>
<td>I2, I1, I3, I5</td>
</tr>
<tr>
<td>3</td>
<td>I2, I1, I4</td>
</tr>
<tr>
<td>4</td>
<td>I2, I1, I3</td>
</tr>
<tr>
<td>5</td>
<td>I2, I3</td>
</tr>
<tr>
<td>6</td>
<td>I2, I3</td>
</tr>
<tr>
<td>7</td>
<td>I2, I4</td>
</tr>
<tr>
<td>8</td>
<td>I1, I3</td>
</tr>
<tr>
<td>9</td>
<td>I1, I3</td>
</tr>
</tbody>
</table>

Table 1: Dataset Contains Transactions with Items I1, I2, I3, I4, I5 (based on [1])
Let the dataset in Table 1 and the FP-Tree of this dataset in Fig. 5. A list that holds the frequent itemset of size one is created and sorted as follows: $L= \{I2:7, I1:6, I3:6, I4:2, I5:2\}$. The frequent itemsets can be discovered when assuming two as minimum support. First, we start from the leaf nodes. Let consider I5 in the chain “I5, I3, I1, I2” (see Fig. 4, the dashed line rectangle). I5 here has one as support which does not comply with the assumed minimum support so it can be excluded. Then there are I3, I1, and I2 which have acceptable supports that equal to or higher than 2. Hence the itemsets [I3, I1, I2], [I1, I2], [I2] are frequent itemsets. Note that there are also other frequent itemsets in the FP-tree depicted in Fig. 4.

![Fig. 4: FP-Tree for Dataset in Table 1 (based on [1])]  

### 2.3 Execution Engines

Several existing analytic platforms exist which can execute data mining algorithms. Three of these analytic platforms are covered in this thesis. These platforms are Spark, Mahout, and WEKA. These analytic platforms have been chosen according to their popularity and the compatibility of the used technologies. These analytic platforms are mentioned in this thesis as “execution engines”.

#### 2.3.1 Spark

Apache Spark is an open-source platform for distributed analysis of huge dataset [6][7]. Spark has been developed as a part of a research project at UC Berkeley.

The programming model of Spark depends on a data-sharing abstraction which is called Resilient Distributed Dataset (RDD). Spark allows RDD to be stored in memory. Hence, that enhances the performance of algorithms that require several iterations through the data. Spark also has a high level of fault tolerance since RDD can rebuild itself in case of fault by using lineage method. This method depends on tracking the transformation operations that are done on RDDs. Hence, these operations can be executed again on the base data to rebuild the lost partitions.

Spark offers a collection of functions in an API that is available in Java, Scala, R, and Python. Hence, Spark can be easily integrated into a wide range of applications that are written in those languages. By using these functions, the user will be able to interact with and generate RDDs.
Spark presents a library that holds several machine learning algorithms, which is called MLlib [8]. MLlib consists of scalable and efficient implementations of several machine learning algorithms covering different techniques such as classification, regression, clustering, ARD, collaborative filtering, and dimensionality reduction. MLlib provides APIs for Java, Scala, R, and Python as well.

2.3.2 Mahout

Apache Mahout is an analytic platform that offers machine learning algorithms on top of Hadoop [9]. Mahout has been developed as a sub-project of Apache’s Lucence project in 2008. Mahout supports a wide range of machine learning algorithms the covers distributed linear algebra, regression, classification, clustering, association rule discovery, and recommendation engines [10]. Since Mahout works on top of Hadoop, it depends on the MapReduce programming model. Mahout is implemented in Java, and it provides a Java API for calling its implemented algorithms.

The implemented algorithms in Mahout can be classified into two main categories; sequential and parallel algorithms [11]. The sequential algorithms are those that executed without Hadoop at all, e.g., logistic regression, and user-based recommendation. The parallel algorithms are the algorithms that are executed by using Hadoop, e.g., k-Means, and Random Forest.

2.3.3 WEKA

WEKA is a machine learning tool that has been introduced in 1993 at the University of Waikato, New Zealand. WEKA stands for “Waikato Environment for Knowledge Analysis”[12][13][14]. WEKA aims to provide a tool for machine learning that targets novice users in the machine learning domain.

WEKA is written in Java, and it offers a simple graphical user interface that allows inexpert users to select and execute an extensive range of machine learning algorithms. These algorithms cover classification, regression, clustering, ARD, and attribute selection.

In contrast to Spark and Mahout, WEKA, stand-alone, does not support distributed execution. For that reason, some works are aiming to give WEKA the ability of parallel execution and supporting HDFS too. DistributedWekaHadoop¹ is a package that has been developed to support loading and to save HDFS files, and it provides wrappers for executing Hadoop jobs and tasks. Another package is called DistributedWekaSpark² which allows WEKA to be executed on top of Spark to make use of Spark scalability and in-memory execution. However, these packages support only a few data mining algorithms, and they have no extensive documentation. Hence, in this thesis, only pure WEKA has been added to the developed tool.

WEKA requires the data format Attribute-Relation File Format (ARFF) as an input data format. However, WEKA provides functions to convert and read data from CSV files as well [15]. The ARFF is merely a text file with ASCII encoding. This file lists the metadata in the first part of the file. These metadata are the attributes along with their names and types. The data itself will be listed in the second part of the file.

¹ http://weka.sourceforge.net/packageMetaData/distributedWekaHadoop/index.html
3. Related Work

Nowadays, several execution engines exist for executing data mining algorithms. These execution engines differ by their characteristics, purposes, and performance. The plethora of these execution engines escalates the difficulty of selecting the most appropriate execution engine according to specific user-defined criteria. However, there are several related works which aim to help the user in selecting the most appropriate execution engine. These works are mainly benchmarks for evaluating the performance of these execution engines. However, these benchmarks have several issues regarding their objectivity and the considered measures.

As mentioned before, the existing execution engines have different purposes and characteristics. Hence, there is a need for a concept that aggregates several execution engines in a single platform. Consequently, allow the user to execute data mining algorithms on different execution engines using this single platform efficiently.

There is also another issue to be addressed in the area of data mining algorithms execution. This issue is that executing data mining algorithms requires a certain level of programming skills in order to fine-tune the algorithms with the specific parameters. However, several works aim to help users without programming background to execute data mining algorithms efficiently.

In Section 3.1, benchmarks on different execution engines and different data mining algorithms are discussed. Section 3.2 covers an existing model for executing data mining algorithms on different execution engines. Section 3.3 discusses related works that allow the user without programming skills to execute data mining algorithms efficiently.

3.1 Execution Engines and Data mining Algorithms Benchmarks

This section covers an overview of two types of existing benchmarks in the area of execution engines and data mining algorithms. These benchmarks aim to help users to choose between different execution engines and different data mining algorithms according to their performance.

These benchmarks can be categorized into two types. (1) Benchmarks that are conducted using a single execution engine to evaluate the performance of different data mining algorithms. (2) Benchmarks that are performed on single execution engine or different execution engines to evaluate the performance of these engines.

Data mining algorithms are in a continues development for the sake of enhancing their performance. That allows new implementations of different data mining algorithms to be released from time to time. The variety of data mining algorithms makes the user’s decision more difficult regarding which data mining algorithm to choose. For that purpose, benchmarks of different data mining algorithms have been released [16][17][18][19][20]. R. Narayanan et al. introduces a benchmark suite that consists of fifteen different data mining algorithms from different data mining techniques [16]. These techniques are classification, clustering, ARD, structure learning and optimization. Data mining Algorithms in MineBench has been implemented in C and C++ without using any other data mining execution engines. Later on, several works use MineBench suit to examine the performance of these data mining algorithms regarding different measures such as the consumption of resources [17] and to examine the scalability performance.
of these algorithms too [18][19]. However, a previous work solely focuses on single data mining technique.

O. Altun et al. focus on benchmarking the clustering data mining technique using data mining algorithms that are written in C. The benchmark represented in the previous work focuses only on runtime measure. The previously mentioned works consider data mining algorithms that are implemented using several programming languages without using any of the existing data mining execution engines. Consequently, the result of the benchmark in these works heavily depends on the quality of the concrete implementations of these data mining algorithms. Moreover, the size of the dataset is usually fixed within each data mining technique in the previously mentioned works. Hence, the impact of changing the size of the dataset is not examined for each data mining algorithm.

Nowadays, data mining applications are expanding and are used for different purposes. Hence, several execution engines for data mining algorithms are released frequently to fit the different purposes. However, these execution engines can execute several data mining algorithms, but they differ in their performance and their characteristics. That motivates several works in the area of benchmarks to evaluate the performance of these execution engines [21][22][23][24] [25][26][27]. These benchmarks are categorized into two types. The first type is the benchmark that contains several execution engines to compare their performance [21][22][23][24][25]. Previous works conducted benchmarks for testing the suitability of these execution engines for specific real-world scenarios. For example, M. Fritz et al. introduces a benchmark of big data systems to test their effectiveness and their performance for energy procurement sector [21].

In contrast to the previously mentioned work, other works focus on benchmarking different execution engines generally without a specific use case scenario. S. Shi et al. present a benchmark for deep learning platforms that covered three commonly used neural networks [22]. The benchmark has been conducted using several hardware platforms to examine the influence of changing the hardware platforms on the performance. Another benchmark is represented by the work of S. Chintapalli et al. which focus on the performance of stream processing on different execution engines, i.e., Strom, Flink, and Speak [23]. L. Wang et al. represent a benchmark suite of big data systems which covers several application scenarios and diverse datasets [24]. As the last example of this kind of benchmarks is the work done by S. Pafka [25]. The previous work focuses on benchmarking several execution engines for classification regarding different perspectives such as scalability, speed, and accuracy.

The second type of execution engines’ benchmark is the benchmark that discusses the performance of the single execution engine [26][27]. For example, S. Huang et al. introduce a benchmark suite for Hadoop regarding several performance measures such as speed, throughput, HDFS bandwidth, and the consumption of resources [26]. Whereas, M. Li et al. represent a benchmark suite for Spark execution engine using different data mining algorithms.

The previously mentioned works aim to help the user to choose between different execution engine and different data mining algorithms. However, there are several issues regarding these works. Firstly, the previous works mainly ignore the impact of varying the dataset size on the performance entirely. Secondly, they depend mainly on the runtime measurement as a key-performance with almost ignoring other measurements such as resources consumption like
RAM and CPU or the quality of the data mining algorithm execution. Thirdly, not all existing execution engines nor all existing data mining algorithms have been covered in a benchmark by previous work. Finally, no such a work offers the user the automatic selection of the appropriate execution engine depending on the user demand.

### 3.2 Single Platform for Multiple Execution Engines

It has been discussed in the previous section that several execution engines exist. However, various execution engines are used for different purposes and use cases since they differ in execution performance. Hence, it is essential to have a concept that aggregates several execution engines in a single model to give the user the flexibility to choose between these engines according to the use case.

Apache Beam\(^3\) is an open-source programming model for data processing. Apache Beam depends on the work of Akidau et al. [28]. This work introduces a data flow model to be used for analyzing huge, streaming, and unordered data. That model relies on the pipeline concept where pipelines represent the data analyzing processes. The pipelines are data processing steps that are connected as a series. Afterwards, the implementations of these pipelines are defined separately.

Similarly, Apache Beam uses the data pipelines concept to define data processing steps. Apache Beam offers Software Development Kits (SDKs) in different languages such as Java, Python and GO. Hence, it can support wide-ranged applications that are written in different languages. By using these SDKs, the user can define data pipelines. Afterwards, the user can execute these pipelines on different execution engines which are supported by Apache Beam itself. Currently, six different execution engines are supported: Apache Apex, Apache Flink, Apache Gearpump, Apache Samza, Apache Spark, and Google Cloud Dataflow. However, the user must provide the execution engine that will execute these pipelines explicitly.

Apache Beam offers a solution for defining data pipelines then, executing them on different execution engines. However, Apache Beam does not support the automatic selection of the execution engine depending on the performance of each engine.

### 3.3 Executing Data mining Algorithms Without Programming

Data mining, according to its importance, is a demand for different kind of users from different domains. However, executing data mining algorithms requires a certain level of programming skills. Hence, several tools are developed in order to fill the gap between the execution of data mining algorithms and users without programming experiences.

Referring to Section 2.3.3, WEKA offers a GUI so that users without programming experience can efficiently execute data mining algorithms. This GUI is called WEKA Knowledge Explorer [29]. By using WEKA Knowledge Explorer, the user can navigate and execute all WEKA supported data mining algorithms. Moreover, WEKA Knowledge Explorer offers a visualization for the dataset. This visualization facilitates a better understanding of the values and the structure of the corresponding dataset.

There are several works in this area that uses pipes and filters architecture to fine-tune parameters of the data mining algorithms graphically. These works rely on defining the data mining process as blocks which represent filters. These filters are connected by pipes which

---

\(^3\) [https://beam.apache.org](https://beam.apache.org)
represent the data flow between these filters. Examples of these works are: RapidMiner⁴ [30], KNIME⁵ [31], and Microsoft Azure Machine Learning Studio⁶. The previously mentioned works allow the user to execute data mining algorithms graphically by using visual elements (see Fig. 5, Fig. 6, Fig. 7). The user can easily define data analysis pipelines and execute different data mining algorithms by adding the corresponding elements to the canvas.

Fig. 5: Snapshot from RapidMiner While Fine-tuning the Parameters of k-Means

Fig. 6: Snapshot from KNIME While Fine-tuning the Parameters of k-Means

⁴ https://rapidminer.com
⁵ https://www.knime.com
⁶ https://studio.azureml.net
Fig. 7: Snapshot from Microsoft Azure Machine Learning Studio While Fine-tuning the Parameters of k-Means

It has been discussed before that executing data mining algorithms requires a certain level of programming skills in order to fine-tune the algorithms with the specific parameters. However, such interfaces allow novice users, who do not have a strong programming background, for fine-tuning the parameters of data mining algorithms easily. Hence, providing such methods for fine-tuning the parameters of the data mining algorithms allows the user to concentrate more on designing the steps of processing the data and defining data mining processes to execute rather than concentrating on the programming level.
4. Concept

The plethora of execution engines, as discussed in the previous chapter, makes the selection of the most appropriate execution engine according to the user criteria more difficult. This thesis aims to design a novel way to solve the problem of selecting the most appropriate execution engine automatically for executing a data mining algorithm. The size of the dataset and the needed data mining algorithm play a vital role in selecting the most appropriate execution engine based on user demand. The selection process depends on the result of the previously conducted benchmark. The benchmark captures only general characteristics of the dataset. That means, no private nor corporate information about the data is stored in the benchmark which allows for several users and several companies to merge their benchmark databases. That perhaps allows for achieving a better selection of the execution engine since the selection process depends on the benchmark database.

The introduced approach in this thesis aggregates several execution engines with various data mining algorithms which offers a single, abstract interface to several execution engines. Hence, it allows the user to execute data mining algorithms on different execution engines according to the user need. Moreover, this approach allows users without programming background to execute the data mining algorithm easily.

Section 4.1 covers an overview of the introduced approach in this thesis. Section 4.2 discusses the reliable, objective benchmark which is used in that approach for the selection process. Section 4.3 covers the process of selecting the most appropriate execution engine and discusses the factors that are used in the selection process. Finally, Section 4.4 discusses the introduced interface in this thesis.

4.1 Overview

The selection of the most appropriate execution engine depends on a hypothesis which is executing a data mining algorithm on similar datasets, using the same execution engine, yields comparable performance. In order to make this comparison possible, there is a need for a measurement system. This measurement system produces a benchmark with the performance measures of the previously executed data mining algorithms using different datasets.

M. Bourne, et al. represented issues regarding designing and implementing measurement systems for manufacturing companies [32]. These measurement systems are used for capturing and tracking the performance of the company. Afterwards, these measures are used for enhancing the performance of the company by a reflection step. The reflection step is done by taking decisions that affect these measures in a positive way.

According to that work, there are three steps for designing and implementing such a measurement system. These steps are: (1) defining what measurements to measure, (2) capturing these measures, and (3) making use of these measures to enhance the whole process.

Fig. 8 shows the workflow of designing and implementing such a measurement system. Firstly, the workflow starts by identifying what to measure and how. Secondly, collecting these measures. Thirdly, storing and reviewing these measures, Finally, reflecting these measures to achieve the key objectives.
Identifying the measures depends on the key objectives of the company. In another word, finding the performance measures hinge on what the company aims to enhance. The following are some examples of these measures: number of lost orders, cash generation, time spent on research, etc. [32].

![Workflow of Designing a Measurement System for a Company](based on [32])

The proposed approach in this thesis for selecting the most appropriate execution engine has a similar concept to that represented in the previous work. The concept in this thesis is to collect the performance measures of the execution engines via the benchmark. Afterwards, reflecting these measures in the selection process of the most appropriate execution engine. Hence, the same steps that are in the previous paper are required.

In order to achieve the selection process of the most appropriate execution engine, there is a need for a benchmark. This benchmark must cover different data mining algorithms executed on different datasets and using different execution engines. The process of selecting the most appropriate execution engine has four steps. The first step is to determine the key objectives of executing data mining algorithms and to identify what to measures depending on these objectives. The second step is to build a reliable benchmark that contains the performance measures of previously executed data mining algorithms using different executing engines. The third step is to review the result of the benchmark and to perform the selection process depending on the result of that benchmark. The fourth step is to reflect that result by executing the required data mining algorithm on the selected execution engine.

Selecting the most appropriate execution engine represents the reflection of the benchmark result to enhance the execution performance. However, enhancing the execution performance does not mean enhancing the implementation of the data mining algorithms but choosing the execution engine that has better performance for that specific data mining algorithm with a similar dataset.

4.2 Benchmark

Refereeing to the work that is discussed in Section 4.1, the first step of designing a performance measuring system is to identify what to measure. However, identifying the measures depends heavily on the key objectives. In the area of data mining, there are several objectives to be considered. The first objective is the fast execution of the data mining algorithms. The runtime of executing a data mining algorithm on a dataset is affected by several factors. These factors are the dataset itself, the data mining algorithm, the parameters of this data mining algorithm, and the implementation of this data mining algorithm.

The second objective is to minimize the consumption of resources used for executing data mining algorithms. The resources that are in scope, in this thesis, are CPU and RAM. Since these resources considered as costly hardware, there is a need to minimize their consumption
when executing data mining algorithms. The last objective is to achieve a high-quality result of the data mining algorithms. The quality of the data mining algorithms is affected by several factors. These factors are the concrete implementation of the data mining algorithm and the characteristic of the dataset such as the size of this dataset.

Fig. 9 shows the designed measures depending on the previously mentioned key objectives. The first measure is the runtime of executing the data mining algorithm by the corresponding engine. The second and the third measure is the consumption of resources which are represented by CPU and RAM usage.

![Fig. 9: Designing Measures Depending on Key Objectives](image)

The last measure is the quality of the data mining algorithm. However, the quality measure differs according to the data mining technique. In the case of the classification technique, the quality is represented by the accuracy which is the percentage of the correctly predicted values by the classification model [1]. The quality measure of the clustering technique is represented by the sum of squared errors between all items in the cluster and the centroid [1]. However, there are no quality measures for ARD technique since the quality measures are defined in advance as parameters for the data mining algorithms, i.e., minimum support and minimum confidence.

4.3 Most Appropriate Execution Engine Selection

It has been mentioned before that the selection of the most appropriate execution engine depends on a hypothesis which is executing a data mining algorithm on similar datasets, using the same execution engine, yields comparable performance.

The selection process depends on user-defined criteria which are represented by the measures that listed in Section 4.2. The benchmark contains measurements of previously executed data mining algorithms on different datasets. Hence, the selection process can rely on this benchmark for finding the most appropriate execution engine depending on the performance of the similar datasets. The similarity between datasets is represented by a distance metric which considers data characteristics. For the purpose of this work, the following characteristics are of interest: (1) the size of the dataset in bytes, (2) the number of features, i.e., columns that the dataset has, and (3) the number of entities in the dataset.
Hence, four steps are required for selecting the most appropriate execution engine. These steps are depicted in Fig. 10. Firstly, submitting a dataset with a corresponding data mining algorithm and user-defined criteria. Secondly, retrieving all benchmark results that are related to the same required data mining algorithm. Thirdly, computing the distance metrics between the submitted dataset and all datasets in the previously retrieved benchmark data. Fourthly, determining the minimum distance metric which leads to the most similar dataset in the benchmark. Finally, selecting the most appropriate execution engine by considering the performance metrics of the most similar dataset. Depending on user-defined criteria, the selected execution engine is that engine which has the most appropriate performance on that similar dataset.

![Diagram of the selection process]

**Fig. 10: Process of Selecting the Most Appropriate Execution Engine**

### 4.4 Model Interface

As discussed in Section 3.3, it is essential to provide an approach for execution engines which allows the users without programming background to execute data mining algorithms efficiently. For that reason, this thesis introduces a model, which is depicted in Fig. 11, for abstracting these execution engines. Hence, the user can efficiently execute data mining algorithms by using this model with two modes.

The first mode is the manual selection of the execution engine where the user can choose a specific execution engine for executing data mining algorithms. In this mode, four inputs are required for the execution: (1) the dataset, (2) the name of the execution engine, (3) the name of the data mining algorithm, and (4) the required parameters for that data mining algorithm. By using the first mode, the user will be able to execute data mining algorithms by passing the previously mentioned parameters without the need for programming. Hence, novice users will be able to concentrate more on the application and the result of the data mining algorithms rather than spending time and effort on the programming part of the data mining algorithms.
The second mode is the automatic selection of the most appropriate execution engine depending on the user-defined criteria. The required parameters are: (1) the dataset, (2) the name of the data mining algorithm, (3) the required parameters for that data mining algorithm, and (4) the user-defined criteria (runtime, CPU usage, RAM usage, or quality). The user can choose several criteria to be considered together for executing one single execution engine. However, the chosen criteria will be considered according to their importance from a user point of view. For example, when the user chooses runtime and RAM usage respectively as defined criteria, then the selected execution engine will be the one with the best runtime as the highest priority. However, if several execution engines have almost the same runtime, then the execution engine with the lower RAM usage will be chosen.

Fig. 11: Concept for Automated and Manual Selection of Execution Engines
5. Implementation

This thesis introduces an approach to solve the problem of selecting the most appropriate execution engine depending on user-defined criteria for executing data mining algorithms. The selection process depends on the result of a previously conducted benchmark. This approach aggregates several execution engines with various data mining algorithms to offer a single, abstract interface to several execution engines. Hence, this approach allows the user to execute data mining algorithms on different execution engines according to the user need. Moreover, this approach allows users without programming background to execute the data mining algorithm efficiently.

The prototypical implementation of the proposed approach in this thesis has four main modules. These modules are the execution engines module, the benchmarking module, the automatic selection module, and the data transformation module. The execution engines module consists of several execution engines and several implemented data mining algorithms.

This section discusses the implementation details of the prototypical implementation. In Section 5.1, an overview of the prototypical implementation is covered. In Section 5.2, the class structure of the existing packages is discussed. The prototypical implementation in this thesis has a service-oriented architecture. In Section 5.3, the service architecture and addresses of endpoints that are implemented in this thesis are discussed. In Section 5.4 the database that is used for the prototypical implementation along with the entity relationship diagram of that database are covered. Finally, in Section 5.5, the used libraries for the prototypical implementation are listed.

5.1 Overview

In Fig. 12, the architecture of the prototypical implementation of the proposed approach in this thesis is depicted. The prototypical implementation has a loosely coupled architecture and consists of several modules. The loose coupling of this prototypical implementation allows each module to be used separately and managed individually. These modules have been divided according to the functionality of each module. These modules are: automatic selection module, benchmarking module, database module, data transformation module, and execution engines modules.

Each module represents a separated service that can be accessed via a REST API. The benchmarking service is the responsible service for conducting benchmarks and storing the result in the database. The benchmarking service is connected to all execution engines. By calling the benchmarking service, it runs a benchmark on the corresponding execution engines. That benchmark contains all available data mining algorithms within the corresponding execution engine. The data mining algorithms are executed several times on different datasets in order to achieve a solid result. Afterwards, the result of that benchmark will be stored on the database.

The database contains the benchmark results and the services’ URLs of all available data mining algorithms within each execution engine. The stored service’s URLs are used afterwards by the benchmarking service. The database itself is implemented as a service too. That allows the user to retrieve benchmark results and all available data mining algorithms efficiently by calling the database service.
The automatic selection service depends on the stored benchmark results to execute the required data mining algorithm on the most appropriate execution engine according to the user-defined criteria. The user-defined criterion can be runtime, RAM usage, CPU usage, and quality. Hence, the selected execution engine depends heavily on the user-defined criterion. For example, if the user selected the runtime criterion, then the selected execution engine will be the one with the lowest execution runtime among other engines.

![Diagram of prototypical implementation architecture](image)

**Fig. 12: Prototypical Implementation Architecture**

The data transformation service is responsible for the data transformation steps. These steps are required for transforming the data into the corresponding format in order to be accepted as a valid input by the corresponding execution engine. The automatic selection service uses this service in order to prepare the data for the selected execution engine. However, this service can be called by the user directly in order to transform the data to the valid format that can be used with the user preferred execution engine.

Each execution engine in this prototypical implementation is represented as a separated service. A specific execution engine service can be called by the end user directly for executing data mining algorithms. In that case, the user takes advantage of executing data mining algorithms by sending a request to the corresponding service along with the required parameters without the need for extensive programming for fine-tuning the parameters of the data mining algorithm.

The prototypical implementation is developed in Java programming language. The Java implementation allows for adding more execution engines to the prototypical implementation since most of the execution engines have a Java API. All input datasets for the prototypical implementation are stored in HDFS and can be accessed by each of the previously mentioned services.

### 5.2 Class Structure

The following subsections discuss the concrete implementation of each of the modules that are listed in Section 5.1. Each of these modules is implemented as a separated Java package. More precisely, in Section 5.2.1, the functionality of the `executionEngines` package and its sub-packages are discussed. In Section 5.2.2, the `benchmark` package that is responsible for conducting the benchmark is discussed. In Section 5.2.3, the `automaticSelection` package that is responsible for executing data mining tasks on the most appropriate execution engine is
discussed. In Section 5.2.4, the dataTransformation package which is responsible for transforming the data into the required data format for each execution engine is discussed.

5.2.1 The executionEngines Package

Each execution engine in the prototypical implementation is represented by a separated package. For the sake of well-organized packages, all execution engines packages are existing in one single package. This package is called the execution engines package which is depicted Fig. 13.
The `executionEngines` package contains five interfaces and a single class which is called `MiningTask` in addition to the packages that represent all execution engines. These interfaces and the single class aim to unify the implementation of each execution engine. More specifically, there is one interface for each data mining technique. This interface addresses all data mining algorithms in the corresponding data mining technique as function calls. Afterwards, this interface will be implemented with the concrete implementation of the data mining algorithms in each execution engine separately.

All data mining tasks from different execution engines have common attributes. These attributes are runtime, CPU usage, RAM usage, and quality. Hence, `MiningTaskInterface` describes the characteristics’ properties of the execution, e.g., getters and setters of these characteristics’ properties. All data mining techniques interfaces extend `MiningTaskInterface`. Moreover, the implementation of the functions of `MiningTaskInterface` is independent of any implementation of a particular execution engine. Hence, there is a single class for all execution engines that implements the `MiningTaskInterface` which is `MiningTask` class.

![Fig. 14: Spark Execution Engines Package](image)

---

**SparkExecutionEngineController**

- `SparkExecutionEngineController()`
- `kmeans(Map<String, Object>): ResponseEntity<String>`
- `gaussianMixture(Map<String, Object>): ResponseEntity<String>`
- `hierarchicalClusterer(Map<String, Object>): ResponseEntity<String>`
- `latentDirichletAllocation(Map<String, Object>): ResponseEntity<String>`
- `fpGrowth(Map<String, Object>): ResponseEntity<String>`
- `kmeans(Map<String, Object>): ResponseEntity<String>`
- `latentDirichletAllocation(Map<String, Object>): ResponseEntity<String>`
- `hierarchicalClusterer(Map<String, Object>): ResponseEntity<String>`
- `gaussianMixture(Map<String, Object>): ResponseEntity<String>`
- `linearRegression(Map<String, Object>): ResponseEntity<String>`

**Cluster Mining Task Interface**

- `output: String`
  + `kmeans(String, Integer, Integer): String`
  + `gaussianMixture(String, Integer, Integer): String`
  + `hierarchicalClusterer(String, Integer, Integer): String`
  + `latentDirichletAllocation(String, Integer, Integer): String`

**Classification Mining Task Interface**

- `output: String`
  + `decisionTree(String): String`
  + `randomForest(String): String`
  + `supportVectorMachine(String): String`
  + `naiveBayes(String): String`

**ARD Mining Task Interface**

- `output: String`
  + `ARDMiningTask()`
  + `fpGrowth(String, String, Double, Double): String`

**Regression Mining Task Interface**

- `output: String`
  + `linearRegression(String): String`
As mentioned before, each execution engine is implemented in a separate package. Fig. 14 shows the spark package which is the Spark execution engine as an example of this kind of packages. The spark package contains classes that represent each of the investigated data mining technique. Each data mining technique class implements the corresponding data mining technique interface from executionEngine package. Moreover, these classes extend MiningTask class from the executionEngine package in order to have access to the common functions of all data mining task objects. The SparkExecutionEngineController class is responsible for implementing the endpoints and handling HTTP requests to the Spark execution engine. Each function in SparkExecutionEngineController is responsible for handling a specific data mining algorithm by performing several steps. For example, kmeans() performs the following steps after calling its endpoint: (1) receiving the k-Means algorithm requests, (2) extracting the passed algorithm parameters from the request, (3) calling the corresponding k-Means algorithm with the received parameters, and (4) returning the response along with the result of the executed data mining algorithm.

The prototypical implementation contains three execution engines which are: Spark, Mahout, WEKA. Each execution engine contains several data mining algorithms. Table 2 shows the currently supported data mining algorithms in the prototypical implementation for each execution engine.

<table>
<thead>
<tr>
<th>Data mining Technique</th>
<th>Data mining Algorithm</th>
<th>Spark</th>
<th>Mahout</th>
<th>WEKA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classification</td>
<td>Decision Tree</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Random Forest</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>SVM</td>
<td>✓</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Naïve Bayes</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
</tr>
<tr>
<td>Regression</td>
<td>Linear Regression</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>k-Means</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Clustering</td>
<td>Gaussian Mixture</td>
<td>✓</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Hierarchical</td>
<td>✓</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Latent Dirichlet Allocation</td>
<td>✓</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>ARD</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Table 2: The Implemented Data Mining Algorithms in the Prototypical Implementation
The (√) indicates that the data mining algorithm is already implemented in the prototypical implementation. Whereas, the (−) sign indicates that this algorithm is not supported by this execution engine at all. Finally, the (X) indicates that the algorithm is supported by that execution engine theoretically, but it has not implemented in the prototypical implementation. That refers to that there is no extensive documentation existing for these algorithms.

Executing a data mining algorithm on a specific execution engine requires an HTTP request. The request is sent to the endpoint of the corresponding execution engine. The HTTP request contains the following parameters: the name of the data mining algorithm, and the required parameters for that data mining algorithm. After executing the data mining algorithm on the corresponding engine, a response for the previous request is received. This response contains the following: (1) the URI of the output of the data mining algorithm, (2) the runtime, (3) RAM usage, (4) CPU usage, and (5) the quality of the executed data mining algorithm. Fig. 15 shows an example of request and response for calling the k-Means algorithm on Spark.

![Fig. 15: Request/Response Example for Executing the k-Means Algorithm on Spark](image)

### 5.2.2 The benchmark Package

The benchmark package is responsible for running benchmarks and storing the results of these benchmarks on the database. Referring to Section 4.2, the benchmark module aims to capture and store the interesting measures of executing data mining tasks on different execution engines. In order to implement this module, a Java package called benchmark package is developed. The benchmark package is depicted in Fig. 16.

![Fig. 16: Benchmark Package](image)
conducted benchmarks on different execution engines. In order to explain the mechanism of benchmark service, the following example is given. The following is a request that is sent to the benchmark endpoint: http://192.168.209.244:8080/benchmark/spark/5. This request initiates a benchmark operation for Spark execution engine. The 5 indicates the number of execution times for each data mining algorithm available in Spark execution engine. Fig. 17 shows the sequence diagram of executing the previously mentioned request.

Fig. 17: Benchmark Sequence Diagram

Once the previously mentioned request arrives at the BenchmarkController, several steps are performed. (1) Creating an object from Benchmark class with Spark as execution engine and five as number of iterations. (2) Calling retrieveAvailableServices() for retrieving all data mining algorithms in the Spark execution engine and assigning these values to serviceList. Each record from serviceList contains the endpoint’s address of the data mining algorithm along with datasets that will be used for executing the data mining algorithm. (3) Calling executeBenchmark() to start the benchmark process. (4) Sending a request to the Spark execution engine for each data mining algorithm and each corresponding dataset in the serviceList five times. (5) Calling calculateBenchmarkMetrics() to calculate the size of the dataset along with the median value of the measurements, i.e., runtime, RAM usage, CPU usage, and quality, for each of the five requests. (6) Calling storeBenchmarkResult() to store the benchmark metrics of the corresponding data mining algorithm in the database.
Returning the result of the conducted benchmark of all data mining algorithms that are available in Spark to the user as a response of the previously mentioned request.

Executing the data mining algorithms several times and storing the median values of the performance measurements helps for eliminating from the outlier values in the benchmark. However, the user can give only one iteration in the request if there is no need for multiple executions. A benchmark for a specific execution engine can be conducted by calling the benchmarking service along with the name of that engine. That mechanism allows for updating this benchmark efficiently in case of adding a new execution engine or even updating a pre-existing execution engine. Hence, the selection process will be depending on the most recent benchmark for the recent versions of added execution engines.

5.2.3 The automaticSelection Package

The automaticSelection package represents the analyzing service that can execute a data mining algorithm on a submitted dataset using the most appropriate execution engine. The benchmark database contains measurements of the previously executed data mining algorithms on different datasets. Hence, the selection process can rely on the performance measures of the similar datasets for finding the most appropriate execution engine. The similarity between dataset is represented by a distance metric. Several different distance metrics are evaluated in Section 6.3.1. Depending on the user-defined criterion, the selected execution engine will be the execution engine that has the best performance on that similar dataset. In order to achieve that concept, the automaticSelection package is developed which is depicted in Fig. 18.

![Fig. 18: The Automatic Selection Package](image)

The automaticSelection package contains two classes which are AutomaticSelection and AutomaticSelectionController. The automaticSelectionController class is responsible for implementing the endpoints and handling HTTP requests to the automatic selection service.
The AutomaticSelection class contains several functions for implementing the concept of selecting the most appropriate execution engine according to user-defined criteria. Let the user call the automatic selection service with a dataset, k-Means as a required algorithm, and runtime as a defined criterion. Fig. 19 shows the sequence diagram of the automatic selection service when handling the request above.

![Sequence diagram of automatic selection service](image)

**Fig. 19: Automatic Selection Sequence Diagram**

Once the previous request arrives at automaticSelectionController class, several steps are performed to execute the data mining task on the most appropriate execution. (1) Calling selectBestMatchedDataset() to retrieve all benchmark entries from the database with k-Means. (2) Selecting the most similar dataset from these entries depending on a minimum distance measure. (3) Calling selectMostAppropriateExecutionEngine() to select the most appropriate execution engine with the best performance on the most similar dataset according to the runtime. (4) Calling dataTransformation() in case that the input data is not accepted as valid input by the selected execution engine. In this example, the selected engine is WEKA which requires specific transformation steps. (5) Sending k-Means request to WEKA with the transformed input data. (6) Sending the result of the data mining task to the user as a response to the requested data mining task.

### 5.2.4 The dataTransformation Package

Each execution engine accepts the data as input only if the data is in a specific format. For that reason, the data transformation service is developed. This service is represented by a separate package which is depicted in Fig. 20. This package is responsible for transforming the input data into the required format that is accepted by each execution engine. However, the prototypical implementation in this thesis accepts only CSV files that are stored on HDFS. That allows for handling the input data and transforming the data to the required format efficiently.
It has been discussed in Section 2.3.3 that WEKA requires ARFF as input data format. Besides, WEKA does not accept data that is stored in HDFS. For that reason, several transformation steps are required before passing input data to WEKA. As an exceptional case, WEKA accepts CSV format for clustering data mining tasks. However, the dataset still needs to be stored locally and not in HDFS. For that purpose, `wekaClusteringTransformation()` can be called to download the dataset from HDFS to local storage. In the case of classification algorithms in WEKA, the data must be in ARFF format. Moreover, the possible values of the class label should be provided in the metadata of the ARFF file. The previously mentioned steps can be done by calling `wekaClassificationTransformation()`. `wekaClassificationTransformation()` performs several steps to transform the data into the required format. (1) Converting the dataset from CSV to ARFF format. (2) Downloading the dataset on local storage. (3) Iterating the whole dataset for searching for the unique values of the class label. (4) Adding the unique values of the class label to the metadata of the ARFF file.

WEKA accepts only datasets in one-hot coding with ARFF format for ARD data mining algorithms. A function in the `dataTransformation` package is responsible for that which is `wekaARDTransformation()`. The data transformation for ARD in WEKA is done by the same steps for classification transformation that are mentioned before except the third step. The third step, in this case, is transforming the dataset into a particular form that is required for executing FP-Growth in WEKA. WEKA requires each row in the dataset to contain number of attributes equal to the number of all items in the dataset. Then each row is filled by Boolean values which are true if the item exists in the corresponding row and false otherwise.

Mahout requires an input file in a SequenceFile format for clustering algorithms. SequenceFile format belongs to Apache Hadoop that is used in MapReduce as input and output format [33]. The responsible function for such transformation is `mahoutClusteringTransformation()`. The implementation of classification in Mahout does not support the automatic splitting of the train and test data for calculating the accuracy of the classification model. For that reason,
mahoutClassificationTransformation() splits the dataset into train and test data into two separated datasets as input to the mahout classification algorithms.

Spark does not require any special transformation steps since the submitted datasets, in this prototypical implementation, must be in CSV format and stored in HDFS which perfectly fit to Spark input data requirements.

The design of the previously mentioned packages has several advantages regarding the maintainability, extensibility, and loose coupling. More precisely, implementing each service in a separated package gives the ability to use each service separately without any dependency on other packages or services. For example, the dataTransformation package can be used as a separated service without any need for calling another service. That case is also true for each implemented execution engine. Another advantage of this design is ease of adding more execution engines to the prototypical implementation. More precisely, adding another execution engine requires only creating a new package for this new engine and implementing the interfaces that are existing in the executionEngines package. Adding more data mining algorithms to an existing engine is also easy by just adding different implementations of the data mining algorithms interfaces.

5.3 Service Architecture

The prototypical implementation in this thesis has a service-oriented architecture with a REST API. However, binding and service discovery are out of the scope of this thesis. Providing endpoints via a REST API to the prototypical implementation allows for more scenarios and uses cases for that implementation.

For example, by using the endpoints, the prototypical implementation can be used from different applications. These endpoints can also be deployed to implement graphical user interfaces using several available technologies that support calling HTTP requests. On the other hand, the endpoints give the flexibility of accessing the prototypical implementation from different devices when running the prototypical implementation in the cloud.

Since the prototypical implementation is developed using Java, the services are implemented in Spring Boot Framework\textsuperscript{7}. Spring Boot is built on top of Spring Framework. Spring Framework is used as an alternative to Java Enterprise Edition (JEE) for building Java applications [34]. Spring offers more efficient methods in the domain of dependency injection and building loosely coupled applications.

Spring Boot offers an efficient approach to build web and RESTful applications on top of an existing Java code. That allows for adding for more execution engines to the prototypical implementation efficiently by using the Java API of these engines.

In Section 5.1, it has been discussed that the prototypical implementation in this thesis has five main services. Fig. 21 shows the URL architecture of the prototypical implementation.

\textsuperscript{7} https://spring.io/projects/spring-boot
Fig. 21: URL Architecture
5.4 Database

The database in this prototypical implementation is responsible for storing the benchmark results and storing the services that represent all existing data mining algorithms. The stored data is used for selecting the most appropriate execution engine as discussed in Section 5.3.3.

The database of the prototypical implementation is implemented using MongoDB\(^8\) which is considered as NoSQL database. Using a NoSQL database here is for the sake of flexibility of changing the schema in the future. For example, there is no need for changing the schema if more metrics need to be stored later on. Another reason for that is that all needed information of the benchmark is stored in the single collection (equivalent to a table in the relational database). Hence, the benchmark collection can be exported and used as a performance benchmark for the existing execution engines efficiently without any joins operations that is required in the relational database systems.

The entity relationship diagram of the database is depicted in Fig. 22. The database consists of two collections which are ExecutionServices collection and Benchmark collection. The ExecutionServices collection plays a role that is similar to the role of the service directory for the services. Each data mining algorithm service is stored in the ExecutionServices collection. Each entry in this collection has the following information: service id, the data mining technique of this algorithm, the algorithm name, the corresponding execution engine, the datasets sample for benchmark usage, and finally the endpoint address of this data mining algorithm. The ExecutionServices collection is used by the benchmarking service in order to retrieve all data mining algorithms along with their dataset samples and the endpoint address of the data mining algorithm (see Section 5.2.2). Besides, the ExecutionServices collection is used by the automatic selection service to retrieve the endpoint address of the required data mining algorithm of the most appropriate selected execution engine (see Section 5.2.3).

The Benchmark collection is responsible for storing the benchmark results. Each entry in the collection represents the performance of the corresponding data mining algorithm with a single dataset sample. Each entry has the following information: benchmark entry id, the performed data mining algorithm, the used execution engine, attributes of the data size, and benchmark

\(^8\) https://www.mongodb.com
metrics. The data size and benchmark metrics are represented by separated objects. However, both objects along with the benchmark information are stored in one single collection which is Benchmark collection itself. The DataSize object contains the following information about the used dataset in the benchmark: dataset name, the volume, i.e., the size of the dataset in Bytes, number of attributes, and number of rows. Metrics objects contain the performance measures of the data mining algorithm. These measures are: runtime in seconds, CPU usage in percentage, RAM usage in percentage, and the quality of the executed data mining algorithm.

The Benchmark collection is used for storing the benchmark result (see Section 5.2.2). Besides, the Benchmark collection is used in automatic selection service to retrieve and compare the size of the datasets and to retrieve and compare the performance of the data mining algorithms for selecting the most appropriate execution engine (see Section 5.2.3).

Storing all benchmark results in one single collection allows for retrieving all benchmark results efficiently without any needed joins. In contrast to a relational database, NoSQL database systems allow for such information to be stored in a single collection without any needed normalization.

The database stores only the general characteristics of the datasets in the benchmark collection without storing any private nor corporate information about the dataset. Hence, that allows for several users and several companies to merge their benchmark databases. As a consequence, a better result may be achieved since the comparing in the selection process depends on more benchmark results.

The database is implemented as a separated service in order to offer an efficient way to access it. By calling the database service, the user can retrieve all benchmark results and all available data mining algorithms services.

5.5 Libraries

The prototypical implementation in this thesis uses several open source libraries in order to implement the data mining algorithms using different execution engines. These libraries offer a collection of APIs to call and execute different data mining algorithms using these engines.

In this section, all used libraries for implementing the execution engines in this thesis are discussed. More precisely, spark-core 2.2.0 and spark-mllib 2.2.0 are the used libraries for implementing Spark execution engine. The mahout-core 0.9 library is used for implementing Mahout execution engine. The weka-stable 3.8.2 which is used for implementing WEKA execution engine. Finally, spring-boot-starter 1.5.2 which is used for implementing the services’ endpoints.

In order to implement Spark execution engine, two libraries are used which are spark-core and spark-mllib. The spark-core library contains several packages that offer all required functions to deal with Spark as an engine and HDFS files as input data. By using this library, the user can define, manage, and interact with RDD data-sharing abstraction (see Section 2.3.1). The spark-core library also contains a specific package which is called hadoop-client. This package is responsible for offering an API to communicate with Hadoop components such as HDFS. By using the hadoop-client package, the user can access several functions to deal with HDFS files. The spark-mllib\(^9\) library is an open source, distributed machine learning library. All data mining

---

\(^9\) https://spark.apache.org/mllib
algorithms in Spark are implemented using the API that provided by the MLlib library. Using this library refers to the fact that the data mining algorithms in this library have an outstanding performance regarding the speed and the scalability [8]. The performance of the machine learning algorithms in MLlib is examined in several works [35][36][37][38]. Besides, the API that is offered by MLlib library has extensive documentation that allows the user to execute data mining algorithms effectively. MLlib consists of scalable and efficient implementations of several machine learning algorithms covering different techniques such as classification, regression, clustering, ARD, collaborative filtering, and dimensionality reduction.

The mahout-core library offers several packages that are used to implement Mahout execution engine. These packages offer implementations of several scalable machine learning algorithms along with an API for using these algorithms. The mahout-core library covers several machine learning algorithms from distributed linear algebra, regression, classification, clustering, ARD, and recommendation engines. The disadvantage of this library is that no extensive documentation explains the way of using this API. That affects adding more data mining algorithms for Mahout negatively.

The weka-stable library consists of several packages that offer API to execute data mining algorithms using WEKA execution engine. In addition to the data mining algorithms API, weka-stable library offers several functions to deal with ARFF files and functions for performing several transformation steps that required for input data. The weka-stable library covers an extensive range of machine learning algorithms. These algorithms cover classification, regression, clustering, ARD, and attribute selection. The advantage of this library is that there is well-explained and extensive documentation to use the provided API. Hence, the user can use this API easily.

Finally, spring-boot-starter library is used for building the endpoints that are used in the prototypical implementation. This library provides several notations and functions to build web and RESTful applications on top of an existing Java code.
6. Evaluation

The prototypical implementation, which is explained in the previous section, is used for evaluating the proposed approach. Since the selection of the most appropriate execution engine process has two steps, the evaluation process consists of two parts. The first step is to conduct the benchmark of the existing execution engines and the existing data mining algorithms. The second step is to select the most appropriate execution engine depending on the result of the conducted benchmark.

In Section 6.1 the experimental setup that is used for the evaluation process is explained. More precisely, it contains the information of the used hardware infrastructure, the versions of the used execution engines, and the characteristics of the used datasets for the evaluation purpose. In Section 6.2 the evaluation of the conducted benchmark in this thesis is covered. Finally, in Section 6.3, the selection of the most appropriate execution engine process is evaluated.

6.1 Experimental Setup

For this evaluation, a computing cluster is used which consists of 11 hosts which are ten slaves and one master. The master has 16 GB RAM, 1 TB HDD, and 10-core CPU. Each slave has 32 GB RAM, 150 GB HDD, and 6-core CPU.

The concept of this thesis is evaluated using three execution engines which are Spark, Mahout, and WEKA. The following versions of these execution engines are used: Spark 2.2.0 with MLlib 2.2.0, Mahout 0.9, and WEKA 3.8.2. Spark and Mahout are running on the entire computing cluster in a distributed manner. However, WEKA is running locally using a single host that is represented by one of the previously mentioned slaves. The reason is that pure WEKA does not support distributed execution (see Section 2.3.3).

Three data mining algorithms are used for the purpose of this experiment which are decision tree, k-Means, and FP-Growth. These algorithms are selected since they represent several data mining techniques and belong to the most commonly used algorithms from clustering, classification, and ARD. Moreover, all of these algorithms are implemented in all execution engines in the prototypical implementation. The parameters of each data mining algorithm are fixed during the whole experiment. For k-Means, the number of clusters $k$ is fixed to 3, and the number of max iterations is fixed to 100. For the decision tree, the ratio of training and testing data are fixed to 75% and 25% respectively. The training and testing data are split using random sampling. For FP-Growth, the minimum support and the minimum confidence are fixed to 0.6.

It has been discussed in Section 4 that the concept of this thesis depends on a hypothesis which is executing a data mining algorithm on similar datasets, using the same execution engine, yields comparable performance. In order to evaluate that concept, similar datasets are used. The similarity, in this context, is according to the characteristics of the data such as the number of attributes, the number of rows, and the size in Byte. For that reason, the used datasets, in this thesis, are categorized into two main groups. The first group contains the datasets that are used by the benchmark service to conduct the benchmark using these datasets. The second group contains those datasets that are used for evaluating the selection of the most appropriate execution engine process. The datasets in the second group are selected in such a way that they are similar to datasets in the first group. Furthermore, the datasets for the evaluation are chosen from the UCI Machine Learning Repository [39], the Frequent Itemset Data mining Dataset Repository [40], and SPMF Data mining Library [41].
Table 3 shows the details of the used datasets in the evaluation process. The table shows the following information of each dataset: the name of the dataset along with its origin, the abbreviation that is used later on instead of the full name, the number of attributes (columns), the number of rows, and the size of the dataset in Byte. Datasets marked with a star (*) are used for evaluating the selection of the execution engine. Within the previously mentioned groups, the datasets are categorized according to the applied data mining techniques which are classification, clustering, and association rule discovery. Datasets that are used for benchmarking the clustering data mining technique are derived from the datasets which are used for benchmarking the classification data mining technique. That can be achieved by removing the class label as well as every non-numeric value. Regarding the characteristics of datasets for ARD, each row represents a transaction with several items. The number of items in each transaction is not fixed and can be varying in the range between one and the number of all items that exist in the dataset. Hence, the number of attributes, of ARD datasets, represents the total number of items that exist in the dataset.

<table>
<thead>
<tr>
<th>Data mining Technique</th>
<th>Name</th>
<th>Abbreviation</th>
<th>Attributes</th>
<th>Rows</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Classification</strong></td>
<td>Motion Capture Hand Postures [39]</td>
<td>MHPCLS</td>
<td>13</td>
<td>77,405</td>
<td>11,584,897</td>
</tr>
<tr>
<td></td>
<td>Record Linkage Comparison Patterns [39]</td>
<td>CMPCLS</td>
<td>10</td>
<td>574,913</td>
<td>21,640,215</td>
</tr>
<tr>
<td></td>
<td>Poker Hand [39]</td>
<td>PKHCLS</td>
<td>11</td>
<td>1,000,000</td>
<td>24,538,370</td>
</tr>
<tr>
<td></td>
<td>SUSY [39]</td>
<td>SUSCLS</td>
<td>19</td>
<td>5,000,000</td>
<td>2,240,853,975</td>
</tr>
<tr>
<td></td>
<td>HIGGS [39]</td>
<td>HGSCLS</td>
<td>29</td>
<td>11,000,000</td>
<td>8,035,498,239</td>
</tr>
<tr>
<td></td>
<td>*MoCap Hand Postures [39]</td>
<td>POSCLS</td>
<td>13</td>
<td>77,405</td>
<td>11,584,969</td>
</tr>
<tr>
<td></td>
<td>*Covertype [39]</td>
<td>COVCLS</td>
<td>55</td>
<td>581,012</td>
<td>75,750,822</td>
</tr>
<tr>
<td></td>
<td>* KDD Cup 1999 Data [39]</td>
<td>KDDCLS</td>
<td>39</td>
<td>4,898,431</td>
<td>647,252,664</td>
</tr>
<tr>
<td></td>
<td>*HEPMASS [39]</td>
<td>HPMCLS</td>
<td>29</td>
<td>7,000,000</td>
<td>5,178,435,594</td>
</tr>
<tr>
<td><strong>Clustering</strong></td>
<td>Motion Capture Hand Postures [39]</td>
<td>MHPCLU</td>
<td>12</td>
<td>77,405</td>
<td>11,584,896</td>
</tr>
</tbody>
</table>

36
<table>
<thead>
<tr>
<th>Dataset Description</th>
<th>Method</th>
<th>Size</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Record Linkage Comparison Patterns [39]</td>
<td>CMPCLU</td>
<td>9</td>
<td>574,913</td>
</tr>
<tr>
<td>Poker Hand [39]</td>
<td>PKHCLU</td>
<td>10</td>
<td>1,000,000</td>
</tr>
<tr>
<td>SUSY [39]</td>
<td>SUSCLU</td>
<td>18</td>
<td>5,000,000</td>
</tr>
<tr>
<td>HIGGS [39]</td>
<td>HGSCLU</td>
<td>28</td>
<td>11,000,000</td>
</tr>
<tr>
<td>*MoCap Hand Postures [39]</td>
<td>POSCLU</td>
<td>12</td>
<td>77,405</td>
</tr>
<tr>
<td>*Covertype [39]</td>
<td>COVCLU</td>
<td>54</td>
<td>581,012</td>
</tr>
<tr>
<td>*Gas Sensors for Home Activity Monitoring [39]</td>
<td>SENCLU</td>
<td>11</td>
<td>928,991</td>
</tr>
<tr>
<td>*KDD Cup 1999 Data [39]</td>
<td>KDDCLU</td>
<td>38</td>
<td>4,898,431</td>
</tr>
<tr>
<td>*HEPMASS [39]</td>
<td>HPMCLU</td>
<td>28</td>
<td>7,000,000</td>
</tr>
<tr>
<td>Sign [41]</td>
<td>SIGARD</td>
<td>267</td>
<td>730</td>
</tr>
<tr>
<td>Retail [40]</td>
<td>RTLARD</td>
<td>16,470</td>
<td>88,162</td>
</tr>
<tr>
<td>RecordLink [41]</td>
<td>RELARD</td>
<td>27</td>
<td>574,913</td>
</tr>
<tr>
<td>PowerC [41]</td>
<td>PWRARD</td>
<td>125</td>
<td>1,039,998</td>
</tr>
<tr>
<td>PAMP [41]</td>
<td>PMPARD</td>
<td>82</td>
<td>1,000,000</td>
</tr>
<tr>
<td>*Chess [40]</td>
<td>CHSARD</td>
<td>75</td>
<td>3,196</td>
</tr>
<tr>
<td>*OnlineRetail [41]</td>
<td>ONRARD</td>
<td>2,604</td>
<td>541,909</td>
</tr>
<tr>
<td>*T40I10D100K [40]</td>
<td>TDKARD</td>
<td>942</td>
<td>100,000</td>
</tr>
<tr>
<td>*Accidents [40]</td>
<td>ACCARD</td>
<td>467</td>
<td>340,183</td>
</tr>
<tr>
<td>*USCensus [41]</td>
<td>USNARD</td>
<td>316</td>
<td>1,000,000</td>
</tr>
</tbody>
</table>

Table 3: Evaluation Datasets
6.2 Benchmark

As discussed before, the selection process of the most appropriate execution engine depends on the result of the conducted benchmark. This section discusses the result of the benchmark in order to examine the reliability of the captured measures for the selection process. The benchmark is conducted using the prototypical implementation and the computing cluster that is mentioned before. Each of the data mining algorithms in this benchmark has been executed five times using the three execution engines on the datasets from the first group in Table 3. The shown performance measures, in the following figures, represent median values of the five executions.

Fig. 23 - Fig. 25 show the runtime benchmark result for all data mining algorithms in seconds. It is evident from Fig. 23 that, WEKA outperformed the other engines in executing the decision tree algorithm on small datasets. However, WEKA is not able to execute the decision tree algorithm on larger datasets such as SUSCLS or HGSCLS. The reason is that WEKA loads the entire dataset into the main memory [13][42]. However, the assigned RAM, in this case, is not enough to process these large datasets which causes an out of memory error in case of such large datasets. The performance of Mahout is comparable to the performance of WEKA when executing the decision tree algorithm. However, Mahout has a better performance than WEKA for larger datasets (see PKHCLS dataset). Although Mahout is running on the entire computing cluster in a distributed manner, Mahout is not able to process larger datasets due to out of memory error since the assigned RAM was not enough to process the corresponding dataset. The out of memory error in Mahout is caused only when executing the decision tree algorithm. Hence, the concrete implementation of the decision tree algorithm in Mahout might lead to use more RAM than the assigned RAM while executing the algorithm on larger datasets. Spark has the worst runtime measure for the decision tree algorithm among the other two execution engines. However, Spark can process larger datasets that other engines were not able to process. The reason might be that the data is distributed on several hosts, hence more RAM in total is available. If the total RAM is consumed, then Spark uses the disk to store this data or recomputes the corresponding RDDs using the lineage method each time the RDDs are needed [43].

Fig. 24 shows the runtime benchmark for the k-Means algorithm. Obviously, Spark outperformed the other two execution engines with almost similar performance to WEKA for small datasets (see MHPCLU dataset). Similar to the benchmark of the decision tree, WEKA was not able to process large datasets for the exact reason that explained before. Mahout has the worst performance among other execution engines in executing the k-Means algorithm. This might refer to that the implementation of k-Means in Mahout requires initial clusters to be passed as input parameters to the k-Means algorithm. These initial clusters are generated using Canopy clustering algorithm [44]. This prerequired step causes the overhead in runtime while performing k-Means using Mahout. Obviously, Spark outperformed Mahout in smaller datasets due to the generation of the initial clusters. However, for larger dataset such as HGSCLU, Mahout has a better performance than Spark regarding the execution runtime. This might refer to that Mahout uses SequenceFile as a format for the input dataset for k-Means. SequenceFile uses key-value approach to store the data which provides a more efficient solution to store small blocks of data [45]. Hence, allows for reducing the needed time to access the data. Using SequenceFile might be more efficient than Spark RDDs for larger datasets, that might be the reason for the better performance of Mahout with larger datasets.
Fig. 23: Runtime Benchmark for Decision Tree

Fig. 24: Runtime Benchmark for k-Means

Fig. 25: Runtime Benchmark for FP-Growth
Fig. 25 shows the runtime benchmark for the FP-Growth data mining algorithm. As discussed in Section 5.2.4, WEKA requires the dataset in a particular form in which each row has is filled by Boolean values equal to the number of all items that exist in the dataset. That means the dataset is transformed to a larger dataset due to including the false values for missing items in each row. Obviously, WEKA outperformed the other engines for small datasets. However, WEKA is not able to execute the FP-Growth algorithm on larger datasets such as RTLARD or PWRARD. The reason is that these two datasets have large number of attributes. For example, RTLARD has 16,470 items which increase the required values to 16,470 values for each row. As a consequence, increasing the size of the dataset in such a way that it will not fit into the assigned main memory. Hence, the ability to process the dataset in WEKA can be measured according to the total number of values across the whole dataset, i.e., the number of items multiplied by the number of rows. That gives a reason for the deviation of the execution runtime of PMPARD in WEKA since PMPARD is transformed to a larger dataset compared to the original dataset the is processed by Spark and Mahout. Using the current experimental installation, WEKA was able to process the dataset only if the dataset has less than 90 million values in total. Note that, this maximum value has been determined in an empirical method and it is only valid for the used datasets used and the used hardware infrastructure.

Fig. 26 - Fig. 31 show the consumption of RAM and CPU benchmark for all data mining algorithms. The measures for Spark and Mahout represent the usage of RAM and CPU percentage for the whole cluster whereas the measures for WEKA represent the usage percentage in only the used single host. For example, RAM usage of Spark while executing the decision tree algorithm on MHPCLS dataset is 11.73% of all used RAM in the cluster which equals to 39.41 GB. Whereas, The RAM consumption of WEKA for the same dataset is 16.44% of RAM in the single host which is equal to 5.26 GB. Apparently, 39.41 GB is way bigger than 5.26 GB, although 16.44% is more significant than 11.73% as percentages. Hence, the RAM and CPU usage is not compared across all investigated execution engines. The RAM and CPU consumption measures are retrieved using the Cloudera Manager API [46]. That API allows retrieving the consumption of resources for the whole computing cluster with the ability to retrieve these measures for a specific host too. Example of request/response for Cloudera Manager API can be found in Appendix 2. After executing the benchmark several times and in different points of time it has been noticed that the measures of CPU and RAM usage are not stable even for the same dataset and using the same execution engine. That refers to that, not only the execution engine is running on the cluster, but several applications and several services are also running on the cluster which the execution engines rely on, e.g., Zookeeper for Spark and Mahout. The issue here is that the consumption of RAM and CPU of these services are not depending on the submitted dataset nor they remain stable over time. Besides, it is not possible, at least using the current prototypical implementation, to measure the consumption of RAM and CPU of the execution engine alone. However, the shown median values in the following figures are satisfying since the consumption of resources is in a proportional relationship with the size of the dataset within the same execution engine.

It is obvious from Fig. 26 - Fig. 28 that, the RAM’s consumption of Spark is less than the RAM’s consumption of Mahout while executing the decision tree algorithm whereas Mahout generally has less RAM’s consumption than Spark while executing the k-Means and the FP-Growth algorithms. According to Fig. 29 - Fig. 31, the CPU’s consumption of Spark is less than in Mahout while executing the k-Means algorithm whereas Mahout has less consumption of CPU than Spark while executing the decision tree algorithm and the FP-Growth algorithm. However, the consumption measures of RAM and CPU are very similar across Spark and Mahout while executing the investigated data mining algorithms.
Fig. 26: RAM Usage Benchmark for Decision Tree

Fig. 27: RAM Usage Benchmark for k-Means

Fig. 28: RAM Usage Benchmark for FP-Growth
**Fig. 29: CPU Usage Benchmark for Decision Tree**

<table>
<thead>
<tr>
<th>Datasets</th>
<th>Spark</th>
<th>Mahout</th>
<th>Weka</th>
</tr>
</thead>
<tbody>
<tr>
<td>MHPCLS</td>
<td>4.00%</td>
<td>3.00%</td>
<td>3.22%</td>
</tr>
<tr>
<td>CMPCLS</td>
<td>3.55%</td>
<td>3.15%</td>
<td>3.12%</td>
</tr>
<tr>
<td>PKHCLS</td>
<td>4.00%</td>
<td>3.00%</td>
<td>3.21%</td>
</tr>
<tr>
<td>SUSCLS</td>
<td>5.00%</td>
<td>4.65%</td>
<td>4.14%</td>
</tr>
<tr>
<td>HGSCLS</td>
<td>5.95%</td>
<td>5.50%</td>
<td>4.78%</td>
</tr>
</tbody>
</table>

**Fig. 30: CPU Usage Benchmark for k-Means**

<table>
<thead>
<tr>
<th>Datasets</th>
<th>Spark</th>
<th>Mahout</th>
<th>Weka</th>
</tr>
</thead>
<tbody>
<tr>
<td>MHPCLU</td>
<td>5.70%</td>
<td>4.65%</td>
<td>3.28%</td>
</tr>
<tr>
<td>CMPCLU</td>
<td>5.70%</td>
<td>4.65%</td>
<td>3.28%</td>
</tr>
<tr>
<td>PKHCLU</td>
<td>6.05%</td>
<td>5.50%</td>
<td>4.13%</td>
</tr>
<tr>
<td>SUSCLU</td>
<td>5.00%</td>
<td>4.65%</td>
<td>3.06%</td>
</tr>
<tr>
<td>HGSCLU</td>
<td>6.05%</td>
<td>5.50%</td>
<td>4.13%</td>
</tr>
</tbody>
</table>

**Fig. 31: CPU Usage Benchmark for FP-Growth**

<table>
<thead>
<tr>
<th>Datasets</th>
<th>Spark</th>
<th>Mahout</th>
<th>Weka</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIGARD</td>
<td>1.70%</td>
<td>1.70%</td>
<td>1.60%</td>
</tr>
<tr>
<td>RTLARD</td>
<td>2.94%</td>
<td>2.94%</td>
<td>3.11%</td>
</tr>
<tr>
<td>RELARD</td>
<td>2.98%</td>
<td>2.98%</td>
<td>3.11%</td>
</tr>
<tr>
<td>PWRARD</td>
<td>2.88%</td>
<td>2.88%</td>
<td>3.09%</td>
</tr>
<tr>
<td>PMPARD</td>
<td>3.59%</td>
<td>3.59%</td>
<td>3.59%</td>
</tr>
</tbody>
</table>
As discussed in Section 4.2, the quality measure of clustering data mining algorithms is represented by the sum of squared errors between all items in the cluster and the centroid. However, Mahout does not offer such a function to calculate the sum of the squared error for the created clusters. That is why the quality measure for the k-Means is excluded for this conducted benchmark. However, only quality measure for the decision tree algorithms is evaluated which is represented by the accuracy of the algorithm.

Fig. 32 shows the accuracy measure benchmark for the decision tree algorithm. As discussed before, each dataset is split into training data and testing data with 75% and 25% ratios respectively for all execution engines. The accuracy is represented by the percentage of the correctly predicted values of the testing data by the classification model. From Fig. 32 it is obvious that the concrete implementation of the decision tree algorithm within each execution engine plays a vital role in the accuracy measure [47]. That explains the difference in the accuracy measure for the same dataset across different execution engine. However, it is obvious that the accuracy measures also differ within the same execution engine across different datasets. Several data characteristics affect the accuracy measures of the decision tree algorithm according to D. Oreski, et al [48]. These characteristics, according to their importance in descending order, are the number of features, feature noise, data sparsity, homogeneity, correlation, and number of rows. Hence, the accuracy measure cannot be compared according to the size of the dataset only.

![Fig. 32: Accuracy Benchmark for Decision Tree](image)

### 6.3 Selection Process

It has been discussed in Section 4.3 that the selection process of the most appropriate execution engine depends on a dataset similarity criterion. This criterion considers the characteristics of the dataset. In Section 6.3.1, several distance metrics, regarding the dataset characteristics, will be evaluated whether they can be used as a valid similarity criterion. In Section 6.3.2, the overhead time that is caused by the selection process is evaluated. Finally, in Section 6.3.3, the time savings using the selection process is quantified.
6.3.1 Distance Metrics

Different distance metrics can be defined from the data characteristics to be used as a similarity criterion. In this section, several distance metrics are evaluated. The evaluation is conducted using the second group of the datasets in Table 3 with a (*) sign since they are similar to the datasets which are used for the benchmark. These datasets are used to execute the same data mining algorithms on the same execution engines that with the already existing benchmarks. Evaluating the distance metrics is done via considering whether the selected execution engine, using that metric, has the best performance among other execution engines.

It has been discussed in Section 6.2 that only the quality measure of the decision tree algorithm is available which is represented by the accuracy of prediction. However, the accuracy measure for the decision tree algorithm cannot be compared according to only the investigated characteristics of the dataset. However, several other characteristics should be considered which are feature noise, data sparsity, homogeneity, and correlation. Hence, the accuracy measure cannot be comparable since the previously mentioned characteristics are not considered in this thesis.

According to Section 6.2, the consumption of RAM and CPU measures are not comparable across all of the investigated execution engines. That refers to that the measures for Spark and Mahout address the consumption percentage for the whole cluster (11 CPUs and 336 GB RAM). However, the measures for WEKA address the consumption percentage only for the single used host (1 CPU and 32 GB RAM). Hence, the selection process of the most appropriate execution engine can be evaluated according to RAM and CPU consumption criterion regarding only Spark and Mahout.

Based on the foregoing, the runtime criterion is examined for the evaluation of the most appropriate execution engine selection process across all investigated execution engines. Whereas, the RAM and CPU usage criteria are examined for the selection process using only Spark and Mahout. The execution measures of the executed data mining algorithms using all available execution engines on the datasets in Table 3 with * sign can be found in Appendix 1.

Let the following characteristics of the dataset be used to examine the similarity between datasets. The characteristics are: (1) the size of the dataset in bytes, (2) the number of features, i.e., columns that the dataset has, and (3) the number of rows in the dataset. The previously mentioned characteristics are generic characteristics that can be measured easily. However, other measures represent the characteristics of the dataset such as data sparsity measures, statistical measures, information theoretic measures and noise measures [48]. These measures are excluded in this thesis since they are time-costly to measure compared to the three considered characteristics.

Different distance metrics can be derived from the previously mentioned characteristics. Let \( A, B \) be datasets, then \( F_A, F_B \) are the number of features in \( A, B \) respectively. \( R_A, R_B \) are the number of rows existing in \( A, B \) respectively. \( S_A, S_B \) are the sizes in Bytes of \( A, B \) respectively. Then, \( DM \) is a distance metric between \( A \) and \( B \). Smaller values for \( DM_{A,B} \) indicate a higher similarity and vice versa. In this thesis, five different distance metrics are evaluated. These metrics are represented by different combinations of the previously mentioned dataset’s characteristics. Equation 1 represents the first assumed distance metric which considers all the three previously mentioned characteristics.
\[ DM_{A,B} = |F_A - F_B| + |R_A - R_B| + |S_A - S_B| \] (1)

As a second attempt, let Equation 2 be a distance metrics with considering only the size of the dataset in Bytes since this number is the most significant number between all other characteristics in several datasets.

\[ DM_{A,B} = |S_A - S_B| \] (2)

Equation 3 is a distance metric that considers only the number of rows in the dataset as a third attempt.

\[ DM_{A,B} = |R_A - R_B| \] (3)

Equation 4 represents the third attempt which considers only the number of features in the dataset.

\[ DM_{A,B} = |F_A - F_B| \] (4)

Finally, Equation 5 is the last attempt to formulate the distance metric which considers both numbers of rows and number of attributes of the dataset.

\[ DM_{A,B} = |F_A * R_A - F_B * R_B| \] (5)

The previously mentioned equations are used to indicate the similarity between the datasets in both groups in Table 3. As a consequence, selecting the most appropriate execution engine depending on these similar datasets. In the following subsections, the accuracy of the previously mentioned distance metrics is evaluated. More precisely, in Section 6.3.1.1, the accuracy of distance metrics is evaluated according to runtime criterion. In Section 6.3.1.2, the accuracy of distance metrics is evaluated according to RAM and CPU criteria. The accuracy percentage represents whether the selected execution engine, using that equation, has the best performance among other execution engines according to the corresponding criterion.

### 6.3.1.1 Runtime Criterion

Fig. 33 shows the accuracy of the previously mentioned distance metrics according to runtime criterion. The accuracy percentage represents whether the selected execution engine, using that equation, has the best performance among other execution engines according to runtime criterion. Using Equation 1, only eleven datasets out of fifteen datasets have the best execution engine as a selected engine using that equation. Hence, the accuracy of the selection process using that equation is 73.00%. Equation 1 consists of all previously mentioned characteristics of the dataset. However, the size in Bytes for several datasets are more significant than the number of its features and the number of its rows. Hence, these two numbers are negligible compared to the size of the corresponding dataset.

It is evident from Fig. 33 that Equation 2 has the same effect of Equation 1 with an accuracy of 73.00%. Hence, no gain in accuracy using only the size in Bytes of the dataset. Obviously, Equation 3 has better performance than Equation 2 with 80.00% accuracy in total. Equation 3 achieves very high accuracy for the decision tree data mining algorithm which is the accuracy of 100.00%. However, Equation 3 has the worst performance for FP-Growth. That might refer to that the dataset is transformed in case of WEKA to a larger dataset since all rows must have a Boolean value for each attribute in the dataset (see Section 5.2.4). Equation 4 has the worst performance among all other equation with an accuracy of 46.67%. The number of features in...
a dataset alone cannot be reliable for the selection process since that it might be two datasets which have the same number of features but one of them has more rows. Then the dataset that has more rows will take more time to be processed. Equation 5 considers the number of existing values in a dataset which is the number of rows multiplied by the number of features. The resulted number represents the number of all values that exist in the dataset. Equation 5 achieved a satisfying accuracy that equals 87.00%. Equation 5 is the best distance metric for decision tree with 100.00% of accuracy since it considers all values that exist in the dataset.

![Graph](image)

**Fig. 33: Accuracy of Distance Metrics Equations According to the Runtime Criterion**

Referring to Section 6.2, Mahout, in case of the decision tree algorithm, and WEKA were not able to process larger datasets since the assigned RAM was not enough to process these datasets which causes out of memory error. The cases where the selected execution engine was not able to process the corresponding dataset, because of the previously mentioned reason, are listed in Table 4. Obviously, the first four equations lead to such wrong selections. However, Equation 5 does not cause any of these wrong selections.

<table>
<thead>
<tr>
<th>Equation</th>
<th>Decision Tree</th>
<th>k-Means</th>
<th>FP-Growth</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equation 1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Equation 2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Equation 3</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Equation 4</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Equation 5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

*Table 4: Times of Selecting A Wrong Execution Engine That Cannot Process the Corresponding Dataset*
Based on the foregoing, Equation 5 is used for the remaining evaluation that relates to runtime criterion since it is the most efficient equation to indicate the similarity between datasets. Besides, Equation 5 does not lead to any wrong selection of the execution engine that cannot process the corresponding dataset. However, it is also possible to use several distance metrics for different data mining algorithms, i.e., one distance metric for each data mining algorithm, to achieve higher overall accuracy. For example, Equation 3 can be used for the decision tree data mining algorithm since it has an accuracy of 100% for that algorithm, and Equation 5 can be used for the other data mining algorithms. As a consequence, increasing the overall accuracy percentage across all data mining algorithms. However, the accuracy of the selection process also may be increased by expanding the benchmark with more datasets that have different characteristics.

6.3.1.2 RAM and CPU Criteria

In this section, the distance metrics above are evaluated whether they can be reliable for selecting the most appropriate execution engine with the minimum RAM and CPU usage regarding only Spark and Mahout. In Fig. 34, the accuracy of all distance metrics equation according to RAM usage criterion is shown. Obviously, all distance metrics achieve the same accuracy. For the decision tree and the k-Means algorithms, the accuracy of all distance metrics is of 100.00%. According to Section 6.2, the RAM consumption of Mahout was less than Spark in the decision tree algorithms for all datasets. That means the distance metric does not play any role for the selection process since Mahout will be chosen as the most appropriate execution engine regardless of the characteristics of the dataset. The same case for the k-means algorithms, but here, the chosen execution engine is Spark. However, for the FP-Growth algorithm, the RAM consumption was not stable during the execution of the FP-Growth on the datasets from the second group (see Appendix 1). The instability of the RAM consumption refers to the unstable consumption of the services other than the execution engine that is also running on the cluster (see Section 6.2). Hence, the selection process cannot rely on the RAM usage criterion although all metrics achieved a high accuracy of selection for the used datasets.

![Fig. 34: Accuracy of Distance Metrics Equations According to the RAM Usage Criterion](image-url)
In Fig. 35, the accuracy of all distance metrics equation according to CPU usage criterion is shown. Obviously, none of the distance metrics achieves a satisfying accuracy. That is because of the previously mentioned reason which is the instability of the CPU consumption of the other services. Hence, the selection process cannot rely on the CPU usage criterion.

Fig. 35: Accuracy of Distance Metrics Equations According to the CPU Usage Criterion

Based on the foregoing, only the runtime criterion can be reliable for the selection process of the most appropriate execution engine. Hence, the remaining evaluation is according only the runtime criterion.

6.3.2 Runtime Overhead

In this section, the runtime overhead that is caused by the selection process is evaluated. The selection process here depends on Equation 5. Fig. 36 - Fig. 38 show the total execution time that is consumed while executing data mining algorithms using the automatic selection service. The execution time is measured in seconds and split into the following: the selection time, the transformation time, and the runtime that represents the actual execution time of the data mining algorithm by the selected execution engine. The selection time is the time that is consumed for the selection process of the most appropriate execution engine which is described in details in Section 5.2.3. The average selection time for the 15 datasets is 0.19 seconds. Obviously, the selection process causes an overhead, but it is negligible compared to the actual execution time of the data mining algorithm.

The transformation time is the consumed time to transform the data into the required format by the selected execution engine (see Section 5.2.4). The transformation time of the dataset heavily depends on its size and the target execution engine too. The data transformation step, in some cases, can be costlier than processing the data itself. For example, the transformation step of HPMCLU to the required format by Mahout costs 738.75 seconds and processing the data with Mahout costs only 488.5 seconds which makes the total cost of transformation and execution reaches to 1227.55 seconds. However, Spark consumes 830.7 seconds to process HPMCLU.
without any transformation steps are required. Hence, the automatic selection process here actually causes overhead with 396.85 seconds. However, the transformation steps for other cases within the 15 datasets cost around 9 seconds in average which is still negligible according to the gained time due to choosing the execution engines with better performance.

Fig. 36: Decision Tree Execution Time Using the Automatic Selection Service in Seconds

Fig. 37: k-Means Execution Time Using the Automatic Selection Service in Seconds
6.3.3 Quantifying Runtime Savings

In this section, a comparison is conducted regarding the execution time of data mining algorithms using both the automatically selected execution engine and the execution engines that have the worst performance. Fig. 39 - Fig. 41 quantify the runtime of these execution engines in order to show the maximum runtime savings that can be achieved by exploiting automatic selection. Fig. 39 shows the execution time comparison for the decision tree data mining algorithm. Obviously, the two datasets \textit{KDDCLS} and \textit{HPMCLS} were able to be processed only using Spark whereas the other engines faced the out of memory error. Hence, there a selection time as an overhead for both datasets. For \textit{POSCLS} dataset, the time gain for using the automatic selection is 47.65\%. For \textit{COVCLS} dataset, the time gain for using the automatic selection is 20.08\%. In the case of \textit{SENCLS} dataset, the time gain for using the automatic selection is 24.02\%. Hence, using the automatic selection for the decision tree data mining algorithm might save up to 47.65\% of the execution runtime.

Fig. 38: FP-Growth Execution Time Using the Automatic Selection Service in Seconds

Fig. 39: Decision Tree Execution Time Using the Automatic Selection and the Worst Engine
Fig. 40 shows the execution time comparison for the k-Means data mining algorithm. The POSCLU dataset is processed using the automatically selected execution engine which achieves 85.73% less execution runtime. In the case of COVCLU dataset, the time gain for using the automatic selection is 91.69%. For SENCLU dataset, the time gain for using the automatic selection is 90.44%. In the case of KDDCLU dataset, the time gain for using the automatic selection is 67.35%. However, for HPMCLU dataset, there was no time gain since the transformation step consumed much time and caused 47.78% waste of time compared to the execution time of the worst execution engine. Using the automatic selection for the k-Means data mining algorithm can save up to 91.69% of the execution runtime, but on the other hand, it might waste up to 47.78% of the execution time due to the data transformation step.

Fig. 40: k-Means Execution Time Using the Automatic Selection and the Worst Engine

Fig. 41 shows the execution time comparison for the FP-Growth data mining algorithm. The CHSARD dataset is processed using the automatic selection of the execution engine which achieves 98.45% less execution runtime. Although Spark can process larger datasets with less runtime, Spark processes CHSARD dataset in an unexpected high runtime. In the case of ONRARD dataset, the time gain for using the automatic selection is 3.91% which is the lowest time gain across all data mining algorithms and all datasets. For TDKARD dataset, there was no time gain due to the selection time which causes 2.71% waste of time compared to the time of the worst execution engine. In the case of ACCARD dataset, the time gain for using the automatic selection is 27.96%. For USNARD dataset, the time gain for using the automatic selection is 9.96%. Using the automatic selection for the k-Means algorithm might save up to 98.45% of the execution runtime, but on the other hand, it might waste up to 2.71% of the execution time due to the selection step. However, this overhead is negligible compared to the execution time.
**Fig. 41: FP-Growth Execution Time Using the Automatic Selection and the Worst Engine**

The selection of the most appropriate execution engine process which depends on Equation 5 has been evaluated in this section. As a conclusion, the selection process in several cases was able to select the execution engine which has the best performance regarding the execution time. Besides, the selection process avoids the out of memory error due to the inability of the execution engine to process larger datasets since the assigned RAM in not enough for the processing operation. However, the data transformation step might cause overhead if the selected execution engine requires these transformation steps to accept the dataset as a valid input.
7. Conclusion

The information era that is created by the steadily growing data in different fields causes the phenomena of drowning in data with suffering from the lack of useful knowledge. That motivated several works to be conducted for the sake of discovering valuable knowledge from this huge amount of data. The KDD process is one process that is introduced to discover useful knowledge by analyzing this data [2]. The most important step in the previously mentioned process is the data mining step.

Nowadays, there are multiple execution engines which implement different processing paradigms for data mining. Especially for novice or inexperienced analysts, it is often not clear which execution engine to choose in order to execute data mining algorithms. Several works were introduced in order to help the user in selecting the appropriate execution engine. These works are mainly benchmarks for evaluating the performance of these execution engines [21][22][23][35][25][26][27]. However, these benchmarks have several issues regarding their objectivity and the considered measures. For example, several previous works in this area ignore the impact of varying the dataset size on the performance. Furthermore, several of these works focus on the execution runtime with ignoring other measurements such as consumption of resources or the quality of the data mining algorithm.

This thesis proposed a novel approach to help novice users in selecting the most appropriate execution engine automatically for executing the needed data mining algorithm. The selection process compares the performance of different execution engines on similar datasets and selects the one with the best performance according to the user-defined criterion. The logic of the selection process depends on a hypothesis which is executing a data mining algorithm on similar datasets, using the same execution engine, yields comparable performance. In this thesis, several distance metrics were introduced which are derived from the generic characteristics of the dataset. The investigated characteristics are: (1) the number of attributes, (2) the number of rows, and (3) the size of the dataset in Bytes. Besides, several criteria of the selection process were introduced which are: (1) runtime, (2) RAM usage, (3) CPU usage, and (4) quality of the data mining algorithm.

In addition to the automatic selection of the execution engine, the introduced approach in this thesis allows the user to choose the execution engine manually for executing data mining algorithms according to the user need. That is achieved abstracting several execution engines with various data mining algorithms.

The proposed approach in this thesis allows users to perform benchmarks automatically for several execution engines using different data mining algorithms and different datasets. The benchmark result is stored so it can be used for the automatic selection later on. The benchmark captures only general characteristics of the dataset. That means, no private nor corporate information about the data is stored in the benchmark which allows for several users and several companies to merge their benchmark databases. That perhaps allows for achieving a better selection of the execution engine since the selection process depends on the benchmark database. The prototypical implementation for the introduced approach has a loosely coupled and extensible architecture that gives it the flexibility to add more execution engines and more data mining algorithms later on.
The prototypical implementation in this thesis performs data transformation steps. These steps are responsible for transforming the input data into the corresponding format in order to be considered as a valid input by each execution engine. That helps novice users to transform the data easily to the required format by the execution engines. From the other hand, the data transformation is used by the automatic selection of the execution engine so that input data will be transformed into the required format by the automatically selected execution engine.

The prototypical implementation of the introduced approach has a service-oriented architecture with a REST API. Providing endpoints via a REST API to the prototypical implementation allows for the efficient execution of data mining algorithms for novice users who do not have a profound programming background. Hence, it allows users to concentrate more on designing the steps of processing the data and defining which data mining algorithm to execute rather than concentrating on the programming level.

The concept of this thesis is evaluated using three execution engines which are Spark, Mahout, and WEKA. Three data mining algorithms are used in this experiment which are decision tree, k-Means, and FP-Growth. These algorithms are selected since they represent several data mining techniques which are clustering, classification, and ARD. Furthermore, they are considered from the most commonly used data mining algorithms [4]. Several datasets of different sizes from different domains are used for the conducted experiment. The evolution of the introduced concept is divided into two main parts: (1) the evaluation of the conducted benchmark and (2) the evaluation of the automatic selection of the execution engine according to the previously mentioned criteria.

The first part of the evaluation discusses and compares the performance of the previously mentioned execution engines regarding the execution runtime, the consumption of resources, i.e., RAM and CPU, and the quality of the data mining algorithms. The experimental results unveil that the previously mentioned measures, except the execution runtime, cannot be reliable for comparing the performance of different execution engines at least for the current experimental installation and the prototypical implementation. The reason is that these measures are not stable because of several factors which are influencing these measures other than the corresponding dataset. One of these factors is that the resource consumptions cannot be compared across all the investigated execution engines. That refers to that resource consumptions in case of WEKA are measured in one single host whereas the consumptions in case of Spark and Mahout are measured in the whole computing cluster including all hosts. Another factor is that several applications and several services are also running on the cluster which the investigated execution engines rely on, e.g., Zookeeper for Spark and Mahout. However, the consumption of RAM and CPU of these services do not depend on the submitted dataset, nor they remain stable over time. Hence, the consumption of the execution engine alone cannot be measured.

The quality measure of the data mining algorithm differs according to the applied data mining technique. Two of the quality measures are discussed in this thesis. The first measure is the sum of squared error for clustering technique. However, this measure cannot be compared across various datasets since this measure depends on the concrete values that exist in the dataset. The second measure is the prediction accuracy of the classifier model in classification technique. However, the experimental results show that the accuracy measure massively differs for each dataset and it does not depend only on the investigated general characteristics of the dataset. In contrast to runtime, the accuracy measures cannot be generalized since that the accuracy
measures differ even in case of datasets with similar sized which are processed using the same execution engine.

The experimental results unveil that WEKA generally can be used for small datasets in order to achieve high runtime savings in contrast to the other distributed execution engines. Generally, Spark is the best execution engine to process large datasets across all investigated data mining algorithms because of the distributed execution on several hosts. However, Mahout has the best performance regarding runtime while executing the FP-Growth algorithm. Finally, Mahout and WEKA have several issues regarding the data transformation steps which are required for the input datasets. These required transformation steps cause time overhead that is added eventually to the execution runtime.

The second part of the evaluation evaluates five different distance metrics regarding the investigated characteristics of the datasets and the selection criteria. These distance metrics are examined whether they can be a valid similarity criterion between datasets since the selection process of the most appropriate execution engine depends on comparing the performance of similar datasets. The experimental results unveil that only the runtime criterion can be reliable for the selection process. Whereas, the other criteria which are RAM usage, CPU usage, and quality cannot be reliable for the selection process since that the depending measures facing several issues because of the previously mentioned reasons. The experimental results show that Equation 5, which considers the number of rows and the number of attributes of the dataset, is the most efficient equation among other introduced equation for the selection process regarding the runtime criterion.

The experimental results show that the selection of the most appropriate execution engine process which depends on Equation 5 can save up to 98.45% of the execution time of the data mining algorithm in some cases. However, in some cases, the automatic selection of the most appropriate execution engine might cause an overhead and waste of time that reach up to 47.78% compared to the execution time of the execution engine with the worst performance. This overhead is caused by the data transformation step that is required to transform that input data into the required format by the automatically selected execution engine. Hence, this prototypical implementation needs improvement to overcome the overhead of transforming the data into an appropriate format to be accepted by the selected execution engine. However, the data transformation step is considered as an issue that is related to the investigated execution engines since they require these specific or even proprietary data formats. Hence, the data transformation step cannot be considered as a disadvantage for the purposed approach of this thesis. Moreover, the proposed approach strives to help users to overcome the issue of the required data transformation step.
8. Outlook

The automatic selection of the execution engine concept which introduced in this thesis can achieve up to 98.45% of time optimization by selecting the most appropriate execution engine that suits the submitted dataset and the needed data mining algorithm. However, this approach can be enhanced and improved regarding several prospects.

It has been discussed before that the automatic selection of the most appropriate execution engine can depend only on the runtime criterion. Hence, the improvement can be done by enabling the other criteria which are the consumption of RAM and CPU and the quality of the result of the data mining algorithm. That can be done by several steps. Firstly, adding the distributed version of WEKA instead of the pure version which allows running WEKA using the whole cluster. That allows for a performance that is comparable to the other execution engines that run in a distributed manner. As a consequence, allowing the measures of resources for WEKA performance to be comparable to the measures of performance for other execution engines that run in a distributed manner using the whole computing cluster. Secondly, enhancing the prototypical implementation in such a way that the captured measures of resources’ consumption represent the consumption of the execution engine alone. Currently, the measures of the consumption of resources refer to the consumption within the whole cluster or within the whole single used host. Hence, these measures represent also the consumption of the other running applications and services in the cluster which the execution engines rely on. That causes instability of these measures each time the execution takes place even though when executing the same data mining algorithm on the same dataset using the same execution engine. Thirdly, extending the current distance metric to be reliable for the quality criterion by considering more data characteristics such as the distribution and the sparsity of the data. Finally, including other criteria that might be of interest to users while executing data mining algorithms such as CPU flops or other performance measures that users, in case of using cloud services, must pay for.

Currently, several data mining algorithms in the implemented execution engines require a particular format for the input dataset. The performed transformation steps in several cases cause overhead in execution time. Hence, an improvement prospect is to enhance the current implementation in order to force the execution engines to deal with a single unified format of the data. That allows for eliminating the time needed for transforming the data which causes overhead in execution time.

The current prototypical implementation consists of three execution engines with different data mining algorithms from different data mining techniques (see Table 2). However, the evaluation in this thesis focused only on three data mining algorithms. One of the prospects for improvement is to expand the implementation to include more execution engine and more implemented data mining algorithms. That increases the capability of the prototypical implementation to execute more data mining algorithms and to utilize the capabilities of other execution engines.

The benchmark in this thesis is conducted using 15 datasets. Enlarging the benchmark with more dataset can be beneficial since the selection of the most appropriate execution engine process depends on this benchmark. One of the prospects for improvement is to execute the implemented data mining algorithms with more datasets which perhaps allows for achieving a more accurate selection process.
The concept of this approach is used to automatically select the most appropriate execution engine for executing the needed data mining algorithm. However, the introduced approach can be extended in order to automatically select the most appropriate data mining algorithm from the needed data mining technique. That helps novice users to execute data mining techniques efficiently using the most appropriate data mining algorithm according to the submitted dataset and the user-defined criteria.

The prototypical implementation uses a REST API as an interacting interface with the end user. However, implementing a user-friendly GUI might facilitate the interaction and enhance the user experience. That helps novice users to provide the required input and fine-tuning the parameters of the data mining algorithm with the help of several visual aids.

In conclusion, several research questions still need answers. Therefore, future work can investigate whether the following aspects affect the efficiency of the automatic selection of the most appropriate execution engine: (1) including more criteria other than runtime to the selection process, (2) reducing or eliminating the overhead time that is caused by the data transformation step, (3) adding more execution engines and more data mining algorithms, (4) enlarging the benchmark database with more datasets for increasing the accuracy of the selection process, (5) extending the concept to automatically select the most appropriate data mining algorithm too, (6) implementing user-friendly GUI.
9. References


All websites were last accessed on 4th of November 2018.
10. Appendix

10.1 Appendix 1

Fig. 42: Execution Measures for Datasets of Second Group for Decision Tree
Fig. 43: Execution Measures for Datasets of Second Group for k-Means

Fig. 44: Execution Measures for Datasets of Second Group for FP-Growth
10.2 Appendix 2

The following are request/response example of Cloudera Manager API for retrieving CPU consumption.

Request:

query=SELECT+cpu_percent_across_hosts+
&contentType=application%2Fjson&
from=2018-11-04T23%3A22%3A01.543Z
&to=2018-11-04T23%3A24%3A01.543Z

Response:

{"items" : [ {
  "timeSeries" : [ {
    "metadata" : {
      "metricName" : "cpu_percent_across_hosts",
      "entityName" : "Cluster 1",
      "endTime" : "2018-11-04T23:24:01.543Z",
      "attributes" : { "active" : "true",
      "clusterDisplayName" : "Cluster 1",
      "category" : "CLUSTER",
      "version" : "CDH 5.14.0",
      "entityName" : "$1",
      "clusterName" : "cluster"
    },
    "unitNumerators" : [ "percent" ],
    "unitDenominators" : [ ],
    "expression" : "SELECT cpu_percent_across_hosts",
    "metricCollectionFrequencyMs" : 60000,
    "rollupUsed" : "RAW"
  },
  "data" : [ {
    "value" : 6.8100000000000005,
    "type" : "SAMPLE",
      "sampleValue" : 20.8,
      "count" : 10,
      "min" : 0.2,
      "minTime" : "2018-11-04T23:22:42.000Z",
      "max" : 20.8,
      "mean" : 6.8100000000000005,
      "stdDev" : 8.168700427036521,
      "crossEntityMetadata" : {
        "maxEntityDisplayName" : "master.ipvs.uni-stuttgart.de",
        "minEntityDisplayName" : "slave-06.ipvs.uni-stuttgart.de",
        "numEntities" : 10.0
      }
    },
    "value" : 3.81,
    "type" : "SAMPLE",
      "sampleValue" : 21.5,
      "count" : 10,
      "min" : 0.2,
      "minTime" : "2018-11-04T23:23:42.000Z",
      "max" : 21.5,
      "mean" : 3.81,
      "stdDev" : 6.5330697225729955,
      "crossEntityMetadata" : {
        "maxEntityDisplayName" : "master.ipvs.uni-stuttgart.de",
        "minEntityDisplayName" : "slave-06.ipvs.uni-stuttgart.de",
        "numEntities" : 10.0
      }
    }
  } ]
},
"warnings" : [ ]
  timeSeriesQuery" : "SELECT cpu_percent_across_hosts"
} ]}
Declaration

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

Date and Signature:

06.11.2018