# POLITECNICO DI TORINO

COMPUTER ENGINEERING DEPARTMENT

Master's Degree Course in Data Science


Master's Degree Thesis

## Predicting the pending time of jobs submitted on a High-Performance Computing cluster through a hierarchical classification strategy

**Academic Supervisor:**
Prof.ssa Tania CERQUITELLI

**Company Supervisors:**
Dott.ssa Enrica CAPITELLI
Dott. Vladi Massimo NOSENZO

**Student:**
Fabio CARFI'

ACADEMIC YEAR 2019/2020

# Acknowledgments

Vorrei aprire questo lavoro di tesi con alcune righe per ringraziare le persone che mi sono state vicine durante questo importante capitolo della mia vita. Non è facile, in poche righe, ringraziare tutti quelli che hanno contribuito alla nascita di questo elaborato.
Spero di non dimenticare nessuno ma, se questo dovesse succedere, *Tu* che stai leggendo fammelo sapere e sarò ben felice di ringraziarti di persona.

Rigrazio la *Iveco*, azienda in cui ho svolto il tirocinio alla base di questo lavoro.
Ringrazio la *Dott.ssa Enrica Capitelli*, il *Dott. Vladi Massimo Nosenzo*, *Francesco Vinci* e l'*Ing. Roberto Piccolo*, per avermi fatto sentire fin da subito parte della loro squadra, per l'aiuto fornitomi in questo progetto e per avermi fatto crescere tanto durante l'intero percorso di tirocinio.

Un ringraziamento di cuore alla mia relatrice la *Prof.ssa Tania Cerquitelli* per avermi seguito costantemente, per avermi fornito ottimi consigli senza i quali questo lavoro non avrebbe visto la luce ma, soprattutto, per quell'entusiasmo contagioso con cui svolge il proprio lavoro.

Il ringraziamento più grande va alla mia famiglia. Grazie ai miei genitori per tutti i sacrifici economici e morali e per il sostegno con cui mi hanno accompagnato fino a questo traguardo.
Grazie *Mamma* perchè, nonostante tutto, hai sempre lottato come una leonessa, con grinta e coraggio, senza lasciarmi mai solo e difendendomi da tutto e da tutti.
Grazie *Papà* per esser stato forte per tutti, per esserci stato sempre, con i tuoi silenzi e le tue battute nere.
Grazie *Fede*, sorellona mia. Grazie a te ho iniziato questo percorso, quasi per sfida. Sei sempre stata la più saggia tra i due, quella con i consigli giusti al momento giusto. Nei momenti bui eri tu a farti carico di tutto. Forse non ho saputo esser bravo quanto te nel dare consigli o anche solo nell'esserci, in silenzio, ad offrire una spalla su cui piangere. Scusami per non averlo capito prima. Ti prometto che sarò pronto ad aiutarti ogni volta che ne avrai bisogno.
Grazie ai miei *Nonni*, *Angelo*, *Pina*, *Carmelo* e *Lina*, a *Zia Giovanna*, *Zia Patrizia* e *Zia Antonella* che, nonostante la lontananza, sono sempre presenti e mi riempiono d'amore incondizionato, viziandomi e straviziandomi nei pochi giorni dell'anno in cui si sta sotto lo stesso tetto.

Per ultimi, ma non per importanza, vorrei ringraziare immensamente i miei amici.
Grazie *Matteo*, *Miry*, *Giulia*, *Ricky* e tutto il gruppo dei *Los Amigos* per esser stati parte di momenti di gioia e festa, ma anche per aver sopportato momenti di sconforto e paura.
Grazie *Faby*, *Simo* e *Davide* per aver fatto parte della mia vita, nei momenti più belli ma anche in quelli più brutti, spronandomi a superare tutti gli ostacoli che la vita mi metteva davanti.
Grazie *Stefy*, *Jenny*, e *Ale* per le serate di totale divertimento passate insieme e per avermi sempre ascoltato, dandomi consigli preziosi.
Grazie *Manfred*, *Bartolo*, *Peppe*, *Giorgio* e tutti gli amici *sciclitani* per la spensieratezza dei momenti trascorsi insieme e per l'affetto dimostratomi.
Vi voglio bene e vi auguro solo il meglio.

*Fabio*

# Abstract

In the context of heavy simulations, which need, therefore, to be performed on HPC architectures to have more computing power than the one provided by a normal PC, since the resources these systems can offer are also limited, it is useful to try to understand how much time passes between the moment in which the job is subdued and the one in which it starts the execution. This is because with this knowledge it would be possible, in the future, to design processes that can define the execution time of simulations, in order to use this information to create procedures that can optimize the resources requests, drastically reducing the waiting times of the results.
The objective of this thesis project, carried out at the Iveco headquarter (Italy) under the supervision of the Politecnico di Torino, is therefore to succeed to implement a prediction system able to estimating the time that a job will have to wait before the necessary hardware resources are supplied for its execution.
To do this, an analysis of the available data was initially carried out to increase its descriptive capacity and to provide guidance on how to proceed.
Later, feature engineering techniques were applied to extract other information to better describe the analysis situation. In fact, in this case, variables were created to try to characterize the context of the HPC cluster at the time of submission of the job, data that were not contained within the dataset available.
Once the information and variables deemed necessary for the execution of this activity have been obtained, we moved to an experimental phase in which supervised machine learning algorithms were used to perform various predictions, modifying the classes, their number and also the quantity of models used to obtain a single result. In fact, at the beginning, multi-class classification experiments were carried out with intervals of the same frequency and others with custom ranges. At a time when it was found these tests yielded low results, binary classifications were analysed, i.e. only using two prediction intervals, to study the behaviour of the algorithms and to find, therefore, those thresholds that allow to obtain appreciable results. From the knowledge acquired through these two types of experiments, it was finally decided to try a hierarchical approach in which several models were used in cascade based on the prediction results obtained in previous levels.
Doing so, a solution was found which presented better results compared to those of the various tests executed. However, the values of the indices obtained in the prediction of the classes show that the system is still improvable. Therefore, the work done during this period will continue with the aim of further improving this system, modifying it and possibly integrating it with other techniques to be analysed in the future.

# Contents

# List of Figures

# List of Tables

# 1. Introduction

The Machine Learning (ML), known also as automatic learning, is a sub-field of Artificial Intelligence that uses algorithms to automatically extract models from the data available. Systems composed by trained algorithm, data and operating parameters (used to adapt the algorithm to the specific environment in the best way possible) are called ***models***. These are very effective in identifying common features or some trends inside the huge amount of data, taking into account a number of variables that no human being would be able to evaluate or notice, variables subsequently inserted within a single functioning algorithm.
A machine learning model doesn't follow explicit and predefined rules, but constantly learns by its experience. This is possible thanks to the availability of a large amount of data which allows you to improve ML techniques that are not only used to train models but, mostly, to extract hidden patterns within the data used to do the training.

The term Machine Learning was coined for the first time in 1959 by Arthur Lee Samuel, American pioneer scientist in Artificial Intelligence field, even if, to date, the most accredited definition by the scientific community is the one provided by another American, Tom Michael Mitchell, director of the Machine Learning department of Carnegie Mellon University: "*A program is said to learn from a certain experience E, with respect to a class of tasks T obtaining a performance P, if its performance in carrying out the tasks T, measured by the performance P, improves with experience E*".

Unlike systems based on predefined rules, which will perform the activities each time in the same way, the performance of an ML system can improve through learning, using huge amount of data as different as possible. From that it can be deduced that results of an ML model are never sure, indeed they are always associated with an accuracy percentage of the model. Based on the application filed, it's up to the user of these algorithms the minimum percentage of confidence to be used [6].

A fundamental aspect in machine learning is to have the starting database as good as possible. With database, or dataset, we mean a series of information collected over time and saved inside files and/or databases (DB) with an appropriate shape and structure. However, data or part of it is not always available immediately and is for this that starting information is analyzed in depth to extract missing data, but also to delete that information (called also outliers) that could compromise the algorithm's learning.

If normally the programmers are called to define deterministic functions that, given a certain input, return always the same output, therefore following the mathematical representation $y = f(x)$, with machine learning this doesn't happen. In fact, in this field are used general mathematical and statistical algorithms which, exposed to a series of data during the training phase and passing through a second phase of evaluation of the results with some parameters optimizations, create autonomously the function. This function, not always known or available for the programmer, is able to predict, given a series of input data, the most probable value of $y$ associated also with the accuracy of the prediction made. In other words, function $f(x)$ is automatically derived without the need for the programmer to define the rules [25].
This way of acting completely change the programming idea, moving from a traditional and

deterministic style, in which programs were written to automatize tasks, to another more automatic but probabilistic. In short, ML create the program (*i.e. the model*) that adapt to the training data and through the knowledge gained, considering also the input data, returns the most accurate probable value.

Machine Learning is made up of three macro-categories:

1. ***Supervised Learning***
2. ***Unsupervised Learning***
3. ***Reinforcement Learning***

The first one group all the algorithms whose purpose is to learn from data, whose output, also called label, is already known from the beginning. Doing so, the model adapts to these and try to predict in the best possible way the data label, to which it is not yet associated.
Differently from the supervised learning, unsupervised includes that have to work with data whose structure or label you want to predict is unknown. Their aim is to explore data structure to get the most important information enclosed within it.
As for reinforcement learning, the purpose of its algorithms is to create a system, also called agent, that improves its performance, based on interaction with the environment. Considering that the data, gathered through these interactions, also contains some "*reward signals*", it's possible to consider this category as a subset of the supervised. A classic example of reinforcement learning is the chess game where, in this case, the agent takes decisions based on the state of the chessboard and the reward can be defined as the victory or the defeat at the end of the game. Broadly speaking, the agent will always try to maximize the rewards through a series of interactions with the environment. [47]

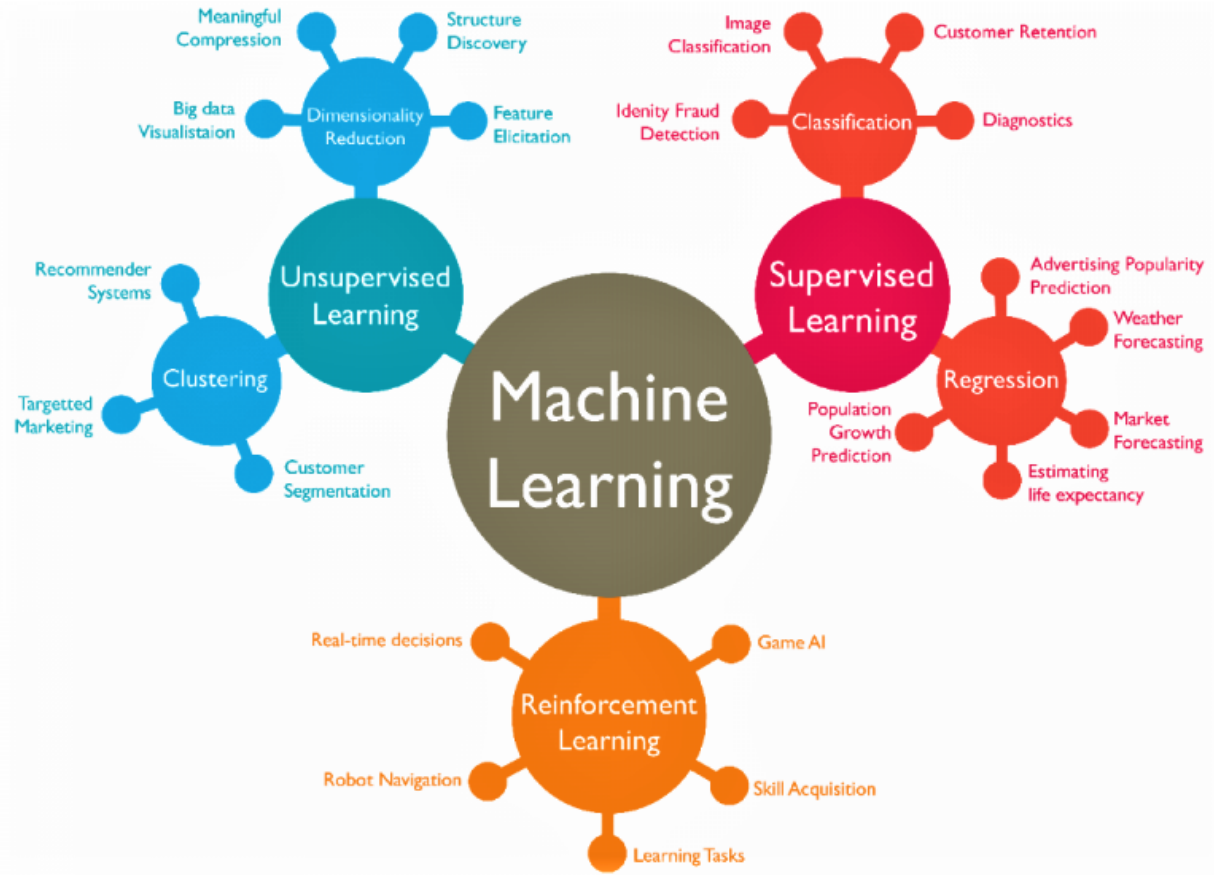



**Figure 1.1.** Macrocategories of machine learning algorithms with some usages [25].

Considering these macro-categories, the first two were widely used in the context of this thesis and the last one will be the subject of a future research grant to continue in the implementation and improvement of the model, overall objective of the project.

Last machine learning improvements have made its techniques more flexible and resilient about being able to be applied to various real-world scenarios, ranging from extraordinary to mundane chases. One need only thinking to machine learning applied to spam filtering which, every day, enhance user experience of millions of users, blocking unwanted or even dangerous emails.
There are many other examples in real-world where these automatic learning rules are applied. Automatic translation of languages is now widely used in the world, with some couple of languages that work better than others. Face recognition is broadly employed for unlocking most of the mobile phones on the market, but also to identify people through security images, thing that is creating privacy issues. In the field of medicine, these techniques are starting to be used to help, not replace, doctors and radiologists in the classification of diagnostic images and to detect certain types of abnormalities [6, 25].

Large companies, including Google, use these artificial intelligence algorithms to improve the usage of available resources and this leads to a reduction of response times and therefore of obtaining requested results. It's in this context that the subject of this thesis is classified: we want to try to optimize, as much as possible, the resources (intended, as explained in section 1.2, as number of CPU and computation licences ) of the High Performance Computing Cluster (HPCC, treated in section 1.3.1) to get the best trade-off between response times and resources used for the operations done.

## 1.1 The company

**Iveco** (Industrial Vehicles Corporation) is an industrial vehicle manufacturing company present all over the world and based in Turin. It was founded in 1975 from the union of five different companies:

1. Fiat Industrial Vehicles (Italy)
2. OM (Italy)
3. Lancia Special Vehicles (Italy)
4. Magirus-Deutz (Germany)
5. Unic (France)

These five manufactures joint their skills to create one of the major players in the global transport sector. Today, Iveco is entirely controlled by CNH Industrial and is an international leader in the development, manufacturing and servicing of a vast range of light, medium and heavy commercial vehicles. It also manufactures city and intercity buses and special vehicles for military uses, for civil protection and for specific missions like, for example, fire-fighting and off-roads.
It has 27 production plants, 6 research centers and 5.000 stores and points for assistance distributed in Europe, China, Russia, Australia, Africa, Argentine and Brazil. It counts more than 25.000 employees all around the world. The global production reaches about 150.000 commercial vehicles per year with a turnover of about 10 billion Euro.


**Figure 1.2.** Iveco commercial vehicles gamma [27].

## 1.2 Terminology

It's important, before starting the explanation of the work done, to introduce some fundamental terminology to fully understand what is explained in this document:

- ***Discipline***: it represents an application field of simulations of the same type executable by the company users. Considering Iveco examples of disciplines are *Acoustic*, *Computation Fluid Dynamic* or *CFD*, *Multi-body*, *Safety*, *Structural* and *Vibration*.
- ***Job***: it's the set of complex operations executed on the HPC computational nodes. They can be of different type, considering also the discipline they belong to.
- ***Queue***: it's a group of HPC computational nodes assigned for the execution of specific jobs, depending also on the need of the job itself. In the subsection 1.3.1 will be explained how HPC nodes are divided for each queue, also specifying the reasons for the choices made.
- ***Software Name***: it's the name of the application, specified by the user, in charge to execute the job and return some results.
- ***Resources***: both concern hardware and software side. Indeed, they are the number of CPU or core needed or requested but also licences of a specific software required to execute jobs.
- ***Pending Time***: it's the time that a job wait until resources it needs will be free and are assigned to it by PBS.
- ***Run Time***: it's the effective time that a job holds the resources given by PBS. To be more clear, it's the time that the software needs to return some results, independently if they are positive or negative (some sort of errors or if the job is stopped by the user that launched it).

Before going further, there is still the need to clarify that the jobs can be of two different types: *explicit* and *implicit*. Both are a possible type to resolve simulation problems and differ in the approach to time increment.

Solvers needed to execute implicit tasks use "step by step" algorithms. Here an appropriate convergence criterion allows continuing or not the analysis reducing the time increment, depending on the accuracy of the results at the end of each step. The problem with these jobs is that each time increment have to converge and this can require too much time.
On the other hand we have the explicit jobs which don't have any non-convergent problems at each increment because the time increment is defined at the beginning and remains constant during the computation. It's important to notice that, to receive an accurate solution, the time increment must be minimal. Therefore, typically, explicit tasks are faster, considering models characterized by many degree of freedom. As well as they haven't problems with convergence, they have also the possibility to overcome issues related to non-linearity of contact and collision.

## 1.3 Application area

Within the framework of automotive industries, the usage of HPC for the resolution of complex physical simulations has allowed a great improvement of production processes. These technologies allow avoiding high computation time and the prototyping need of every vehicular component. Thus, helps enormously on the resolution of problems like, as example, the ones of Computation Fluid Dynamic, also called with the acronym of CFD, or also some of Finite Element Analysis, abbreviated as FEA, but also many other types.

Considering that inside this document will be covered the implementation of a machine learning algorithm whose purpose is to improve the performance of Iveco's HPC cluster, it's fundamental to describe the hardware and software components which form the system to be analysed. Regarding that, data extracted from a code manager called PBS (dealt with in section 1.3.2) were processed. This software component takes charge of the jobs that the user wants to execute and, furthermore, manage the sorting of hardware resources between the jobs waiting to be scheduled.

### 1.3.1 HPC

The High Performance Computing expression, in jargon HPC, is used to describe the use of high power systems realized combining a large number of computational servers, also called compute nodes. These structures are employed for the processing of a big amount of data, but also for the execution of complex computation like mathematical calculation or even crash and stress test simulations (types of job launched on the HPC taken into account in this project). To give a more explicit representation of how much powerful these systems are, it is possible to make a practical example: a normal PC executes about three billion ($3 \cdot 10^9$) operations per second while an HPC can make like $10^{24}$ operations per seconds.
Within these structures, therefore, parallel processing is widely used, that is the possibility that each node has to execute different tasks in parallel with others execute on other nodes. This makes possible to perform tasks requested by the user in the most advanced and efficient way, providing a result as soon as possible [13].

The set of computational nodes is also called HPCC, or High Performance Computing Cluster, and usually is interconnected with a Data Cluster where the calculation outputs of the different jobs executed are saved. Obviously, each node within the cluster must work optimally because as long as even one of them has a problem to compromise the performance of the entire HPCC [38].

In its most simplified form, an HPC cluster is made up of one front-end node, used by the user as interface for accessing to the computational part of the system, and from the set of numerous compute nodes. Normally, the front-end node can be realized through a virtual machine or a web interface while computational nodes are physical hosts to boost performances as much as possible. In more complex systems, however, there is the possibility of clustering the front-end to balance the user's access load [43].

Depending on the chosen architecture there are two traditional approaches for its realization:

- ***Cluster Computing***: it is the simplest approach and combine the computational capacity of a set of computers that can also work independently.
- ***Massive Parallel Processing***: defined also MPP, it combines the computational power of hundreds of processors that, linked together, are placed within the same physical system.

Commercial solutions tend to prefer much more the first approach to achieve high density systems in which to host large amounts of server rank connected together. An architecture example is reported in the figure 1.3 representing part of the CERN system.

**Figure 1.3.** HPCC at the CERN of Ginevra [13].

In the case of Iveco's HPC cluster resources have been grouped together in sub-cluster (that within this document are defined as "queues"), specific for certain solvers, to try to minimize conflicts between jobs during the acquisition of resources. Indeed, each queue has a defined number of nodes associated which implies a different availability of cores and memory disk. Inside table 1.1 are reported all the values to give a complete description of the system.

**Table 1.1.** Distribution of Iveco's HPC cluster resources

| Queue | Number of nodes | Cores per node | Total cores | Memory per node [TB] | RAM per Core [GB] |
|---|---|---|---|---|---|
| CFD | 15 | 16 | 240 | 0.999 | 128 |
| FEA | 14 | 16 | 224 | 0.999 | 128 |
| Front-end | 2 | 12 | 24 | 6.4 | 64 |
| Highperfio | 1 | 12 | 12 | 2.2 | 256 |
| Implicit | 3 | 16 | 48 | 2.2 | 256 |
| Mesh | 2 | 16 | 32 | 0.2 | 256 |
| Veiprod | 1 | 16 | 16 | 2.8 | 128 |

The amount of associated resources for each queue depend on the need of jobs that will be launched above, thus trying to minimize the burden of hardware errors as a memory overflow for example. To avoid this type of problems a further rule has been added for the queue called highperfio, where it is limited to two the number of jobs which can be executed at the same time. The two nodes of front-end, normally, are not used for the execution of the jobs but contain, one at a time, all the users home directories. Thus, only one of them will hold all the folders. In fact, the second node is used just in case the other one has some technical issue. Doing so, there will be simply the need of changing the identification of the reference node, without users realizing it.

### 1.3.2 PBS

The Portable Batch System, also abbreviated with the acronym PBS, is a batch job and computer system resource management package. It accepts batch job, shell script and control attributes. Preserve and protect the jobs until they will be executed and, lastly, deliver output back to the submitter.

It is composed by four principal components [52]:

1. *Commands*: PBS supplies both command line operations and a graphical interface. These are used to execute, monitor, modify and also delete jobs. Commands only can be run if there are the necessary permissions to do so (user commands can be run by anyone while other commands can be run only by an administrator or an operator).

2. *Job Server*: it is the central focus of PBS, the reference server with which it communicates via an IP network. The server's main function is to provide the basic batch services such as the reception, creation, modification and execution of batch job, but also protection from possible system crashes.

3. *Job Executor*: it is the daemon, called also *pbs_mom*, which actually places the job into execution and return the output to the user (if requested by the server).

4. *Job Scheduler*: it is another daemon which contains the site's policy controlling which job is run and where and when it is run. Because each site has its own ideas about what is a good or effective policy, PBS allows each site to create its own Scheduler. The scheduler can communicate with the various pbs_mom to learn about the state of the system resources and with the Server to learn about the availability of jobs to execute.

The main functions that PBS provides to the users are the addition and removal of a job but also the opportunity to monitory its execution in a way that is possible to modify or block it and then restart it, changing the number of resources requested. In addition to the services available to each single user for the management of jobs, PBS autonomously execute the research of accessible nodes to launch the job, reinforces the code policies in case there is the need of insert precise rules and offer also access control functions for possible statistical analysis regarding the usage of the HPC and the system in general.

Inside the thesis project PBS is used to manage the submission of jobs on the specific HPC queues, handling the resources for each one of them and deciding on what nodes launch the execution as soon as the requested resources will be available.

### 1.3.3 How HPC cluster and PBS work together

In this section will be treated and analysed briefly how the two components, discussed previously (see section 1.3.1 for what is about HPC and section 1.3.2 for what is about PBS), work together to offer the computing and simulation services to users.

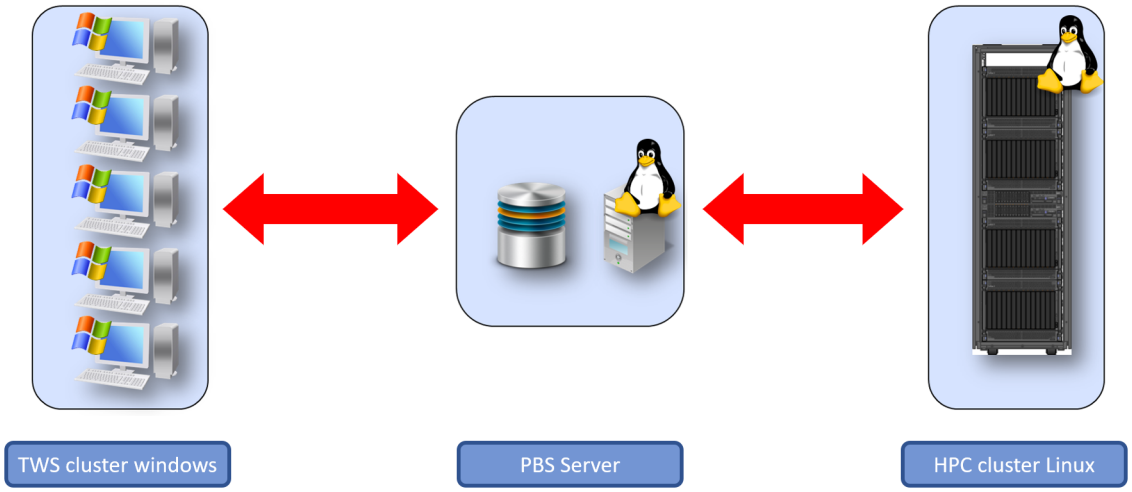



**Figure 1.4.** Representation of the architecture used by Iveco to submit jobs.

As it is possible to see in figure 1.4, the Iveco architecture of the job submission system consists of three main parts. The first one, i.e. the TWS cluster Windows, it is the set of all the technical workstation (from this the acronym TWS) of the users based on Windows operating system. The second component is the PBS server that handles both the transfer of input files from Windows system to Linux one and the steps concerning the execution of the job itself. Lastly there is the HPC cluster that, under Linux environment, group all the computational nodes to perform the required operations. The two clusters are logically separated but linked through the same PBS server.
The operations that the user can perform are three and usually in the following order:

1. *Stage In*: it is the transfer, by the user, of the input files necessary for the execution of the job from TWS Windows to his own home directory inside the front-end nodes.

2. *Job execution via command line*: the user submit the job through the command line specifying the solver to be asked for the execution, the queue (i.e. the set of nodes of the HPC) that will take care of it and also the name of input files from which take the necessary information.

3. *Stage Out*: it is the opposite of the Stage In. It is the transfer on the user personal machine of the files, containing the result of the simulation, present on the front-end nodes to not occupy all their dedicated memory space.

So, as first operation, the user performs a stage in of the input files and run the job via command line. At this point it is the PBS server that take the situation control and executes the specific custom script of the solver previously chosen. After that, the control is passed to the scheduler to firstly make a check on the availability of requested resources. In case these resources can be provided the information received are transferred to compute nodes to execute the job on pbs_mom installed above. Otherwise, the job is queued until resources will be released. At the

end, when the job execution is completed (whether positive or negative), the results are written on files that later are transferred on the personal home directory of the user. From here on, there is the possibility to perform a stage out so there will be the possibility to analyse and, also, store the obtained solutions.

## 1.4 Thesis objective and document organization

Considering the data obtained from the interactions between HPC cluster and PBS, described in section 1.3.3, the objective of this thesis is to define a prediction model that can compute, in the best possible way, the waiting time of each job submitted on the HPC cluster Iveco. To do this, data-driven techniques will be used firstly to understand the internal structures of the data and then to do the training and testing of the model in order to find the solution that return the best accuracy values.
In order to tell the experience lived in the company and describe the various activities carried out, this document is organized in five chapters distributed as follows:

1. ***Introduction***: in this chapter it is provided some basic knowledge to help understand the document. In fact, firstly some information about machine learning at high-level are given, defining the area of use and possible applications. Additionally, some explanations about applied terminology are offered for a better understanding of the topics covered in this thesis. Thereafter, it is provided the main notions regarding the HPC cluster (what it is, what it is for and how it is organized), specifying also the characteristics of the one used by the company, and PBS, explaining also how these two components work together.

2. ***State of the Art***: in this chapter the concept of KDD (Knowledge Discovery in Database), extremely useful process in the machine learning field, will be discussed quite in depth. Subsequently, the various statistical techniques and graphical representations used to understand the data (analysing the internal structures) will be explained. The chapter closes with two detailed sections on machine learning and, more precisely, on the types of algorithms concerning Supervised and Unsupervised Learning (Classification and Regression algorithms for the first one and Clustering and Dimensionality Reduction algorithms for the second one), to understand how these work and how they can return certain values.

3. ***Case study and proposed methodologies***: in this chapter, the initial dataset is described. In the second section, data cleaning and feature engineering operations are analysed; the last ones are used to extrapolate new variables useful for the pending time prediction process. The third section deals with data exploration analysis applied to accounting and context data, to better understand their internal structures. Finally, in the last section the choices made in the selection of classification algorithms are justified and the structure of the best solution is presented.

4. ***Experimental results***: this chapter reports all the results of experiments conducted to find out the best possible prediction solution. Firstly, the results obtained by regression techniques and, then, those obtained by classification ones are discussed.

5. ***Conclusions and Future Work***: in this chapter the results of the experiments carried out are analysed and, subsequently, the future tasks with which we will proceed to work for the project are introduced.

# 2. State of the Art

In this chapter I want to introduce the main concepts to be able to fully understand the work done told within this document. First of all, the notions about Data Mining and KDD (Knowledge Discovery in Database) which groups the various stages to get a complete knowledge of the data taken into account. Afterwards, to understand the structure and the most important information hidden within them, all the statistical analysis carried out to achieve this goal will be treated. Lastly, will be analysed two of the macro categories that make up the Machine Learning, the algorithms of Supervised Learning and those of Unsupervised Learning, essential elements for the implementation of the procedure in order to optimize the use of hardware and software resources of Iveco HPCC.

## 2.1 Data Mining

Advances in digital data acquisition, distribution, recovery and storage technologies have led to the creation of huge databases. One of the main challenges to be faced is to transform these immense data collections, rapidly expanding, into accessible and usable knowledge that can be used later for decision-making support. Attempts to tackle this challenge have brought together researchers from different fields, such as statistics, machine learning, databases and many others, creating a new research area called Data Mining. This term is generally used to indicate one of the fundamental steps of the KDD (Knowledge Discovery in Databases). This is an iterative process, because each step can correct the possible errors made in the previous ones, but also interactive because it is left to the user the limitation of the workload carried out by the system, in order to deal only with the relevant aspects within the research [48].

With the term Data Mining are represented all those analysis techniques that allow to automatically extract important information hidden within datasets. The application of statistical analysis and different search algorithms produces a certain amount of models developed through data, always taking into account acceptable limits of computational efficiency [48]. The aim is to identify clear and meaningful patterns, which are defined trough the usage of different algorithms.
Data Mining is usually divided into two main categories:

- *PREDICTIVE*: focuses on particular properties or attributes, called target attributes or object, of the data. Considering a number of instances such that these attributes are known, the goal is to create a "predictive" model which is able to determine the output value for a new tuple [4, 48]. In case the attribute is nominal, a classification model will be created, while if it is numerical, a regression model will be obtained. Examples of these models are the SVM (Support Vector Machines) and also the neural networks.

- *DESCRIPTIVE*: in this case, differently from the previous one, there is no specific target variable. What we are looking for is a way to describe the important systematic rules present within the data [4, 48]. In this way, can be created models able to express the existing associations between the attributes, clusters to describe similar data subsets or even probabilistic models in order to represent the probabilistic dependence between the attributes. Clustering algorithms are part of this category.

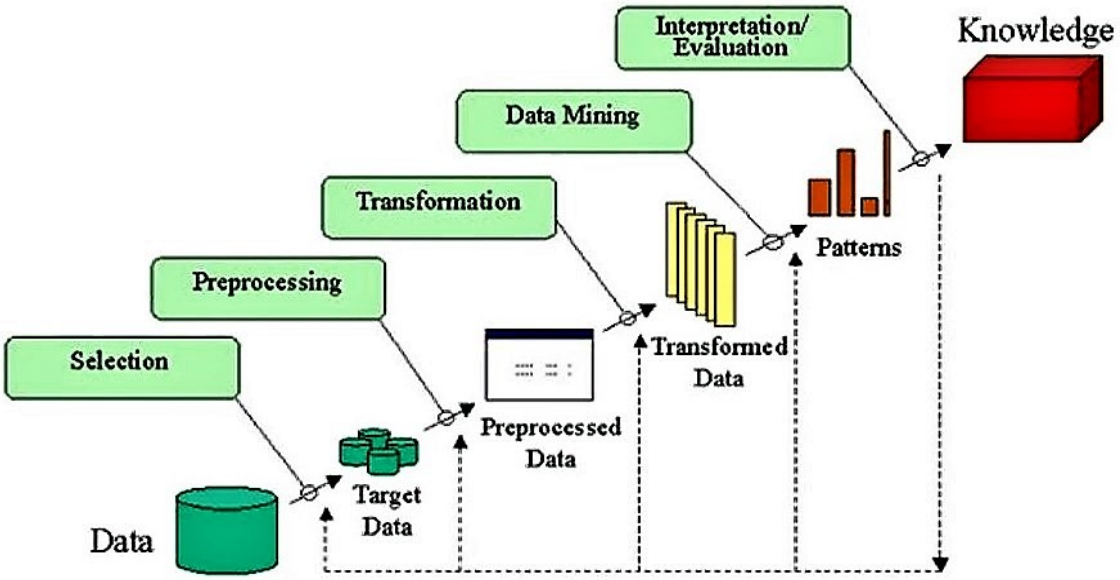



**Figure 2.1.** Representation of the KDD process with all its main steps [3].

The term KDD was coined to emphasize that "knowledge" is the end product of a data-driven discovery.
The KDD focuses on the research for understandable patterns useful for the interpretation of data, thus enhancing the work done with large amounts of real-world information. This process has much in common with statistics, especially as regards the methods used for exploratory data analysis. The knowledge of this field allows us to quantify the intrinsic uncertainty associated with the results when trying to infer general patterns, derived from a subset of data, to the entire population. To ensure that this uncertainty is minimized as much as possible, thus increasing the accuracy of the associations made, all steps are repeated in several operations.

This search process has been defined as a "non-trivial process of identifying valid, novel, potentially useful and ultimately understandable patterns in data" [18]. In this statement the data is a set of cases contained within the databases while the patterns are specific rules that can be applied to that subset of data. As for the adjectives used, instead, with "non-trivial" it wants to indicate the necessary involvement of researches and inferences, meaning that it is not a simple calculation of predefined quantities as could be the one about the mean value of a set of numbers. Discovered patterns should be "valid" on new data with some degree of accuracy. In addition, it is also required that these must be "innovative" and "potentially useful" to bring some benefits to users or subsequent operations. Lastly, it is required that the results obtained are "understandable", if not immediately after some post-processing operations at least, so that they can be really used
So, the KDD "is the process that, starting from the data inside the databases, along with any selection, preprocessing, subsampling and transformation required, and through the application of data mining algorithms to extract all the patterns of interest available, allows evaluating which of these models can really be considered as 'knowledge'"[18].

Data mining performance depends on both the format and the accessibility of the data. In fact, usually, the attributes that influence the accuracy of the result are selected and/or calculated first, in the way of filling the differences that there are between the format of the data as they are saved at the source and the one required by the algorithms. This step is the main key to the success of these techniques, all enclosed within what is later called preprocessing.

In addition to this, there are other phases that make up the KDD, as can be seen in figure 2.1, analysed below:

1. *Selection*: in this step the main purpose is to select the useful data, compared to the information we want to search for, from the various sources available (such as databases). The selection can be made either horizontally, which means going to choose only tuples that meet specific conditions, or vertically, thus deciding to consider only some of the available variables.

2. *Preprocessing*: is a phase of fundamental importance within the process because it allows to increase the descriptive power of the results obtained from the selection, going to improve the final accuracy of the extracted patterns. It takes into account both data cleaning operations, designed to clean data from possible outliers (by making a clear distinction between these and noise), which could compromise the final result of the process, as well as data integration, to fix possible conflicts within the data after the integration from various sources but also to incorporate necessary data which, however, are not present but can be computed (for example, you can find ways to manage outliers without necessarily deleting them or to manage existing missing values).

3. *Transformation*: it brings together all the necessary operations to reorganize the data before they are fed into data mining algorithms for pattern extraction. In practice, we go to build datasets starting from the semantic aspect, based on the previous data collection, going then to work on the syntactic one, which apply some transformations (as, for example, the dimensionality reduction) necessary to fit the algorithm to the data. This emphasizes the main features depending on the final expected results.

4. *Data Mining*: is the step in which some of the available data mining methods are evaluated, trying to decide which one suits in the best way to the data, and, subsequently, where the extraction of the patterns of interest is performed.

5. *Interpretation*: is the final stage of the process in which the obtained results are assessed taking into account three main factors. First of all, we need to make sure that everything has been technically successful. Secondly, it is necessary to examine the validity of the solutions obtained from the point of view of data mining. Finally, it is fundamental to verify that all the necessary conditions have been adequately considered.
   In order to perform this analysis of the results in the best way, usually, graphic representations are also created, able to visualize on specific graphs the models extracted from the data, drawing its conclusions in as much detail as possible but also as simply as possible.

At the end of the entire process, all the acquired knowledge is usually incorporated into the system in order to improve its performance [4, 18, 48].

Nowadays, data mining techniques are widely employed in many contexts of the economic and digital world. Examples of this are the analysis of texts used for the extraction of useful information within the documents, the researches carried out on clinical data, the offer of services to users through the usage of their location obtained by the GPS devices or even the comprehension of their behaviour on websites to understand their interests and offer customized solutions (think about what happens, for example on e-commerce sites), the analysis of data from social networks and many other different situations.

## 2.2 Statistical Analysis

The first step of all the statistical analysis consists in organizing and summarizing the data which are composed by the information gathered on the statistical units that make up the sample.

In addition to supporting the assumptions of the initial work, the exploratory analysis allows to guide the study towards the formulation of new hypothesis and is useful both to verify qualitatively the existence of appropriate assumptions on the considered variables and also to suggest any parametric models to be used for the inferential purposes.

In this section will be touched both the graphical representations, widely used to make incisive and rapid reading of the statistical information (especially in the presence of large databases), and all the tests executed on the single variables values to extract as much information as possible useful for understanding the data structure.

### 2.2.1 Graphical Representations

The human eye, or more precisely the human brain, is remarkably adept at handling visual information in large quantities and complex formats. This goal is achieved by taking shortcuts, making assumptions and interpreting what we see in relation to our experience. This way of treating data, however, can lead to a lack of understanding of what we see, even losing some very significant details within the analysis [7, 12, 53].

The purpose of the graphical presentation of scientific data is to communicate, in the most efficient and immediate way possible, the information contained inside. This is important both for how we tell the results to third parties and for our comprehension and analysis of the data. In fact, some patterns, which could go unnoticed if only statistical analysis were used, are often revealed through graphs [7, 12, 53].

Mainly in this section will be discussed some types of graphs that can be used, within also in this thesis project, for the extrapolation of information from the available data, trying to explain also in general when these representations should be used.
The visual graphs available to us are very numerous and each one highlights some aspects while shadowing others. Given the effectiveness and immediacy of these diagrams to communicate information, it is advisable to carefully choose the type of representation that better fits the message intended to be conveyed or the feature that must be highlighted.

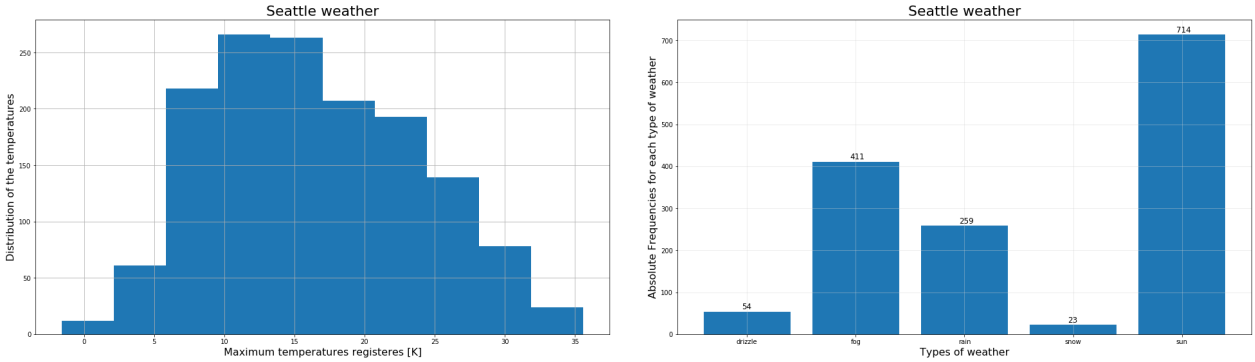
In the case of continuous quantitative variables it is a good idea to use **histograms** while for categorical variables **bar charts** are used. Both are bar diagrams that allow to visualize the variability of a phenomenon, making it easier to interpret the data. These two types of graphs are composed of rectangles built on a Cartesian plane in which the modes are usually reported on the x-axis (or the possible values that the variable in question can take) while on the y-axis the frequencies (or the absolute number of cases). To build these two types of diagrams four elements are needed:

1. *Class*: is the single bar linked to a specific mode.

2. *Class limits*: are the minimum and maximum values of each class.

3. *Class width*: is the interval between class limits (in the case of bar charts this value is the same for each class).

4. *Frequency*: is given by the number of observations belonging to specific classes.




(a) Example of Histogram

(b) Example of Bar Chart

**Figure 2.2.** Both examples are created using the seattle weather dataset.

These graphs are extremely useful in case there is the need to study variability, through the analysis of the shape of the distribution, the average value and the spreading. In addition, they make it possible to assess the extent of the discrepancies between the expected results and the ones actually obtained, allowing in this case to identify the possible reasons of the deviations [11, 16, 39].

To describe a distribution of a data sample through the quantiles is used the **box plot**, a type of graph invented by John Tukey that allows to view at a single glance all the available elements.

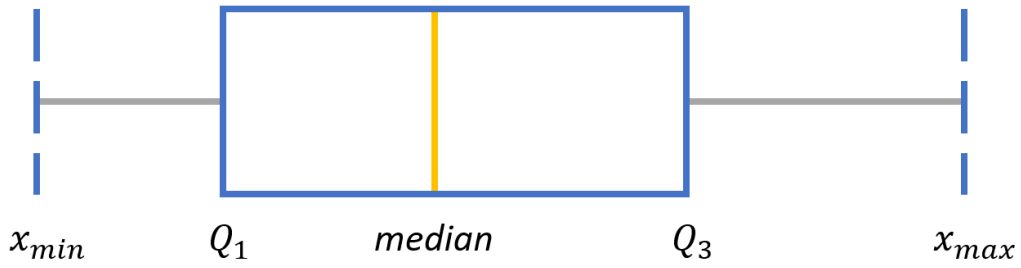



**Figure 2.3.** Graphic explanation of the main components of a boxplot.

The box ends are defined by the first and third quartiles ($Q_1$ and $Q_3$ respectively) while the inner line is the median. The difference given by $Q_3 \breve{} Q_1$, also called interquartile distance (IQR), coincides with the amplitude of the range in which at least the 50% of the data is found. Whiskers correspond to the minimum and maximum values of the data in case there in no interest in identifying the presence of possible outliers. Otherwise, the values of the two whiskers are calculated taking into account the IQR through the following formulas:

$$x_{min} = Q_1 - 1.5 \cdot IQR \tag{2.1}$$

$$x_{max} = Q_3 + 1.5 \cdot IQR \tag{2.2}$$

In this case, values below $x_{min}$ and those above $x_{max}$ are considered outliers and are reported as separate individual points.

This type of graphical representation allows to immediately visualize the distribution of the variables, highlighting the measure of the dispersion, the presence of any anomalous values and its symmetry or asymmetry. A distribution, therefore, can be defined as [11, 16, 39]:

- Symmetric in case the arithmetic mean coincides with the median, thus allowing in this case to deduce the value of the arithmetic mean directly from the graph.
- Asymmetric in case the arithmetic mean is less or greater than the median (depending on whether it is negative or positive asymmetry respectively) and the two whiskers are of different sizes.

In case the modalities of a variable of interest are numerous, the **Pareto diagram** is particularly useful. This type of graph is also composed of vertical bars in which all the values of the variable taken in analysis appear in descending order with respect to the frequencies, all combined with a cumulative polygon in the same scale.

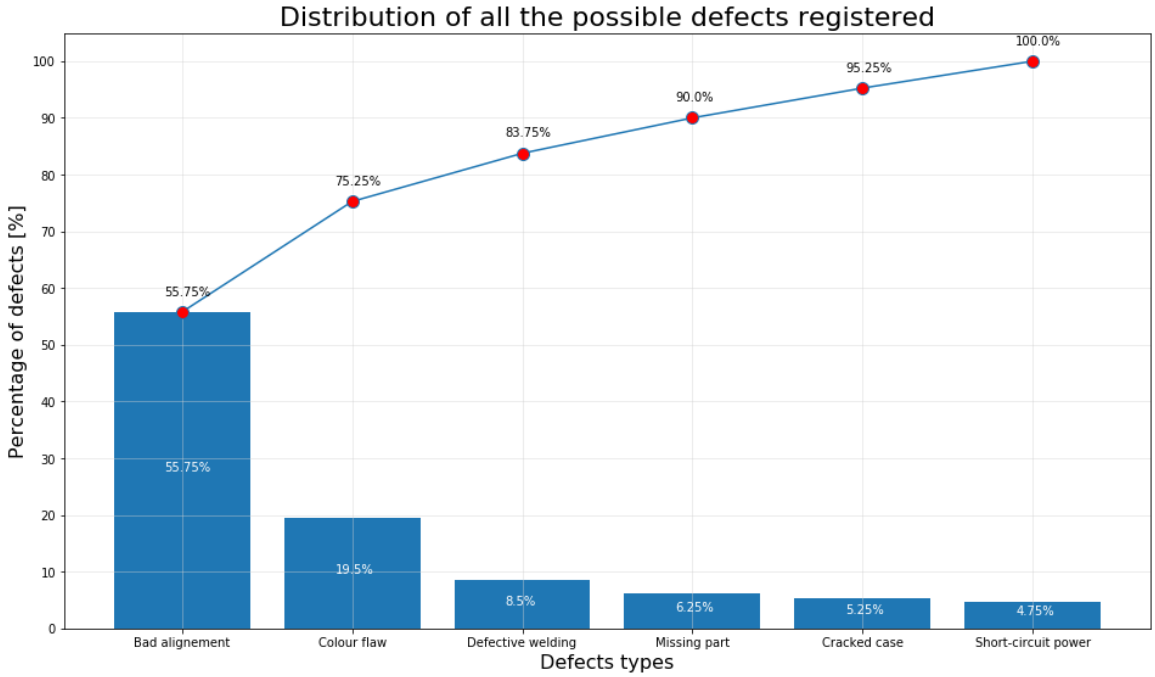



**Figure 2.4.** Graphic example of a possible pareto diagram created by some data involving defection of an assembly line.

The main advantage of these diagrams is the immediate ability to visually separate the modes associated with a frequency (absolute or relative) higher than those less represented in the data. To put it simply, the few relevant cases are clearly separated from the numerous irrelevant ones. Doing so, it clearly catches the eye what are the values that need to be focused on.
The Pareto Principle, also known as the 80/20 law, can be applied to the information obtained from this graph. This principle can be summarized by the following statement: "on large numbers, most effects are due to a minority of causes". In summary, in the event that there is a high amount of data, 80% of the problems that have to be solved is usually generated only by the first 20% of the values of the variable taken in analysis. So, ultimately, we want to say that most of the problems can be solved by going to analyse only a small part of the sample taken into consideration [11, 16, 39].

The above-listed representations are usually used to perform analysis on a single variable of interest at a time. In case, instead, there is the need to identify any relationship between two variables measured with the metric scales (discrete or continuous) we use the **scatter plots**.

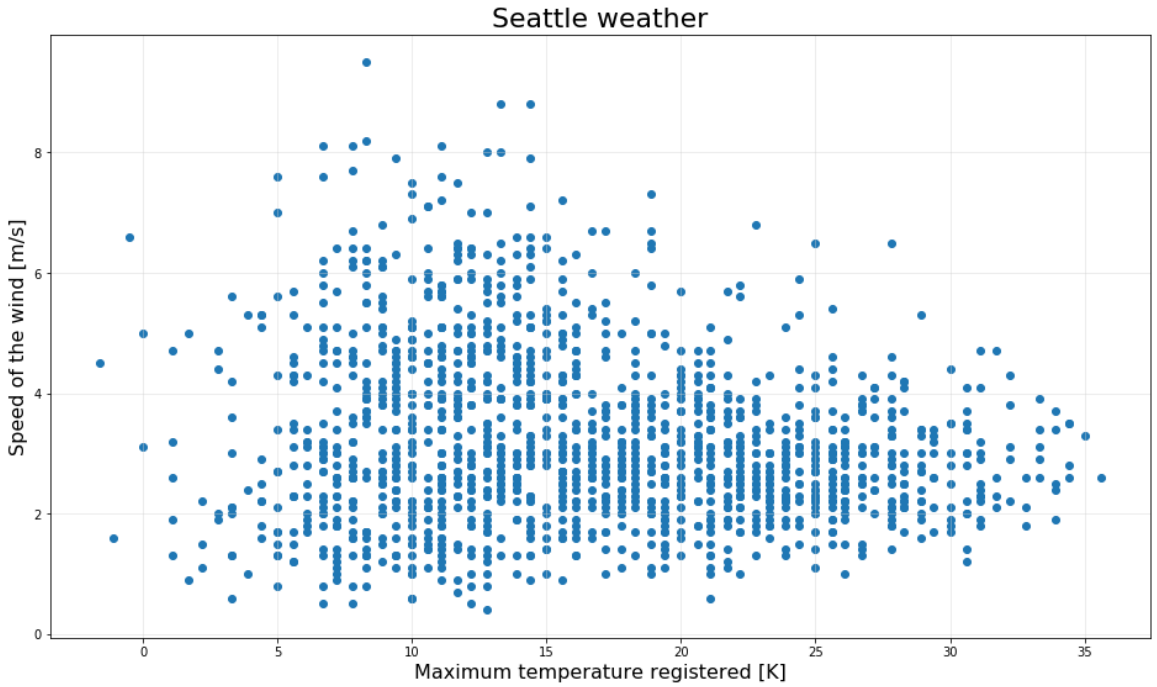



**Figure 2.5.** Graphic example of a scatter plot created by the data inside the seattle weather dataset.

As can be seen in the figure 2.4, in the scatter graph the pairs of variables $x$ and $y$ are represented as points of a Cartesian plane. The explanatory variable is commonly found on the x-axis while the response variable on the y-axis (ordinates axes). Every single point represented is equivalent to a statistical unit while the cloud of points visually describes the relationship between the two variables. It is customary to use these representations to outline areas of high, medium, low density, to analyse the possible correlations present, but also to highlight possible irregularities in the datasets.

### 2.2.2 Position indices and variability

As well as the graphical representation, in the numerical variables' analysis it is useful to have at disposal also numerical indexes of synthesis that allow to describe synthetically the main characteristics of the observations.

The **position index** or **central trend index** is defined as a measure that describes the order of magnitude of the values observed, namely its "centre". It is therefore a single value that can be considered "central" to the frequency distribution.
The main position indices are summarized in figure 2.6.

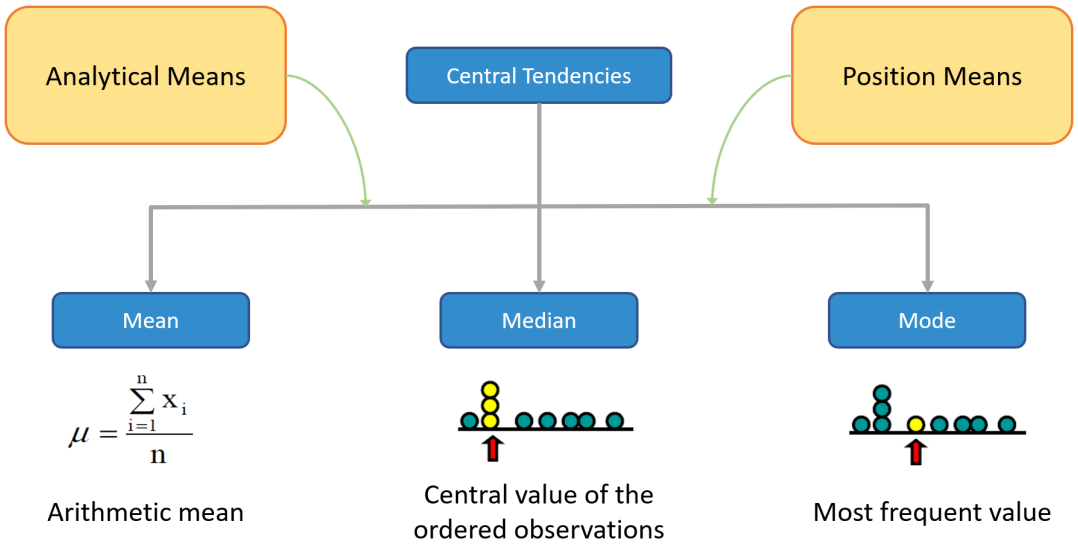



**Figure 2.6.** Summary of the principal positional indices in statistic

Through these position indices an attempt is made to synthesize a statistical distribution, with the aim of collecting the relevant parts of the information contained inside. However, these indices are not sufficient to fully represent the distribution because one important aspect would be overlooked: variability. This value expresses the tendency of statistical units to manifest themselves in different ways. Therefore, the analysis of this trend is fundamental especially when comparing different distributions, this is because, with the same average, they can be different in terms of spreading. The computation of variability can be performed through several indices but the most used for quantitative variables is **variance**.
This index measures the distance of the data from the arithmetic mean, calculated through the squares of the differences. Since the unit of measure of the variance is given by the square of the unit of measure with which the data have been collected, to obtain an index of variability on the same scale of the original data is taken into account its square root, also called **standard deviation** [11, 16, 39].

### 2.2.3 Tests for checking data normality

Many of the statistical procedures, belonging to the group of parametric tests, are based on the hypothesis that the data follow a normal or Gaussian distribution. Since it is of fundamental importance to study the data distribution curve, which explicitly expresses the frequencies (absolute and relative) of the values of the variable under consideration, it is necessary to analyse how well this distribution can be approximated to the normal one [21, 40].

More precise information can be obtained by using certain normality tests to determine whether the sample under examination belongs to the population of normal distributions. These tests, generally, are complementary to the graphical techniques of visualization of the distribution.
Among the most famous tests used in these situations there is the one of Kolmogorov-Smirnov (KS), the one of Lilliefors that is a correction of the KS, the one of Shapiro-Wilk (SW), the one of Anderson-Darling, the one of D'Agostino-Pearson and many others.
These methods compare the results of the sample with a set of values normally distributed with the same mean and the same standard deviation. In this case the null hypothesis states that "the sample distribution is normal". Considering this assertion, if the value obtained from the

test is significant then the distribution will not be normal.
In the case of small databases the normal tests have less power in rejecting the null hypothesis while, for large samples, a significant result would be obtained even in the case of a small deviation from the normal, despite this small gap would not affect the parametric test results. To achieve a better interpretation of the results, the p-value is also calculated, useful to define with more clarity when the null hypothesis is rejected or accepted, which must be compared with a certain level of significance, called $\alpha$, usually set to 0.05. This parameter indicates the percentage of risk of getting a false positive from the normality test, i.e. accepting the null hypothesis when instead it had to be rejected. Ultimately, we can have two specific cases when comparing these two values [21, 40]:

- *p-value* $\leq \alpha$: in this case the null hypothesis is being rejected, reaching the conclusion that the data do not follow a normal distribution.
- *p-value* $> \alpha$: in this case it can be deduced that the data do not follow a normal distribution, failing to reject the null hypothesis. This means that we do not have sufficient evidence to conclude that the data do not follow a normal distribution but, however, we cannot even conclude that their distribution is normal.

Normality tests can be classified into regression and correlation based tests (Shapiro-Wilk), empirical distribution based tests (Kolmogorov-Smirnov, Lilliefors correction, Anderson-Darling and Cramer-von Mises) and, lastly, moment-based tests (D'Agostino-Pearson test) [56]. The main features of these tests will be discussed below.

The Kolmogorov-Smirnov test (also abbreviated as KS) is a statistical test belonging to the most important class of EDF statistics (Empirical Distribution Function), in which the hypothetical cumulative distribution function is compared with the one obtained with the empirical data. In essence, this analysis compares the cumulative distribution of the data with a predicted normal cumulative distribution and calculates the associated p-value taking into account the greater discrepancy between the two distributions.
All what has been said is written as:

$$D = sup_x |F^*(x) - F_n(x)| \tag{2.3}$$

where $F^*(x)$ is the hypothetical normal distribution function with a specified mean $\mu$ and a standard deviation $\sigma$ while $F_n(x)$ is the EDF of the data. The larger the value of $D$ the more the distribution of the data will not be normal.
This test, however, works only in some particular cases where the parameters of the hypothetical distribution (mean and standard deviation) are fully known. To overcome this problem Lilliefors slightly modified the previous test, estimating these two parameters through the available data. Doing so, the formula that encloses these hypotheses, considering a sample of $n$ observation, is:

$$D = max_x |F^*(X) - S_n(X)| \tag{2.4}$$

where $S_n(X)$ is the cumulative distribution function of the selected sample and $F^*(X)$ is the normal cumulative distribution function having $\mu = \bar{X}$, the sample mean, and $s^2$, the sample variance, defined by denominator $n - 1$.
Although the LF test is virtually identical to the KS test, the table for critical values is different, leading to difference conclusions regarding the normality of the data [40, 49].

Another test, widely used to define whether some data follow a normal distribution, is the one of Shapiro-Wilk, which is based on the correlations between the data and the corresponding normal scores. It provides greater ability to detect whether or not a sample belongs to a non-normal distribution compared to the KS test, despite Lilliefors correction. For this reason, the Shapiro-Wilk is the most commonly used test by researchers. The formula that allows us to compute the value from which to subsequently extract the conclusions regarding the normality of the data is:

$$W = \frac{(\sum_{i=1}^{n} a_i x_{(i)})^2}{\sum_{i=1}^{n}(x_i - \mu)^2} \tag{2.5}$$

where n is the number of observations, $x_i$ represents the ordered sample values while $x_{(i)}$ represents the i-th smallest value of the sample and $a_i$ are the tabulated constants calculated from the covariance matrix and the mean of the sample compared to a normally distributed sample. The value of $W$ is always within the range $[0, 1]$ and, if it is low, indicates that the sample is not normally distributed (thus rejecting the null hypothesis) [40, 49].

The Anderson-Darling test is also used to verify that the distribution of a certain sample of data is normally distributed. In this case it is computed how much the data fit to a specific normal distribution function. This analysis belongs to the quadratic class of the EDF because the calculations carried out have to do with the quadratic differences. The Anderson-Darling coefficient is calculated as follows:

$$W_n^2 = n \int_{-\infty}^{\infty} [F_n(x) - F^*(x)]^2 \psi(F^*(x)) dF^*(x) \tag{2.6}$$

where $\psi$ is a non-negative weight function and $F^*(x)$ and $F_n(x)$ regard the same distribution functions discussed before. This test, being a modification of the Cramer-von Mises test in which $\psi = 1$, differs for the fact that it gives more weight to the tails of the distribution. Keeping in mind this detail it is possible to explain the formula of the Cramer-von Mises test as [40, 49]:

$$CVM = n \int_{-\infty}^{\infty} [F_n(x) - F(x)]^2 F(x) dF(x) \tag{2.7}$$

Finally, the D'Agostino-Pearson test which, as first thing, analyses the data to determine the skewness (which quantifies the asymmetry of the distribution) and the kurtosis (which quantifies the shape of the distribution). After that, it computes both how much each of these two values differs from the expected values, considering a normal distribution, and a single p-value from the sum of the quadratic differences. All this is formulas can be expressed in the following way:

$$K^2 = Z^2(\sqrt{b_1}) + Z^2(b_2) \tag{2.8}$$

where $Z^2(\sqrt{b_1})$ and $Z^2(b_2)$ are the standard normal deviations equivalent to observation of $\sqrt{b_1}$ (skewness) and $b_2$ (kurtosis) [40].

### 2.2.4 Connection and Correlation Tests

When the main objective is the prediction of the value of a certain variable of interest, it is absolutely necessary to observe how much and how (positively or negatively, when possible)

the values of other characteristics are related to it. Moreover, it is also interesting to look for possible relationships between pairs of input variables to rule out those that provide the least information. This problem can be addressed by analysing the data obtained from the study of the association between two possible variables ($X$ and $Y$). In this case, $X$ and $Y$ are said to be "connected" when, under one mode of the first, the one of the other is arranged in a special way. The types of associations available are mainly three and differ from each other by the type of variables that it is needed to be compared:

1. both variables are qualitative.
2. both variables are quantitative.
3. the couple is composed by both types, creating a pair of mixed variables.

Analysing the first case, if there are numerous sets, it is appropriate to distribute the data in a double entry table called *contingency table*. These are generally used in statistics to represent the joint frequencies of variables, dividing the total absolute frequency ($N$) into $R$ row categories and $C$ column categories, adding also an extra row and column to represent the marginal frequencies, that are the sum of the column and row values respectively. Since the variables taken into account in this case are of a qualitative type, the most popular independence test used in the scientific field is the non-parametric test (meaning that is independent from the distribution) of $\chi^2$ (pronounced Chi-squared) [9, 23, 31, 57]. This test is based on the observed frequencies, which can be enclosed within the contingency tables, and compares them with the expected frequencies calculated according to the statistical rules, under the condition of independence relationship and therefore of non-association, thus returning important information both on differences and on the categories that represent them exactly [23]. Basically, the expected frequencies represent an estimate of how the cases would be distributed if there were no disruptive effects.
Like any non-parametric test, $\chi^2$ also has conditions that must be met in order to be usable and to obtain truthful results [31]:

a. the data to be analysed must be only frequencies or case counts rather than percentages or other possible transformations.
b. the assignment of a case (also called subject) to a category (representing a cell of the contingency table) must be unambiguous to verify the independence of the cases.
c. the contingency tables shall contain at least 80% of the cells with frequency values greater than 5, at most 20% of the cells with values below 5 and no zeros.
d. the data sample must be large enough as the test is based on the approximate approach.

Suppose that in a particular sample it has been observed that a set of possible events $E_1, E_2, \ldots, E_k$ show up with observed frequencies $o_1, o_2, \ldots, o_k$ and that, according to the rules of probability, there is a second set of possible events with theoretical or expected frequencies $e_1, e_2, \ldots, e_k$. From here we can define the $\chi^2$ test by the following formulation:

$$\chi^2 = \sum_{i=1}^{n} \frac{(o_i - e_i)^2}{e_i} \tag{2.9}$$

Relying on quadratic differences, if these are high, they will consequently result in a high value of $\chi^2$, thus implying that the independence hypothesis applied to the calculation of the expected

values is irrelevant to the observed ones. Ultimately, the null hypothesis of independence between variables is rejected when the value of $\chi^2$ is higher than a specific critical threshold available in the distribution table of $\chi^2$, taking into account also how many degree of freedom are associated with the case in analysis [23, 57].
The advantages of this test include the robustness of the data distribution, the ease of computation, the detailed information that can be obtained, its flexibility in the data processing from multiple groups and the fact that it can be employed in studies that do not meet the assumptions of parametric tests. Nevertheless, the $\chi^2$ test also has limitations such as the need for numerous samples and the difficulty of interpretation when there are large amount of categories in dependent or independent variables.
This test, in some cases, can also be applied to continuous data only if the values are discretized by grouping them in intervals on a continuous scale. Doing so, the $\chi^2$ can be performed after identifying the frequency diagram. The result of the test depends strongly on the width of the intervals and is considered relevant only if the constraint on the frequency value specified in each cell of the contingency table is respected (hypothesis $c$, explained previously) [32].

Since the $\chi^2$ test returns a value that is difficult to be interpreted, being always a number greater than 0 and sometimes also of 1, it is generally associated with a strength statistic (i.e. a correlation calculations) able to give a fairly immediate representation of the result. One of the most widely used strength statistic in science is Cramer V, applied to test data when a significant value of $\chi^2$ has been obtained [31]. It can be easily calculated by the formula:

$$V = \sqrt{\frac{\chi^2}{n(k-1)}} \tag{2.10}$$

where $n$ is the size of the whole sample and $k$ is the minimum size of the dimensions of the contingency table (the minimum value of the number of attributes available in the two variables compared). The values of this statistic always belong to the range $[0, 1]$. One of the major problems of this index, however, is the tendency of returning relatively low values despite the relevance of the results computed with $\chi^2$ but it is the one most used given its ease of application in almost all situations, unlike the other two strength tests (the $\phi$ and the Odds ratio) that are used only in presence of $2 \times 2$ contingency tables.

In case, instead, the variables of the pair taken in analysis, to understand whether they are independent or not, are both characterized by quantitative values should be used the correlation coefficients. These statistics, in addition to expressing the intensity of the bond present in the pair, also show the direction and, in fact, it is for this reason that their value belongs to the range $[-1, 1]$. The result, therefore, varies from a perfect inverse correlation (meaning that to the growth of one variable the other one decrease) up to a perfect direct correlation, passing through the value 0 representing its total independence.
The term correlation is usually used to express any form of relationship, association and connection between two variables. In reality this term refers only and exclusively to a reciprocal relation of linearity and it is for this reason, therefore, that a mathematical relation present between two characteristics does not imply the presence of correlation [36].

One of the most important correlation indices is the Pearson coefficient, used to compare quantitative variables with a normal bivariate distribution [24]. This index measures how much the relationship between the two variables approaches the ideal case, in which each point of the dispersion diagram lies exactly on a straight line. It is represented with $\rho_P$ for a population parameter and with $r_P$ for a sample statistic. Of this coefficient it is possible to express, first

of all, a graphical explanation (shown in figure 2.7). Suppose there is the need to calculate the association between $n$ pairs of continuous data ($(y_i, x_i)$ with $i = 1, 2, \ldots n$). To perform this analysis two different regression lines have to be drawn: the first ($l_1$) corresponds to the linear regression of y on x while the second ($l_2$) corresponds to the linear regression of x on y. They intersect at the point having as coordinates the mean values of the observed $y_i$ and $x_i$ values, respectively, creating an angle $\theta$ which expresses the force of the association between the two variables. In fact, geometrically, the Pearson coefficient is the geometric mean of the slopes of the two lines, which corresponds to $\cos\theta$. For this reason, in the case of total independence, the two lines will be perpendicular ($\theta = 90°$) while if there is complete association, regardless of whether it is positive or negative, the two lines will overlap ($\theta = 0°$) [2].

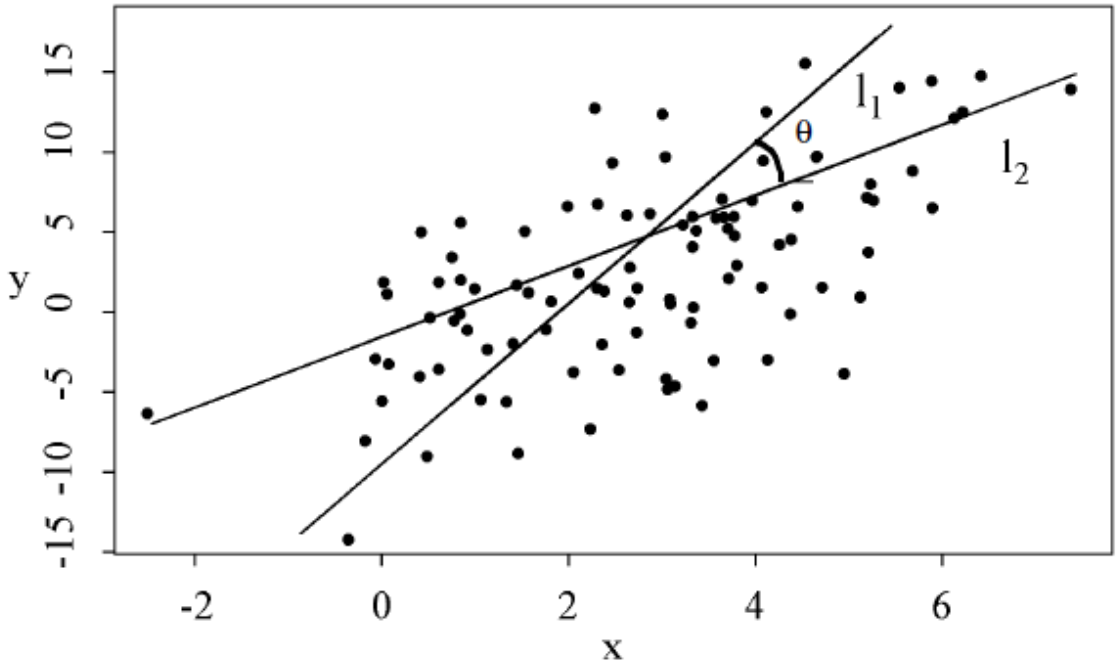



**Figure 2.7.** Geometrical Interpretation of the Bravais-Pearson correlation coefficient [2].

From a purely mathematical point of view, Pearson's correlation coefficient is expressed by the following formula:

$$r_P = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{[\sum_{i=1}^{n}(x_i - \bar{x})^2][\sum_{i=1}^{n}(y_i - \bar{y})^2]}} \tag{2.11}$$

where $x_i$ and $y_i$ are the i-th values of the two variables whose correlation is to be studied. The numerator is called covariance and the denominator is the product of the mean quadratic deviations of the variables $x$ and $y$. The covariance represents the level of dependence between the two variables and can take the value 0, in the case where varying one of it the other remains constant, the maximum value in positive absolute value when the points are all aligned on a rising straight line or the maximum value in negative absolute value when the points are all aligned on a decreasing straight line.
The Pearson coefficient is highly influenced by extreme values, which can overstate or dampen the strength of the relationship, and therefore is inappropriate when even one of the variables is not normally distributed. In the case where a large population operates only over a narrow range of data, the correlation coefficient tends to decrease in absolute value. Hence, it is very difficult to interpret a statistic calculated from selected data based on the values of a variable. It is of paramount importance to note that, if the assumptions associated with the index are not met, the probability statements on correlations can highly influence the magnitude of the

result obtained.

When the variables to be analysed are quantitative, but do not have a normal bivariate distribution, two other coefficients are used: the Spearman correlation coefficient and the Kendall correlation coefficient.

The Spearman's rank correlation coefficient was among the first to be processed. It is a non-parametric technique that evaluates the monotonic relationship, linear or not, present between the two variables under consideration. It is usually represented with $\rho_S$ for a population parameter or with $r_S$ when dealing with a sample statistic [36]. From a geometrical point of view, the Spearman coefficient is calculated in the same way as the Pearson one, but it implies a simple transformation. In fact, instead of using the i-th couple of the collected values, the pairs of assigned ranks are considered, taking into account their relative magnitude. This transformation allows to combine the data according to the same scale of measure, the so-called scale of ranks, compared to the original one. Since the ranks do not follow a normal bivariate distribution, the correlation, in this case, can be interpreted geometrically as was the Pearson coefficient. In fact, the one of Spearman is an indication of the monotonicity of the present relationship. This means that, after replacing $y_i$ and $x_i$ with the corresponding ranks, the monotonicity of their relationship does not imply a linearity between the corresponding grades. Such index assumes value 1 when to the increment of the first variable increases also the second one and $-1$ when to the increment of the first the second decreases, but without that these increments are constant like in the straight line [2].
From the mathematical point of view, the Spearman coefficient is expressed by the formula:

$$r_S = 1 - \frac{6 \sum d_i^2}{n(n^2 - 1)} \tag{2.12}$$

where $d_i = X_i - Y_i$, $X_i$ and $Y_i$ values are respectively the ranks of the variables $x_i$ and $y_i$ and $n$ is the number of observations present in the sample. The magnitude of these differences gives an idea of the relationship between the two sets of tanks. A perfect association would be obtained in case $\sum d_i^2 = 0$, thus obtaining an $r_S = 1$ and therefore a perfect correlation. Instead, the larger the differences between the ranks, the less perfect the correlation between the two variables. The most important defect inherent in the computation of this coefficient is the fact that the result is estimated in excess if, for at least one variable, there are many equal ranks (meaning if there are many equal values). Beyond that, the characteristic that distinguishes the coefficient of Spearman from the Pearson one is the robustness in presence of extreme values.

The Kendall's rank correlation coefficient, also known as Kendall's $\tau$ (Tau), is applicable, as non-parametric measure, to the same type of data as the one of Spearman. It is used to define the degree of similarity between two sets of ranks, taking into account the same set of objects. Given a pair of variables $X$ and $Y$, in order to calculate the $\tau$ of Kendall it is necessary to order the values couple $(x_i, y_i)$ according to the increasing order of $X$. Subsequently, for every $y_i$ both the greater and the smaller values that follow it are counted, defining $s_i^{max}$ and $s_i^{min}$ respectively. If there is one of the following values equal to $y_i$, the count of both the corresponding $s_i^{max}$ and $s_i^{min}$ is increased by 0.5.
By mathematical formulae, the Kendall's rank coefficient can be expressed as:

$$\tau = \frac{2S}{n(n-1)} \tag{2.13}$$

where $n$ is the number of couples and $S$ is computed as $S_{max} - S_{min}$, that is the difference between the higher total values ($S_{max} = \sum s_i^{max}$) and the lower total values ($S_{min} = \sum s_i^{min}$).

From the formula can be deduced that $\tau$ is the difference between $S_{max}$ and $S_{min}$ expressed as a fraction of $\frac{n(n-1)}{2}$, that is the total number of ways in which $n$ units can be compared two at a time. It is important to notice that for this test is always checked the equality $\frac{n(n-1)}{2} = S_{max} + S_{min}$.
The differences between Spearman coefficient and Kendall Tau are mainly three:

1. the reliability of confidence intervals, which are minor in the first one.

2. the ease of computation, which is always greater in the first one.

3. Kendall tau shall be from the lack of excessive estimation of the result in the case of a high presence of equal ranks.

Furthermore, these coefficients use different scales of assessment and are not directly comparable numerically. However, they have the same ability to detect the presence of association relations within the population, using the same amount of information present in the data.

Finally, to be able to analyse the last possible type of association, need to be treated the case when there is the need to calculate the degree of connection between a quantitative variable and a qualitative one. In this case, it is useful to study the average dependence that exploits the additional information in having a quantitative character.
Considering a qualitative variable $Y$ and a quantitative one $X$, there is independence on average when the conditioned averages are identical to the marginal average, that is when the conditioned averages of $X$ with respect to $Y$ are equal, denoting that $Y$ does not influence the mean of $X$.
To compute this statistic, starting from a mixed distribution, it is necessary to create a contingency table in which we break down the overall variability of the quantitative character with respect to the modalities of the qualitative one, thus performing the so-called *variance decomposition*. At this point the total variance of the $X$ is expressed as the sum between the mean value of the conditioned variances and the variance of the conditioned averages that in formulas can be represented as:

$$V[X] = E[V_{X|Y}] + V[E_{X|Y}] \tag{2.14}$$

where the first addend is the unexplained internal variance, given by the sum of the variances of the individual conditional distributions of $X$ with respect to the modes of $Y$, and the second one is the explained variance, equivalent to the variance of the conditioned averages with respect to the general mean of $X$.
To measure the strength of the dependency bond, the correlation ratio $\eta^2$ (eta-squared) is used, a normalized index that exploits the variance splitting property. This is an asymmetric index obtained by comparing subordinate distributions of $X$ with $Y$ modes. In formula, it can be expressed as:

$$\eta^2 = \frac{V[E_{X|Y}]}{V[X]} \tag{2.15}$$

This index assumes values in the range $[0, 1]$. With the value 0 there is a total independence in average, meaning the variability of $X$ does not depend on the modes of $Y$, while with value 1 the dependence in average is maximum, meaning a total dependence between the two variables. Finally, it is important to notice that, being an asymmetric index, the eta-squared allows to verify the relationship between X and Y, but nothing can be said about the inverse relationship.

## 2.3 Supervised Learning Algorithms

When dealing with a huge amount of data it is desirable to have a certain type of automation or data-driven process that can extract the "knowledge" to use, in turn, for future predictions [17]. In these cases, there is the possibility to use Supervised Learning algorithms that are employed when the data already have the target variable, that is the feature that have to be predicted. The term "supervised" refers to the training dataset in which this variable is already known and, therefore, supervises the learning of the algorithm [8, 47, 50]. In fact, the starting idea is that the data provide the model with examples of situations similar to those to which it will be subjected in the future [54]. These algorithms create, therefore, a certain function $f$ able to map the values of the input variables ($x$) into the values of the output variables ($y$, i.e. the target variable). The goal is to approximate this function so well that, when there are new data, there is the chance to predict the $y$ with the greatest accuracy possible, considering that this value will never reach 100% [8, 17].
Models follow an iterative learning process because, whenever the algorithm makes a prediction, feedbacks or corrections are provided to improve performance which, once acceptable accuracy thresholds have been reached, are the condition of termination.

When choosing a supervised learning algorithm, some important aspects need to be considered. First of all, it is essential to study the main characteristics of the data to understand: how they are distributed, how precise are the associated values, how to find some irregularities inside and, finally, see if there is a certain linearity. In addition, it is also necessary to consider the complexity of the model or of the function that the system must learn [50]. One of the major drawbacks of using these algorithms is the need of large amount of data available for training the model in the best way possible.

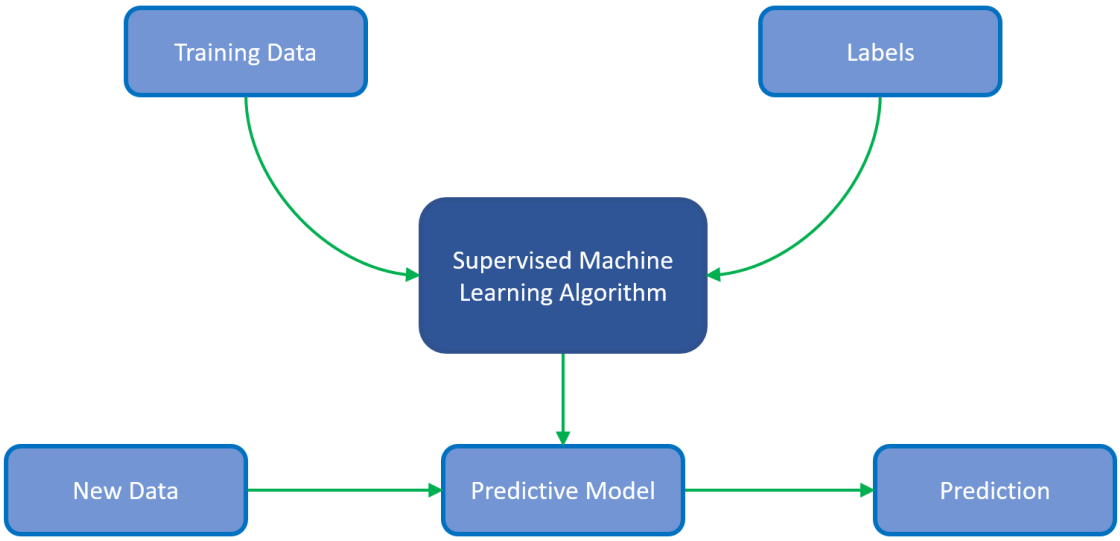



**Figure 2.8.** General pattern for the creation of a supervised learning model [47].

Generally, the first thing to do is to separate the data into two distinct groups, the training set and the test set, both of which have the target variable associated. This separation is necessary because, usually, the training is carried out with the data belonging to the first group and, subsequently, to understand the degree of reliability of the model, the second group is used to calculate its performance. In this way it is possible to estimate whether the selected algorithm can adapt to the available data and, also, find possible violations, such as the use of features

not useful for the purpose, bad choices in the selection of algorithm parameters and insufficient training data [17].

Figure 2.8 summarizes in general the main steps for the creation of a predictive model of supervised learning. In fact, the input variables and labels, which together compose the labelled data, are passed to an algorithm able to extrapolate from the data the $f$ function necessary to mathematically connect the two characteristics. This creates a predictive model that can provide predictions for new data not yet associated with a target variable [47].

Depending on the type of the target variable, two categories of algorithms can be distinguished:

- Classification algorithms
- Regression Algorithms

The first (classifiers) will be treated in section 2.3.1 while the others (regressors) will be treated in section 2.3.2.

At the beginning, it is considered a good practice to train more than one algorithm and then choose the one that best suits the data, using performance measures. In the case of classification algorithms, the evaluation metrics that can be used are confusion matrices, accuracy, prediction, recall and F-measure.
The *Confusion Matrices* are computation methods used to study the performance of classification algorithms. They are tables able to compare the real values of the target variable and those generated by the model. In this way it is possible to visualize in a very simple way the values associated with True Positive, True Negative, False Positive and False Negative [37]. In general, for $n \times n$ matrices (where $n$ is the number of classes), these values can be calculated as follows:

- *True Positive (TP)*: for each class, it corresponds to the specific cell belonging to the main diagonal of the matrix.
- *True Negative (TN)*: for each class, it is the sum of the cells in the main diagonal, except for TP.
- *False Positive (FP)*: for each class, it is the sum of all values in the corresponding column, except for the TP.
- *False Negative (FN)*: for each class, it is the sum of the values in the corresponding row, except for the TP.

The *Accuracy* is used to calculate the number of correct predictions (regardless of class) compared to the total of those performed [4, 17].

$$Accuracy = \frac{Number\ of\ corrected\ classified\ objects}{Number\ of\ classifications} = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.16}$$

It is a very simple measure to understand and calculate but it is not appropriate in case there is a different concentration of classes or when they have no homogeneous relevance, corresponding to cases where some are more than others.
The *Precision* is the number of correct predictions compared to all those performed to which has been assigned the same class.

$$Precision = \frac{Number\ of\ objects\ correctly\ assigned\ to\ a\ class}{Number\ of\ objects\ assigned\ to\ that\ class} = \frac{TP}{TP + FP} \tag{2.17}$$

The *Recall*, instead, is the number of correct predictions assigned to a certain class compared to all those performed that belong to that specific class.

$$Recall = \frac{Number\ of\ objects\ correctly\ assigned\ to\ a\ class}{Number\ of\ objects\ belonging\ to\ that\ class} = \frac{TP}{TP + FN} \tag{2.18}$$

These last two measurements, however, alone do not provide any real information about the performance of the algorithm [4]. Therefore, it is more commonly to use a metric that summarizes and unifies the precision ($p$) and recall ($r$) values called F-measure or F1 score.

$$F\text{-}measure = \frac{2rp}{r + p} = \frac{2 \cdot TP}{2 \cdot TP + FN + FP} \tag{2.19}$$

It represents the harmonic mean between precision and recall and takes values within the range $[0, 1]$. The harmonic mean tends to approach to the smallest of the two numbers considered and, therefore, if it assumes a high value then also precision and recall are high.
In the case of multi-class classifiers, it is necessary to compute accuracy, precision, recall and F-measure for each individual class. The arithmetic averages of each of these metrics give an overview of the model, returning four values that characterize it.

In the case of regression algorithms, two specific measures can be used: the mean squared error (MSE) and the correction of the Determination coefficient (adjusted $R^2$). These measures estimate the level of agreement between the predictions made by regression algorithms and the actual values assumed by the values of the target variable.
The *MSE* indicates the mean quadratic difference between the expected and actual data values. In Formula, it is represented as:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{2.20}$$

This measure has the advantage of being easy to understand, but it has the disadvantage of having an unlimited scope, as it is possible to get a value that may even be greater than 1, and for this are required some tabular references to draw conclusions.
The *adjusted* $R^2$, on the other hand, measures the percentage of data variance that the regression model can explain and takes values within the range $[0, 1]$ [17].

Model validation techniques may also be used to understand whether an algorithm fits well or not the data. Such validation is the useful process to understand how acceptable the results are. Usually, an estimate of the error is made immediately after the creation of the model, giving only an idea of how well this works. In fact, the model could have learned from over-fitting or under-fitting situations, thus failing to give a precise indication of how it will behave with new data [4, 22]. In order to make the validation phase more effective, partitioning techniques are applied, which includes the various types of cross-validation such as holdout, k-fold and leave-one-out.

The *Holdout* is a method that provides a fixed division of the dataset, reserving two-thirds of the data for the training set and the remaining one-third for the test set. It is appropriate for large datasets but is conditioned by the presence of a high variance, since it is not known a priori which data become part of the train set and which of the test set. In case it is repeated several times, mixing each time the data, it is called repeated holdout.
The *K-Fold* is usually used when there are not enough data to get a good training phase. In fact, by reducing the size of the train set there is the risk of losing important information, thus

increasing the final error. Using this method the data are divided into $k$ folds, of which $k-1$ are used as the train set and the remaining one is used as a test set, all repeated $k$ times so that all groups can be used as test. In this case, the estimated error is equal to the average of all the $k$ relative errors computed. This reduces the bias significantly, since most of the data is used for fitting, as well as variance, since all folds are used as test set at least once. It is a method that provides a good estimate of the model but is computationally expensive in case there are a lot of data. Usually, as values of $k$ are commonly used $k = 5$ and $k = 10$ and it is possible to notice how the case with $k = 3$ is extremely similar to holdout. Nevertheless, the choice of this value depends very much on the case being analysed.
Finally, there is the *Leave-one-out cross validation* (LOOCV). This technique is a particular case of the K-Fold in which $k = n$, meaning that only a record at a time is part of the test set. This method returns very exhaustive results but there is the problem that the process is heavy, computationally speaking, reason why it is usually used for very small datasets [4, 22].

### 2.3.1 Classification Algorithms

The classification is a subcategory of Supervised Learning that aims to predict discrete labels of new records, based on rules extracted from previously collected data. Prediction classes are discrete, unordered or even enumerative values that can be seen as a peculiar characteristic of a specific group. In case the classes to be predicted are only two, we will talk about binary or dichotomous classification, otherwise we have to deal with a multi-class classification [25, 47].

Training data are used to obtain better boundary conditions which, in turn, determine the rules for grouping each target class. Once the boundary margins are defined, then, the next task is to predict the class to be associated with new data [44]. Among the most commonly used algorithms there are the Logistic Regression and Naïve Bayes classifiers, both of which are part of linear models, Decision Tree, Random Forest classifier, K-Nearest Neighbors, Support Vector Machines and also some artificial neural network algorithms like Multi-layer Perceptron [25].

The **Logistic Regression**, despite its name, is a classification algorithm that tries to predict discrete values by using independent input variables. In short, it predicts the probability with which a certain event manifests itself, adapting the data to a $logit(p)$ function also defined as $\ln(\frac{p}{1-p})$, where $p$ is the probability that the characteristic of interest presents itself [44, 47]. This algorithm allows to understand the influence that some variables have on the target one but introduces the problem to work well only in the situations of predicting dichotomous variables [20, 41].

The **Naive Bayes** classifier is an algorithm based on both the assumption of independence between the input variables and of the conditioned probabilities, going to create trees based on the possibility that certain events occur [14, 44]. Such classifier assumes that the presence of a particular characteristic in a class excludes the presence of any other characteristic. In this way, there is the possibility to build a Bayesian model, so-called due to the fact that is based on the Bayes theorem, which is simple and functional in case of huge datasets. These are algorithms known for their prediction speed, but also for their poor accuracy statistics [20, 41, 44].

The **Decision Tree** is a very versatile algorithm that divides the dataset into two or more homogeneous sets based on the most significant attributes, to make the groups as distinct as possible. In this way, a model, able to predict the values of the target variable using simple decision rules deduced from the characteristics of the data, is created [41, 44]. Each tree is made up of nodes, each one representing a specific attribute in a group, branches, each of one

is equal to the values that certain attributes can take, and leaves, representing classes that can be assigned to an instance. In fact, the new records are classified with a top-down approach, starting from the root of the tree and going down to a specific leaf [14, 42].
Initially, the attribute that best divides the train set is to be searched, going to use methods such as the Gain or the GINI index. The same procedure is then repeated later for the creation of the other nodes, thus creating subtrees until all the features have been analysed and all the leaves have been generated.
As there is the possibility of overfitting the algorithms on training data, there are two approaches used to avoid this situation:

1. Stop the algorithm before getting to the point where the model perfectly fits the data.

2. Pruning the branches to delete those created by induced decisions.

In case two decision trees have the same type of controls and the same accuracy, we choose the one that has the least number of leaves and, therefore, the simplest at decision level and more understandable algorithm [42].
One of the most important advantages about Decision Tree, besides the easy visualization, is the comprehensibility since it is so easy to understand how certain classes have been assigned to an instance, through a path inside the tree. On the other hand, there is the problem that large trees created would not generalize in the best way, since even only a small variation of the values could imply totally different results [20, 42].

The **Random Forest** is a meta-estimator that adapts to a series of decision trees on various sub-samples of the dataset and uses the mean to improve the predictive accuracy of the model, thus also controlling overfitting situations [20]. Each tree within the group is constructed from a sample extracted by replacement from the training set. Also, when splitting each node during the building of the tree, the best separation is searched between all or only a part of the input variables. This will reduce the variance of the model. In fact, individual decision trees typically show high variance, thus creating overfitting situations. Doing so, decision trees with uncoupled predictive errors are produces and, by averaging these predictions, some errors can be undone. In this way, these algorithms achieve reduced variance by combining different trees. In practice, the reduction of variance is often significant and, therefore a better overall model is obtained [41]. Compared to normal decision trees, Random Forest reduces overfitting and is generally more accurate, but is a difficult and complex algorithm to implement and, in addition, is slow in real time predictions [20].

The **K-Nearest Neighbors (KNN)** is a classification algorithm based on learning by analogy, using the definition of similarity between tow objects such as distance functions (including the Euclidean, Minkowski and Hamming ones) [28, 42].
The dataset is described by numerical attributes that can be represented in an $n$-dimensional space. The moment there is the need of predict the class for a new record, the first operation performed is the selection of the $k$ nearest records. At this point, to the new data will be assigned the most frequent class within the selected $k$ points. This hyper-parameter $k$ is used to regularize the algorithm and, usually, is selected via cross-validation. If $k = 1$ then there is a particular case where the class is assigned taking into account only the closest point to the new record, while if $k = n$ the classes of all the points are taken into consideration, regardless of distance, thus creating problems in the situations of unbalanced classes (despite high $k$ values however help in the reduction of the noise effects). It is a good rule, in general, to select an odd value for this parameter to avoid draws for the label association [10, 28, 42, 44]. The choice for the value of $k$ is generally very difficult and depends mainly on the data and, therefore, on the

situation being analysed.
The KNN is among the easiest algorithms to implement, is very robust to the presence of noise inside the train set and is very effective for large datasets, being able to find patterns even very complex, but, keeping stored all the available cases, is very heavily, computationally speaking, compared to other algorithms and its results are difficult to interpret [10, 20, 42, 44].

The **Support Vector Machines (SVM)** are widely used algorithms based on margin delineation able to separate groups of data of different classes. In fact, we represent, as first thing, all the points belonging to the dataset in a $n$-dimensional space, where $n$ is the number of input variables. In case this number is greater than two, the separation lines will become hyperplanes, although the algorithm steps do not change [10, 14, 44].
The main problem of these algorithms, then, is to define what are the boundary that best split the data, searching for those that maximize the width, minimizing the classification error. The points that lie on these boards are called *support vectors* and are the only ones that influence the position of the separation line (the one parallel to the margins and placed in the centre of them) [10, 44].
So, in case the classes are linearly separable, the wider are the boundaries the higher will be the prediction accuracy of the classification model, thus showing that the classes have low similarity to each other. The problem, however, is that most real datasets are not linearly separable and any split line would involve prediction errors. Indeed, a margin is defined as being violated when a given record is wrong classified or, although the classification is correct, it is within the margins [10]. When linear separations are not available it is possible trying to split the data into classes through polynomial or any other type of functions, thus defining Kernel SVMs in which it can be specified the type of separation to adopt.
In SVMs, the boundaries are also chosen as minimization of the equation $\frac{1}{m} + C \sum p_i$ , where $m$ is the width of the margins, $p_i$ is the penalty for each single point and $C$ is a hyper-parameter (like $k$ for the KNN) used to balance the trade-off between margin width and classification errors. When $C$ is very low the classification penalties become irrelevant. Its value can also be chosen through cross-validation [10].
This type of algorithms is very effective when dealing with multi-dimensional spaces, requiring little computing power and offering a simple model interpretation and memory efficient, since it uses only part of the training points within the decision function. They allow, however, to identify a limited set of patterns, despite the use of Kernel SVMs, and do not directly provide estimates of probabilities, which can be calculated using an expensive 5-fold cross-validation [20, 41].

The **Multi-layer Perceptron (MLP)** is the most widely used model in the field of neural networks and is based on the back-propagation algorithm to perform the training. The main components, represented in figure 2.9, that make up a neural network are essentially three:

1. *Neurons*: where, in turn, the results of the previous neurons are processed and, subsequently, the output is defined according to a specific activation function. A set of unconnected neurons at the same depth is defined as layer.

2. *Connections*: connect neurons belonging to different layers. These connections can be weighted in such a way that the output of previous units follows specific rules for the operation of the network.

3. *Activation Function*: determines the behaviour of the neuron based on its excitation level, which corresponds to the sum of the values received in input by other neurons.

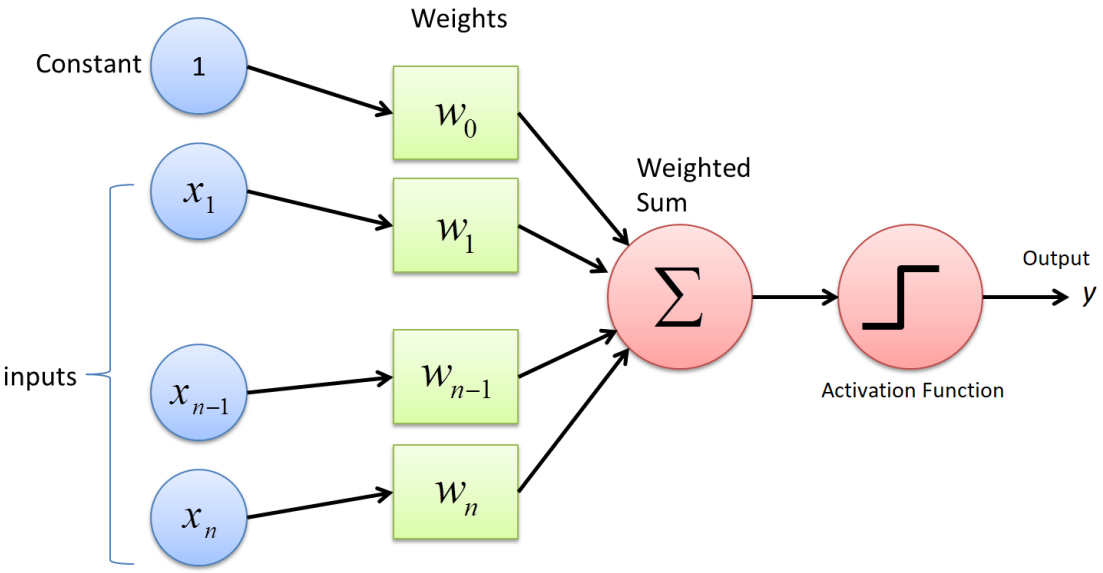



**Figure 2.9.** Representation of the components that constitute a neuron [1].

The MLP or, more in general, all the algorithms belonging to neural networks are composed of three types of layers: input, output and hidden layer. The first two are usually unique, while the number of the last one may vary depending on the problem in analysis [46]. The number of neurons in the input layer is equal to the number of input variables while the one of the output layer depends on the number of classes that could be predicted.

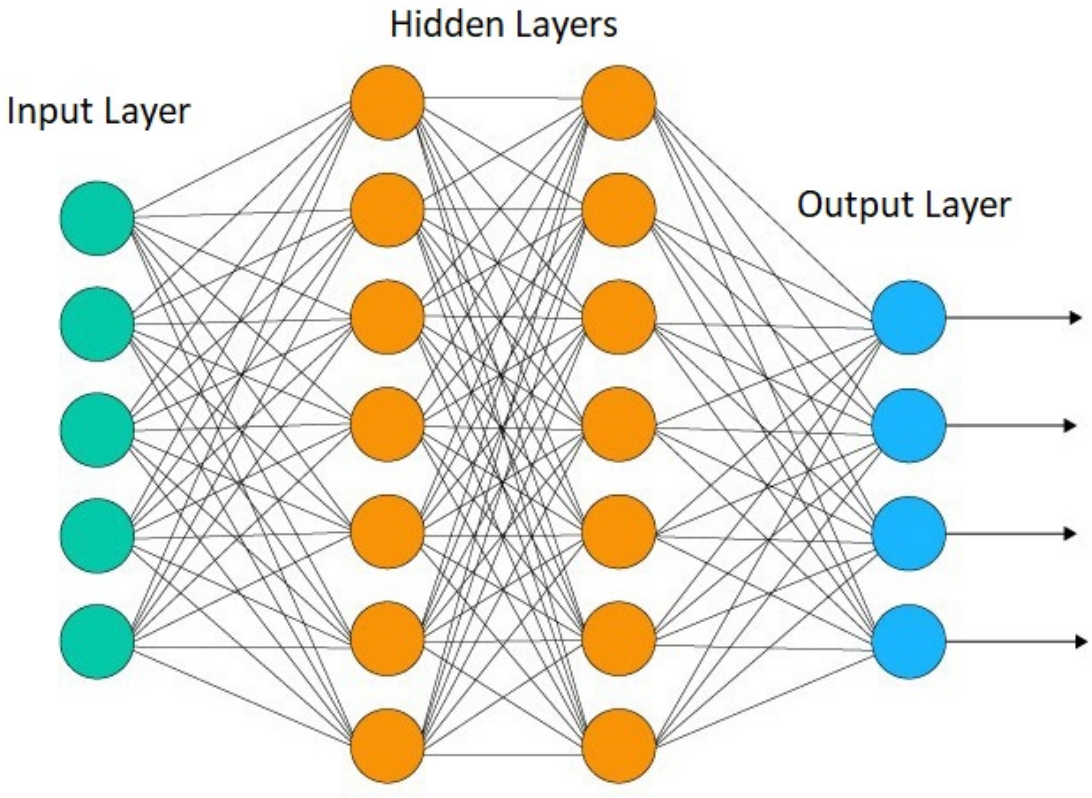



**Figure 2.10.** General representation of the architecture of an artificial neural network, in which the possible layer types are highlighted [19].

The architecture of the MLP and, more generally, of the neural networks, shown in figure 2.10, is extremely important, as the lack of a connection can make the network unable to solve the

problem of calibration of coefficients, while an excess of connections can cause an overfitting on the training data [46].
MLP tries to lean a function $f(\cdot) : R^m \rightarrow R^n$ by training on a dataset, considering m input variables and n different classes that can be predicted. Each neuron within the hidden layers transform the values from the previous layers, using a weighted linear sum, followed by a non-linear activation function $g(\cdot) : R \rightarrow R$ that decides the output to be sent to the next neurons. The learning process of this algorithm concerns the adaptation of the weights of each individual connection between neurons, to obtain a minimal difference between the predicted output and the desired one.
It is considered a good algorithm as it is able to learn real-time models, also characterized by non-linear functions. The problem, however, is that the hidden layers of the MLP have a loss function that can have more than a minimum local, requiring several random initializations of weights to try to improve accuracy. It is also required to tune some parameters including the number of hidden layers, neurons inside them and iterations. Finally, the MLP is also sensitive to feature scaling [41].

### 2.3.2 Regression Algorithms

In case the target variable to be predicted is a continuous numerical value, regression algorithms are used, based on a number of predictors (input variables) and an output variable, to try to determine a mapping function as precise as possible [25, 47].

The simplest and fastest algorithms in this field are those belonging to linear regression (like the method of least squares), but they return, in the vast majority of cases, some mediocre results [25]. They group together models used to predict the target variable where there is a linear relationship between the target variable and the input ones [47].
This kind of algorithms can be divided into three categories: simple, multiple and polynomial linear regression. The first is the simplest and concerns only those models in which the variable to predict depends exclusively on a single predictor. The resolution of these problems is usually entrusted to the computation of the inclination of a line with the formula $y = w_0 + w_1 x_1$. The second is an extension of the previous one and is used to solve problems where the output depends on several input variables, following the function $y = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_n x_n$, selecting only those features that best help in the prediction to be executed. In these two cases, the result always concerns the delineation of a straight line that best approximates all the data. However, if these methods are not accurate enough, polynomial functions, defined as $y = w_0 + w_1 x_1 + w_2 x_1^2 + \cdots + w_n x_n^n$, are used. These are anyway part of linear regressions, although the function is not, because there is a linear combination of coefficients. In order to use this last type of algorithms, however, it is first necessary to preprocess the data to calculate all the exponents of the needed variables [30].
Regression of the **Ordinary Least Squares (OLS)** is a statistical method of analysis that estimates the relationship between some independent variables and the dependent one. The least squares estimators allow to minimize the sum of the square of the differences between the observed and the expected values of the target variable, to create a straight line that crosses the values of the features of interest. When, however, some model variables are related, the error estimation becomes very sensitive to possible outliers, thus creating situations of multi-collinearity [41].
**Ridge** regression solves some OLS problems by imposing a coefficient size penalty. To do this, a hyper-parameter $\alpha$ is used. The larger this value, the more robust the coefficient will be to the collinearity [41].
The **Lasso** algorithm is a linear model used to estimate sparse coefficient. It is adopted in

cases where are preferred solutions with fewer non-zero coefficients, thus reducing the number of features on which the solution provided will depend [41].
Another commonly used linear model is the **ElasticNet**. This algorithm is trained by regularizations in accordance with $l_1$ and $l_2$ coefficients. This combination allows learning sparse model, where few weights are different from zero, as in Lasso, while maintaining the regularization properties of the Ridge algorithm. It is useful when there are many related features [41].

In addition to this kind of regressions there are other algorithms, used in classification problems, which can also be adapted to work with regression problems. Among these it is possible to find some algorithms already discussed in section 2.3.1.
As for the Decision Tree, the general principle is the same used in the classification, but with a single change: after having followed all the rules for the creation and having crossed the entire tree, it gets a specific leaf that, in this case, contains not the class to be assigned but a multitude of values whose average is the result of the prediction [30].
The Random Forest algorithm, as in the classification, is the combination of multiple decision trees to improve its accuracy statistics but, if used for regression, the result of the prediction is computed as the average of the values obtained in each individual tree [30].
The KNN is also an algorithm that can be applied to the regression problems by modifying the calculation method of the target variable for a specific record to which is assigned the mean of the values of the k nearest points instead of the most frequent class. In these cases, however, the contribution of each record is weighted according to the distance [28].
Also SVMs can be used for regression problems, taking the name of *Support Vector Regressor (SVR)*. In this case, the same principles as the SVM classifiers are employed and adapted for predicting continuous values. In fact, error margins ($\epsilon = prediction - realvalue$) are defined. This value determines the error region and the number of support vectors. Boundaries are drawn in parallel to the separation hyperplane, with a distance varying according to the $\epsilon$ value. Only the points outside the margins will contribute to the computation of the final cost, not considering those points whose value is close to the exact prediction. As in SVM classifiers, various kernel functions can be applied, depending on the type of relationship between the variables [15, 41, 51].
The Multi-layer Perceptron is also an algorithm that lends to both classification and regression problems. Here, again, the basic principle of the algorithm, specified in section 2.3.1, remains the same, applying the back-propagation in both the cases, but the activation functions in the output layer are no longer available. Thus, the quadratic error is used as a loss function and the output consists of a set of continuous values [41].

## 2.4 Unsupervised Learning Algorithms

Differently from Supervised Learning, in Unsupervised the algorithms work with data that is not known a priori neither the values associated with the target variable nor their internal structure. In fact, the term "unsupervised" is used precisely because there is no help from the outside to arrive at certain solutions. Algorithms are left to themselves to discover and present the fundamental structures present in the data [5, 8, 47].
The main purpose of these algorithms is to explore the structure to obtain information about different patterns present and extract important notions, in order to increase the efficiency of the decision process [5, 47].

This kind of algorithms is usually grouped into two large categories that will be analysed respectively in the sections 2.4.1 and 2.4.2:

1. *Clustering*: allows to divide the data into groups containing similar records.
2. *Dimensionality Reduction*: allows selecting only the most relevant information for solving specific machine learning problems.

### 2.4.1 Clustering Algorithms

Clustering can be considered one of the most important problems of Unsupervised Learning and tries to find structures within a collection of data that do not have labels, as was in the case of supervised learning. More generally, it is a technique of data exploration that can organize them in various subgroups (called clusters) without having any previous knowledge [34, 47]. The creation of these clusters depends on the notion of similarity which, in most cases, coincides with the distance between two points. In fact, two points will be as much similar as their distance will be minimal and, therefore, the creation of the groups will happen in such a way that the intra-cluster distance (the one between points belonging to the same cluster) is minimal, while the inter-cluster distance (the one between points belonging to different clusters) is maximum. Doing so, the widest possible separation margins will be defined [26, 34, 47]. Finally, it is possible to say that a cluster is a set of records "similar" to each other and the most "dissimilar" possible from records belonging to other groups.

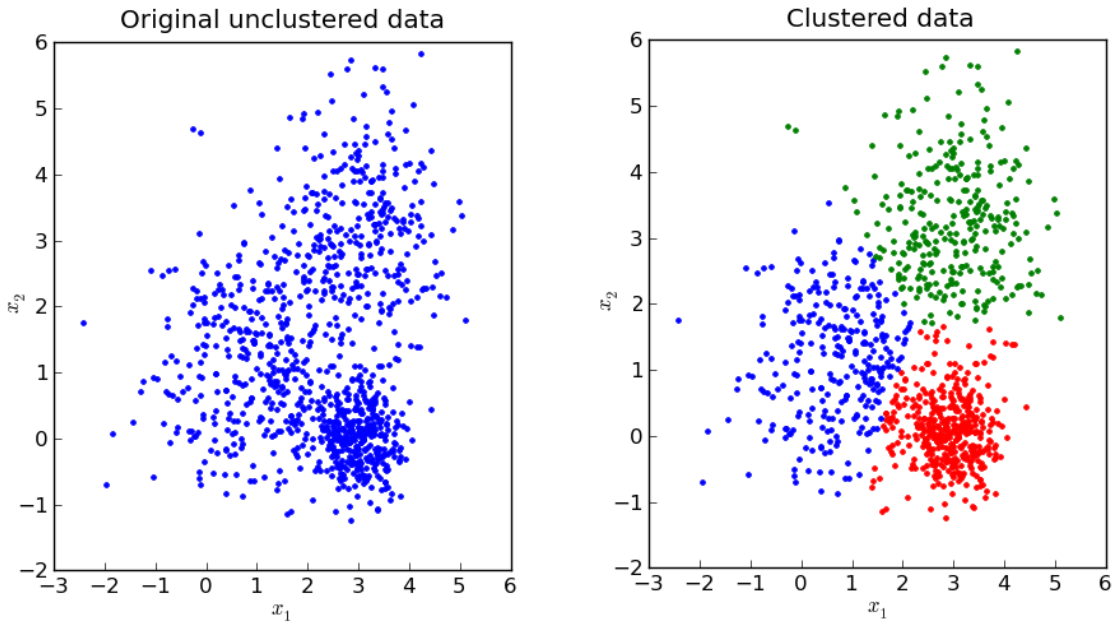



**Figure 2.11.** Example of a clustering process on 2D data representation [55].

The ultimate goal of clustering is to determine the internal grouping in an unlabelled dataset.it can be shown, in this sense, that there is no better criterion than others that is independent of this objective. Consequently, it is the user who decides which is the most suitable type of grouping its needs.

These techniques can be divided mainly into four distinct categories [4, 26, 34]:

1. *Exclusive Clustering*, where the data are grouped in exclusive way, making each record belong to a single cluster.
2. *Overlapping Clustering*, in which, unlike the exclusive, clusters can be overlapped, implying that records may belong to multiple groups at the same time.

3. *Hierarchical Clustering*, in which clusters are organized with a hierarchical structure, using dendrograms for their visualization.

4. *Probabilistic Clustering*, where a fully probabilistic approach is used.

The most used algorithm for clustering operations is definitely the *k-Means*, which is the easiest to implement and belongs to the exclusive clustering category.
The definition of the number $k$ of clusters to be searched within the data is up to the user and, from the arrangement of the initial centres, depend the final results returned by the algorithm. Usually, the centroids are chosen in such a way that they are as far away as possible from each other.
The first phase of this algorithm, in addition to the definition and arrangement of the centroids, also provides for the assignment of each point to a specific cluster, going to compute the distance from each centroid and choosing the one with the least distance. This step can be said to be concluded when all the points are assigned to a single group. At this point, there is the calculation of the new centroids as barycentres of the clusters created, as average of the points belonging to the same group. After that, the assignment of points of all the clusters is performed again, repeating the process until there is a stabilization of the groups, meaning that the difference between the old and the new centroids (considering two consecutive iterations) is below a certain tolerance threshold [34]. This minimizes the quadratic error within the clusters and creates groups of equal variance.
K-Means is a very efficient and easy to implement algorithm but is extremely sensitive to random selection of initial barycentres. In fact, it may happen that one of these does not present any point in the neighbourhood thus creating an empty cluster, since the various iterations would not change the content. It can be shown that, although the algorithm always finds a result, global convergence cannot be guaranteed and therefore, usually, multiple executions are made even with different initialization centroids (both by position and by number). Finally, it is important to notice that the algorithm returns circular and separable clusters and is very sensitive to both outliers and clusters with different shapes and densities [4, 26, 34].

Another type of clustering widely used is the hierarchical one employed in those situations in which there is the need to look for structures regarding connections existing between data. Two types of hierarchical clustering can be defined [26]:

1. *Agglomeration*: groups are built with a bottom-up process starting from as many clusters as the individual data, iterating until there will be only one cluster.

2. *Divisive*: it is characterized by the reverse process, top-down, starting from a single cluster and iterating until there will be a cluster for each single data.

Between the two types, the first is the one most used as it is simple to implement. Given a set of $n$ elements to classify and an $n \times n$ similarity matrix, the process starts always by assigning each record to a different cluster to have $n$ clusters. After that, the two most similar (and therefore closest) clusters are selected and jointed to create a new one and decrease by one the total number, also updating the similarity matrix. This process is repeated several times until there will be a single group [34].
This type of data partitioning also has a specific graphical representation called *dendrogram* where all the possible groupings are represented. A horizontal cut inside this graph represents a set of possible clusters and allows visualizing, in simple way, the maximum distance between points belonging to the same cluster in that specific situation (the vertical distance between the individual points and the cut) [4].
Even this algorithm is quite used as it is not necessary for the user to define the number of

clusters to be created, since the algorithm return all possible solutions. However, they can only be applied in those cases where the dataset is very small, as hierarchical clustering takes a lot of time to complete (with a complexity $O(n^2)$) and also a lot of memory resources.

In general, clustering problems are very time-consuming, more or less, when dealing with large datasets. In addition, besides depending on the definition of distance, these algorithms return results that can be interpreted in different ways, depending on the analysed situation and on the context [34].

### 2.4.2 Dimensionality Reduction Algorithms

The dimensionality reduction is an unsupervised learning problem that requires the model to reduce the number of features within the dataset used, deleting or combining variables that would have a limited or null effect on the result [25, 45]. Doing that the dimensionality of the data would be reduced, thus allowing a better visualization. Just think that the human being can see up to a maximum of three dimensions, while, normally, the data that are analysed far exceed this value.
The dimensionality reduction algorithms allow, therefore, to remove those variables that have many missing values or outliers and to eliminate the characteristics having low variance, which would not contribute to the understanding of the data or, even, would compromise it, thus simplifying its internal structures. Therefore, a data cleaning process must always be done before performing any other analysis [5, 25].
These operations are extremely useful since curse of dimensionality must always be taken into account. Often, in fact, the data used have high dimensionality, wherein each observation is characterized by a large number of variables, creating major complications in case the storage space is limited and there is low computing power to run a machine learning algorithm [47]. The curse of dimensionality is a huge problem that affects the entire field of machine learning and is based on the fact that, as the variables of interest increase, increases proportionally also the number of records useful for the representation of all possible combinations of values. In addition, the more features to be used, the more complex the model will be, even creating potential data overfitting problems. There is also to consider that, as the number of variables increases, the data become increasingly scattered within the space they occupy, thus decreasing the definitions of density and distance between points [4, 45].
The main purpose of dimensionality reduction, therefore, is to avoid overfitting and to select only those features that bring important information to the model, thus also reducing the effects of the curse of dimensionality. In fact, by doing so, it reduces the time and space required by data mining algorithms, as well as allowing a better visualization of data and the elimination of redundant variables [4, 45, 47].

Techniques that can be used for dimensionality reduction include *Aggregation* and *Sampling*. The first one is a process that allows to combine two or more attributes into a single one. This is done to reduce the number of variables on which the model would depend but also to stabilize the data, since aggregated attributes usually have less variability. This creates smaller representations in volume, providing anyway similar results for the same analysis. The second, indeed, is used when working with very large dataset. With this technique, in fact, is selected only a highly representative sample, that is a part of data that contains approximately the same properties as the original dataset, on which to perform the various studies. In doing so, it is possible to achieve appreciable results, which would differ slightly from those that would be calculated if would be used the entire dataset, but saving time and resources [4].

Finally, there are two other techniques worth mentioning. The first is called *Feature Selection* and is the process of identification and selection of important variables to be taken into account within the dataset [45]. This is a technique that, like the others, allows to decrease the dimensionality of the data, removing those variables that would not bring any more information to execute specific predictions, reducing therefore also the training time of the algorithms. In addition, there is a reduction in the complexity of the model which also implies a greater interpretation of the results. The selection of these characteristics is usually carried out through the computation of certain statistical values such as variance, correlation coefficients or the Chi-squared, in order to have useful data to derive some conclusions.
The second techniques, however, is the *Feature Engineering* that allows to generate new features from those already existing, applying some transformation or specific calculations [45]. This method, combined with the previous one, allows to increase the accuracy of prediction, creating new variables that can bring more information to the model, slightly increasing its complexity. Usually, therefore, a certain trade-off between the two operations is needed so it is possible to get the best possible result without having to increase too much the waiting time and the required resources.

# 3. Case study and proposed methodology

Inside this chapter the steps of the KDD process are analysed in order to obtain appreciable results of pending time prediction for each individual job submitted.
The first section describes the database and, subsequently, all the attributes present inside the original dataset are discussed, to understand the use within this thesis project. In the second section all the individual steps of the data cleaning process performed on the data are explained, in order to clean them up of all the possible outliers present. Additionally, it will also analyse the feature engineering process used to extrapolate information, from the starting data, that are able to describe the context at the exact moment of submission of each individual job on the architecture. The third section deals with all the statistical and graphic analyses carried out both on the accounting data, present within the source data, and on the context variables, in order to broaden the overall view of the available data and to understand its intrinsic aspects. Finally, the fourth section explains the choices made to conduct the various experiments, reported in chapter 4, and illustrates the best solution obtained to be able to predict the pending time with the greatest possible accuracy.

## 3.1 The proposed methodology

The proposed methodology follows the KDD main steps, starting from the analysis of the available data up to the implementation of a model able to predict, more or less precisely, the time interval of the target variable.

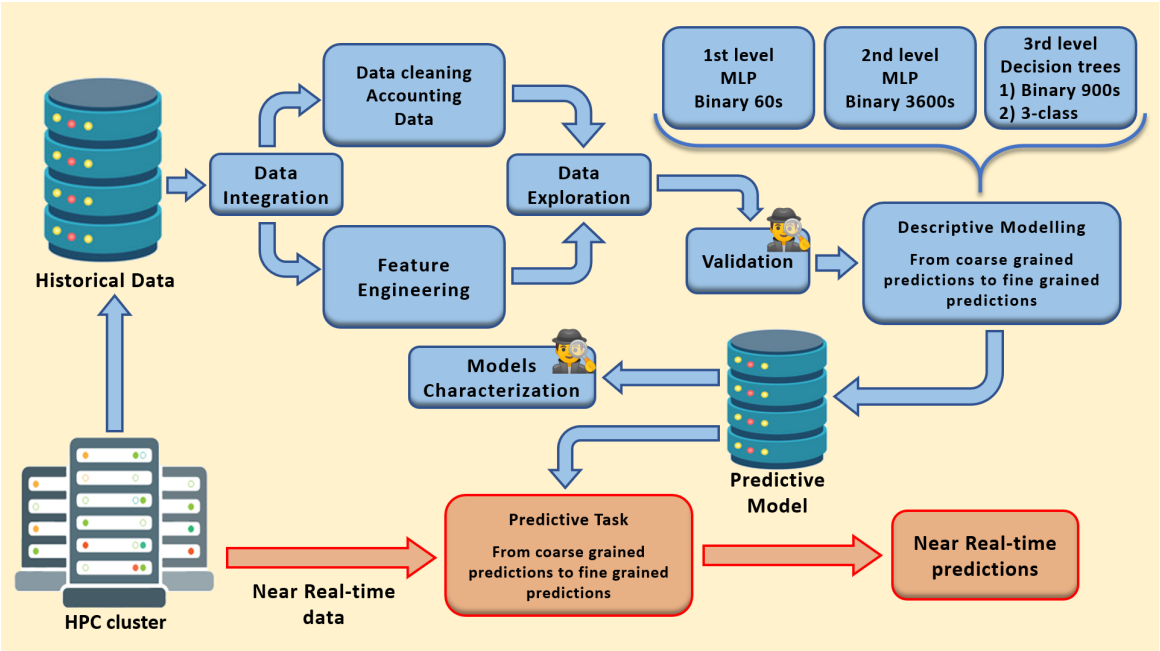



**Figure 3.1.** Schema of the proposed methodology.

In figure 3.1 the steps that have allowed a first realization of the prediction model, objective of the thesis plan, are reported.
As it is possible to see this schema follows a precise colour code, grouping the blocks in two

main categories: the blue phases belong to the off-line process, that is executed with data previously collected and historicized, while those in orange characterize the online process, which works on data created in real time. However, the model is not yet integrated into the management processes of the HPC cluster Iveco but, in case of new and well-structured data, the implemented algorithm would still be able to return a response.

Given the small amount of information on the data made available by the company, Data Integration, Data Cleaning, Feature Engineering and Data Exploration operations have been carried out which are extremely useful respectively for organizing and cleaning up the data, extract new ones and then perform statistical analyses to get a broader view of the problem. Then we moved on to the experimental phase, called Validation in the schema reported in figure 3.1, where several tests were carried out to understand the behaviour of ML algorithms. Thanks to this phase we arrived at the implementation of the model that will be later presented in section 3.5.1.

## 3.2 Data Structures

In order to carry out this thesis activity, the company provided a database containing various tables having inside all the information about the hardware components of the different HPC cluster, the submission of jobs and also other details about both hardware and software resources (such as the licences required for the job execution). Among all the tables, it has been considered only the one with the accounting data defined by all those variables useful to characterize the submission of a job within the HPC cluster Iveco, which are reported and described below:

- ***job_id***: is equivalent to a sequential number assigned by PBS to a specific job when it is submitted to the HPC cluster. As will be explained in section 3.3, this attribute is part of the primary key that will be assigned to the dataset.
- ***user***: is a unique alphanumeric identifier by which it is possible to recognize the user who requested the submission of the job.
- ***ncpu***: represents the number of CPUs that the user requires during the job submission procedure to run it.
- ***que***: is equivalent to the user-specified queue on which to execute the job. Users choose different queues based on the discipline to which the job belongs. The technical characteristics of each queue, and thus the distribution of the hardware resources of the Iveco HPC cluster, have already been reported in the table 1.1.
- ***submit, start*** *and* ***end***: represent respectively the dates of submission, start and end of the job in UNIX format, indicating the number of seconds elapsed from the time of Epoch (January 1, 1970).
- ***d_submit, d_start*** *and* ***d_end***: are the equivalent of the times previously described but represented as a formatted string. Then, in section 3.3, it will also be explained why d_submit has been chosen as the second attribute that, together with job_id, composes the primary key of the dataset.
- ***pend_time***: is the time between the moment the job is submitted and the one it is executed. Within this thesis it represents the target variable that the algorithm will try to predict in the best way possible.

- ***run_ time***: is the period of time the job appears in a running state on the HPC cluster, ending when the results are returned to the user.

- ***whole_ time***: is the total time between the moment the job is submitted and the one when the results are returned to the user. In essence, it is the sum of the two pend_time and run_time variables described above.

- ***dim_ inp*** *and* ***dim_ out***: are attributes which, technically, should contain respectively the size of the input files and the output results.

- ***cluster***: is the tag associated with the HPC cluster used, since the company has several HPCCs, in such a way there is the possibility to track jobs performed on specific clusters.

- ***softname***: represents the name of the software with which the user wants the job to be executed. Usually, the choice of the solver to be used is highly connected to the type of discipline to which the job belongs and, consequently, to the queue on which it will be launched.

- ***numlic0, numlic1*** *and* ***numlic2***:they indicate how many licences, belonging to a specific type, are used to ensure that the job can be executed correctly.

Only some variables listed above will be used specifically to try to achieve the intended purpose, other ones will be totally ignored or even deleted while others could be used in future works to complete the entire project that provides for the optimization of both hardware and software resources of the HPCC and also of the scheduling of jobs in the various queues.

## 3.3 Semi-Supervised Data Labelling

Starting from the dataset provided, containing 105 thousand records, some data cleaning operations were necessary to delete some possible errors within the data. In a first phase we tried to better understand the type of data available and make it easier to identify possible outliers or entire records that did not provide useful information for the purpose of the project and that could even compromise its final results.
Following, there are all the steps implemented for the data cleaning procedure represented within figure 3.1:

1. The first concerns the removal of entire attributes that do not provide any useful detail for the prediction of the target variable. The first two deleted variables, as mentioned in section 3.2, are dim_inp and dim_out, because, for each record, they always present the same value, that is 0. This is because, initially, with these attributes there was the intention to save the size of the input files and their relative output ones, but then the procedure to perform these operations was no longer implemented and the variables remained meaningless.
   A further attribute deleted from the dataset is cluster, which contains the name of the system on which the job was launched. The problem is that the records within the table taken into consideration were all launched on the same HPCC and therefore the tag is constant for each record. In fact, the differentiation between records, based on the architecture that was used for execution, was made at database level, using different tables containing jobs executed on specific clusters.

2. Three other variables which do not provide any useful information are numlic0, numlic1 and numlic2. As explained in section 3.2, these attributes contain the number of licences of three different types used to execute the job. The main problem, however, is that only a small part of these records is associated with the correct values, while most have only 0 as it. This is due to a bug in the procedure used to save this data, that can find this information only for specific solvers.
   These variables have not been completely eliminated, as the three previous ones, because, although they do not provide information for the case study analysed in this thesis and present a majority of outliers, it may be possible in the future to make changes by going to search for the correct values directly in the files generated by PBS during the submission of jobs (also called "raw" files because they save the data in a very compact format that is also difficult to read). For these reasons, these three variables will not be considered at this stage.

3. After performing some graphs to better understand the type of variables available, it was noted that, as regards the softname attribute, the categorical values contained were case-sensitive strings when it was expected to find unique names for each software available. To solve this problem all the values of this variable have been transformed to the corresponding ones containing only lowercase characters. These changes, in addition to others that will be described below, can be noted in the figure 3.3, where the situation before and after the data cleaning process is reported.

4. Again with regard to the softname attribute, there were two other cases on which it was necessary to take action. The first concerns the presence of records associated with software no longer used, such as fire and permas. In fact, all the records having one of this software as the value of the variable have been deleted, being no longer used by the used and not bringing, therefore, useful information for the development of the plan. The second case concerns all records having as softname value the string "undefined". The data, in the first period of use of the system (2014-2016), had associated this string when a software was requested whose name was not present in a specific list. Initially, we tried to save these records by defining specific conditions based on the launch queue, trying to assign the real name of the solver who had executed the job. The problem found is that, going on, it was noticed that these rules could not generalize well all the present cases, not giving therefore the certainty that the assigned software was really the used one, with the risk of adding an error component to the prediction system. So, later, it was decided to delete all these records, cancelling a good part of the original dataset.

5. Also with regard to the que attribute have been carried out cleaning operations similar to those previously analysed for the softname. In fact, following the exploratory analyses and the graphs in the figure 3.2 reported in section 3.4.1, were deleted all those records presenting as value the strings "permas_bm", "fast" and "workq", as these queues are no longer present within the architecture now being considered. In addition, it was also necessary to take actions for a further possible value of this variable, that is "normal". This concerns the first year of use of the architecture (2014-2015) in which, initially, there was no subdivision of resources by queues and then there was a single queue, subsequently renamed "normal". As a result, the tuples associated with the latter case have also been eliminated, as it was not possible to create conditions such that these data could get back inside the correct values.

6. By carrying out exploratory analyses within the dataset, two different cases of data duplication were also noted. The first concerns the exact copy of all the attributes of some

records repeated several times. This was due to an error in the data collection script that did not perform any check during the database saving operations. To overcome this problem only one tuple per group was held, deleting the other duplicate records. The second case is the duplication of only job_id values that were initially believed to be unique within the table. Analysing this situation, it was found that this attribute was not considered unique, as it was a value assigned by PBS, whose variable was easily alterable in case of technical problems. In fact, the problem of this last duplication occurred due to a shutdown of the system in the first half of 2015 that reset the PBS counter. Since all the records affected by this error technically could be considered as useful data, it was decided to use as primary key of the table the pair (job_id, d_submit), as announced in the section 3.2, thus eliminating the problem of non-uniqueness in the identification of records.

7. During the descriptive analyses carried out for each variable of interest, discussed in the section 3.4.1, it was noted that some records in the dataset had $run\_time = 0$. This was a problem because it implied that the job had never been executed. In fact, later, it was understood that these situations were due to jobs performed on nodes with technical problems, which were seen by PBS as free nodes, and therefore usable in the scheduling phase, but which, in truth, were not able to perform any operation.
   Continuing the exploratory analysis specifically for this attribute it was found that many tuples had extremely low run_time values, a sign that the solver stopped the execution due to errors within the model to simulate, as confirmed by users. To overcome this problem, considering that there was no variable available (even in PBS raw files) that could say with certainty whether the execution of the job had ended correctly or not, it has been defined, together with the users, a reasonable threshold such that jobs with run_time less than that value could be considered as wrong. This threshold has been set at 900 seconds (15 minutes), as, for execution times below this limit, users are used to launch simulations locally instead of occupying the resources of the HPC cluster.

8. The last step of the data cleaning procedure was designed to find out, in some way, which jobs were successful and those that had encountered errors both at the software level (errors in the model) and at the hardware level. During this search it has been noticed that in the raw files of PBS exists a variable, called Exit_status, to which are associated numerical codes able to put in relation the single job with a specific event (the meaning of each code can be found online at the NASA page [35]). Although these codes did not have the meaning we hoped for (that is categorizing the job in general as concluded correctly or not), as they only explain the work done by PBS but do not give information on the execution by the solver, through this step it was possible to delete only a part of the jobs that did not return the expected results. In addition, all the jobs performed in 2014 and the first half of 2015 were completely deleted due to the lack of PBS raw files for this period, thus making impossible to find this information.

Starting from about 105 thousand records, after having applied all these data cleaning operations, we have arrived at a fairly clean dataset containing a little more than 25 thousand records, minimizing the possibility of prediction errors caused by external contamination. Among these steps, the one that most contributed to this marked cut inside the dataset was the threshold set at 900 seconds with regard to run_time values that, alone, allowed the deletion of more than half of the data available. The second most useful step in terms of cleanliness was the elimination of $sofname = "undefined"$ which counted about 27 thousand records, of which, however, 16 thousand already met the condition of the 15 minutes on run_time.

Inside the database, to make some clarity on the available hardware resources for each individual queue, has also been added a table containing all the relevant information, which have already been reported in the table 1.1, so to have a fixed and stable reference.

The main problem, however, after performing the entire cleaning procedure, is that in the current table there is no reference to variables that can describe the status of the HPC cluster at the submission time of each individual job. Having only the target variable available but not the input attributes, it is extremely difficult to reach the goal set for this thesis. Therefore, a feature engineering process, as represented in figure 3.1, was carried out to compute variables able to define the context of the architecture when a job is taken over by PBS.
Following are reported the new variables identified:

- ***jobs_in_queue***: is the number of jobs that, when a new job is submitted, are waiting for resources to be able to run.
- ***jobs_running***: is the number of jobs that are considered running when a new job is submitted.
- ***time_in_que_tot***: considering all the jobs waiting for resources to be executed, it is the sum of the individual waiting times, that is the sum of the time intervals that go from when these jobs were submitted to the taken over, by PBS, of the new job. To simplify and clarify the concept, if at the time a job is submitted there are two jobs waiting, one from an hour and the other from 30 minutes, the value of this variable will be equal to 1 hour and 30 minutes, that is their sum.
- ***time_running_tot***: considering all the jobs defined in running, it represents the sum of their execution times, that is the sum of the time intervals that go from the moment in which these have been executed to the one in which the new job has been submitted on the HPC cluster.
- ***cores_available***: this attribute allows to identify the number of available cores on the HPCC, limited to the queue on which the job is executed. Always accept only integer values since the number of cores allocated to a system can never be decimal, being physical components.
- ***nodes_available***: defines the number of nodes currently available on the HPC cluster for the specific queue on which the job was executed. Depending on the queue considered, this attribute can take a decimal value since, in some cases, several jobs can be run consecutively on the same node.
- ***state***: this is a variable that can define what happened to the job at the time it was submitted. In fact, it can only take two types of values, "RUNNING" and "QUEUE", which, respectively, define whether the job will immediately get the resources it needs to be executed or whether it will have to wait for them to be released before being assigned to it.

The feature engineering process, to extract the variables listed above, uses information about both the queue on which the job was launched, since each has a different resource availability, and both submit, start and end times associated with its execution. To perform this process, however, almost all the available data were used, applying only part of the data cleaning operations previously analysed, as the jobs considered as wrong also occupied the resources of the HPC cluster and therefore conditioned the times of other jobs submitted later. More precisely, the data cleaning operations used also in this phase are the selection of the data with

a value of que not present in the table 1.1, the deletion of the columns dim_inp and dim_out, the transformation of all the softname values in order to obtain unique values for each usable software, the elimination of those jobs with run_time and pend_time equal to 0 and, finally, the deletion of duplicates explained in point 7 of the previous list.
The first part of the process iterates on all the que values available in the HPCC, allowing to select only jobs executed on a specific queue and to initialize the variables representing the technical specifications. After that, it is cycled on every single job selected, called $j_x$, and, for everyone, it is searched separately those jobs, submitted previously, that could be considered as running or waiting for resources. To identify the first category of jobs (those in running) the following conditions were used:

1. $d\_submit < d\_submit_{j_x}$
2. $d\_start < d\_submit_{j_x}$
3. $d\_end > d\_submit_{j_x}$

In this way it was possible to identify all those jobs launched before the job considered was submitted but concluded after that time. Instead, the following conditions were used for the selection of the records considered as in pending:

1. $d\_submit < d\_submit_{j_x}$
2. $d\_start > d\_submit_{j_x}$
3. $ncpu \leq ncpu_{j_x}$

The first one is the same as in the previous case, the second serves to define the waiting state of a job, while the last is used to take into account the execution policies defined by the company for the specific architecture. This is because, among the pending jobs, those that require less resources will always be executed first.
This procedure has allowed us to compute the context variables, previously analysed within this section, for about 90 thousand records of the original dataset, that is about 90% of the entire amount of data. Only later were selected the information belonging to those records considered corrected and whose primary keys were presented within the cleaned dataset.

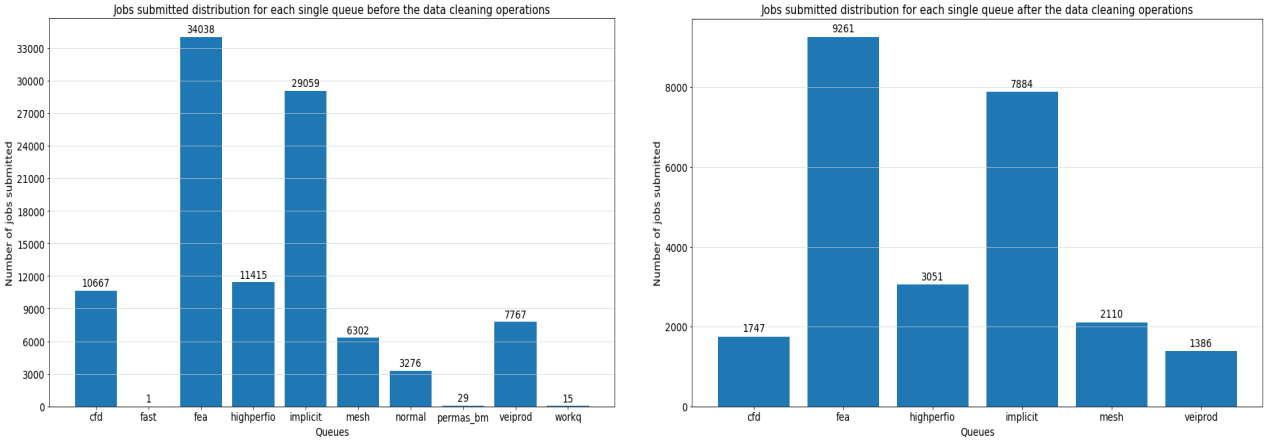
## 3.4 Data Exploration

The analysis phase of the available data is extremely important to understand in depth the information with which to work. As can be seen from the schema shown in figure 3.1, these operations have been carried out both on accounting and context data. With this process it is intended to extract as much knowledge as possible to be able, then, to use it when there is the need to make choices for the composition of the prediction algorithm. This extraction, in this case, takes place mainly through statistical analysis and graphical visualizations.

### 3.4.1 Analysis on accounting data

The first statistical analyses, after the data cleaning procedure, were carried out on the accounting data since they were available from the beginning.
We began to study the distribution of categorical variables, in this case represented by que and softname, to understand more deeply the type of data available to us. In the figures 3.2 and

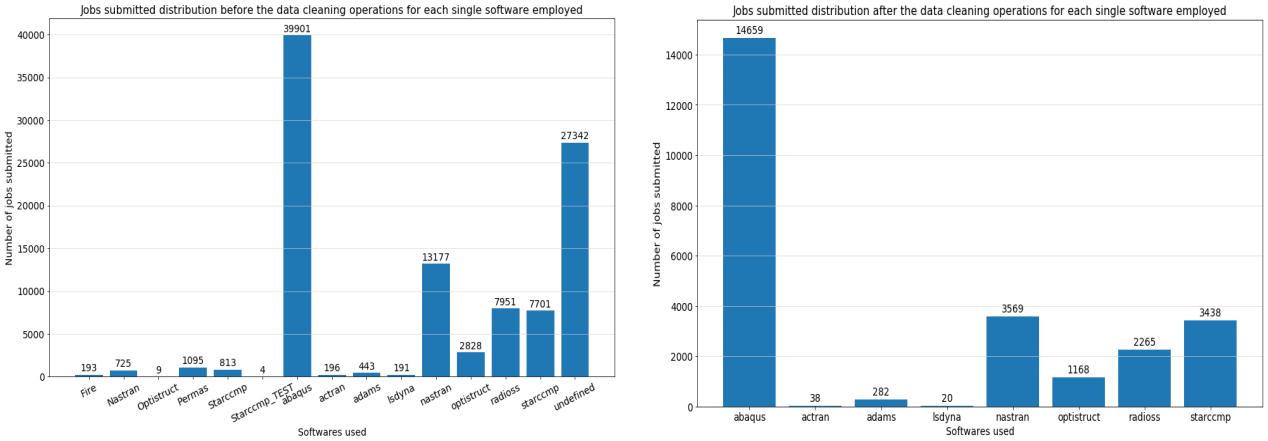
3.3 are reported the distribution of the attributes in question both before and after the data cleaning operations, so that it is possible to notice the changes discussed in the section 3.3.




(a) Jobs distribution by queue before data cleaning (b) Jobs distribution by queue after data cleaning

**Figure 3.2.** Distribution of jobs considering the different queues available both before and after data cleaning operations.

As for the distributions shown in figure 3.2, it can be noted that, despite the cleaning operations, the queues maintain more or less the same proportions, and therefore the frequency of use remains constant.
The same thing can be noted in the case of the distribution of software available, shown in figure 3.3, used for the execution of jobs. In fact, the cleaning operations halved the types of software available, but only because these were repeated.




(a) Jobs distribution by softname before data cleaning (b) Jobs distribution by softname after data cleaning

**Figure 3.3.** Distribution of jobs considering the different softwares available both before and after data cleaning operations.

From the graphs reported in the figure 3.3 it can also be noted how the application of the data cleaning has caused the drastic, but necessary, cancellation of 27 thousand records united by the value “undefined”. Despite this, also for the variable softname it can be noticed that the proportions do not change and, therefore, also in this case the frequencies of use of the software for the execution of the jobs remain constant.

After having extrapolated information from the categorical variables available, it was decided to study the level of correlation present among the numerical variables belonging to the dataset, and, to do this, it was necessary to examine its distribution since this is a fundamental

condition for deciding the correlation index most suited to the situation under analysis. Precisely for this reason it was decided to use the Shapiro-Wilk normal test, analysed in the section 2.2.3 since it is the one generally most in use. Through this test the results of $W$ and p-value are obtained as shown in the table 3.1.

**Table 3.1.** Results of the Shapiro-Wilk normality test for some accounting variables

| Variable | $W$ | p-value |
|---|---|---|
| ncpu | 0.6766 | 0 |
| submit | 0.9725 | 0 |
| start | 0.9726 | 0 |
| end | 0.9725 | 0 |
| run_time | 0.3261 | 0 |
| pend_time | 0.2577 | 0 |
| whole_time | 0.3777 | 0 |
| numlic0 | 0.7703 | 0 |
| numlic1 | 0.4039 | 0 |
| numlic2 | 0.0158 | 0 |

Looking at the results of this test we would say that the null hypothesis, that is that the distribution of data for the these variables is normal, is completely rejected since the p-value is 0 for all variables and therefore below the threshold value (by convention equal to 0.05) indicating the level of significance of the test. The problem is that one of the conditions of the Shapiro-Wilk test, regarding the dataset's size, in this case is violated as there are more than 5000 records inside the dataset. To overcome this problem the distribution charts of the variables of interest were generated, shown in figure 3.4.

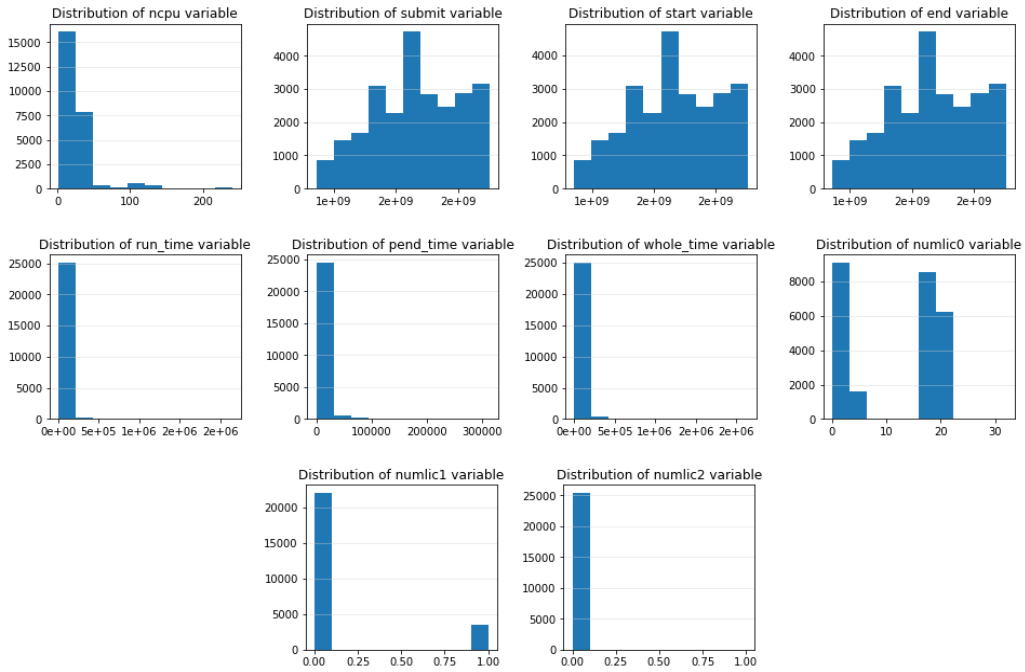



**Figure 3.4.** Distribution of the most important accounting variables.

As can be seen, these graphs confirm the assumptions previously made through the p-value. From the representations none of the variables has a distribution that follows the normal trend. From this information it was understood that, in order to calculate the correlation matrix

between the numerical variables, it was not possible to use the Pearson coefficient, which sets as a condition the normal distribution of the variables on which the computation is to be carried out. Therefore, between the Kendall ranks coefficient and the one of Spearman it was decided to use the first, given its reliability, thus obtaining the matrix shown in the figure 3.5. The values reported were derived from the data after the cleaning procedure, to find possible real relationships and not caused by the effect of outliers.

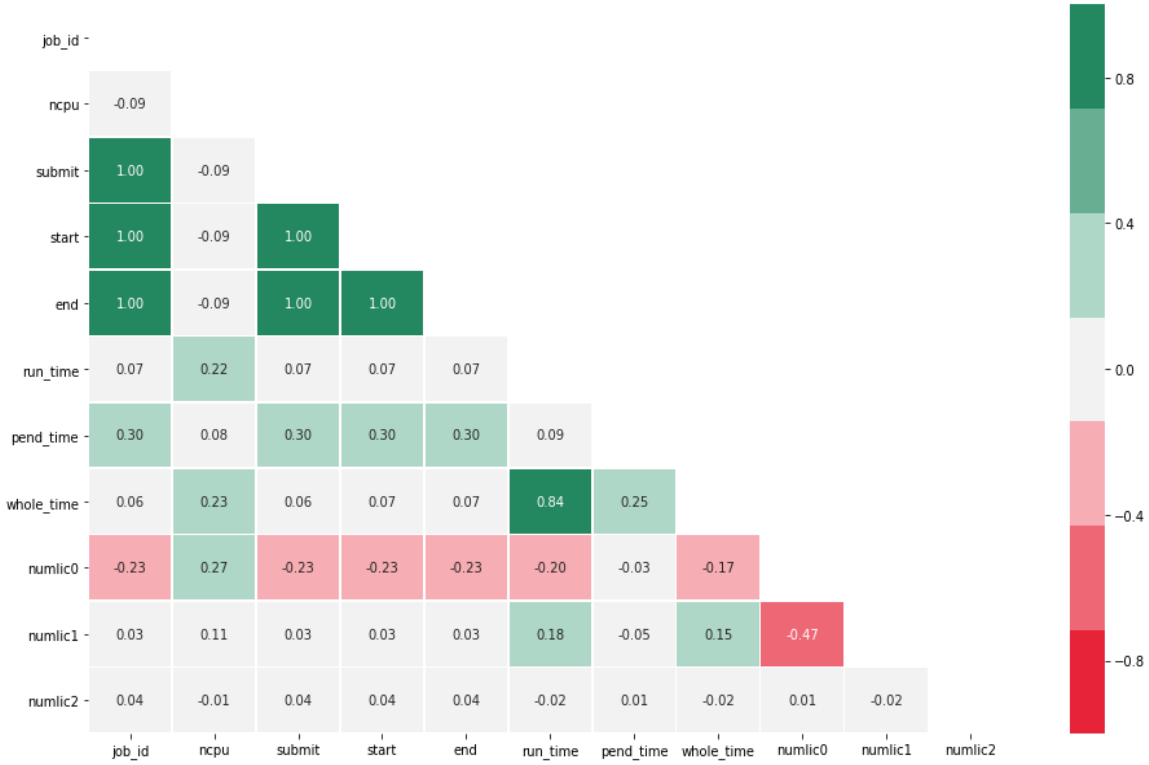



**Figure 3.5.** Correlation matrix representing the level of correlation between couples of accounting variables.

These values show how the submit, start and end times are extremely interrelated. In addition, these three variables are also perfectly correlated with the job_id, a sign that, after cleaning the data, there were no more failures that could have changed the value of the PBS counter, but instead made it a sequential and increasing value to every job submission.
Another relationship that jumps to the eye is the one present between the variables whole_time and run_time. This is quite normal since the first one is given by the sum of the second with the pend_time. In addition, the discrete correlation between whole_time and pend_time suggests that the values of this last one are generally lower than run_time values and therefore, in most cases, they will contribute less to the final whole_time value.

Considering that this type of matrix is used to display correlation indices for numerical variables, it was decided to calculate $\chi^2$ (Chi-squared) and its related Cramer's V for pairs of categorical attributes or containing discrete numerical variables. The results of this statistic, reported in the table 3.2, always refer to the data cleaned up.

**Table 3.2.** Results of the Chi-squared index and its relative Cramer's V for some couples of accounting variables

| Variable 1 | Variable 2 | $\chi^2$ | Cramer's V |
|---|---|---|---|
| que | softname | 49100.571 | 0.621 |
| ncpu | que | 46619.146 | 0.605 |
| softname | ncpu | 41968.350 | 0.485 |

From these values it emerges that all the pairs of variables present a medium correlation. The lowest of the three is the last, which means that there is no great relationship between the number of CPUs required and the software used. As for the second pair it is possible to see a slightly higher value for the value of Cramer's V, which fully reflects the situation since the number of CPUs is also required according to the technical characteristics of the specific queue, reported in the table 1.1, as well as depending on the complexity of the job to be executed. The result of the first pair, however, highlights a situation in which the rules, defined by the company, in which the solvers can use only some queues available according to the type of discipline to which the job belongs, are more or less respected.

For what regard the pair of mixed variables, that is those with a categorical attribute and a continuous numeric one (not discrete because otherwise $\chi^2$ with the relative Cramer's V would have been applied), the index of $\eta^2$ (Eta-squared) was used. The results of these analyses are reported in the table 3.3 and are always computed from the cleaned data.

**Table 3.3.** Results of the Eta-squared index for some couples of accounting variables

| Variable 1 | Variable 2 | $\eta^2$ |
|---|---|---|
| que | run_time | 0.067 |
| que | pend_time | 0.028 |
| que | whole_time | 0.064 |
| softname | run_time | 0.101 |
| softname | pend_time | 0.072 |
| softname | whole_time | 0.113 |

As can be seen from these values, none of the three continuous numerical variables taken into accounting has a relationship with the two categorical attributes. This means that the job times do not depend in any way on the queue on which it is launched or on the software used for its execution. Most likely, these attributes will depend on the properties of the model itself, which are not currently available, and the context of the HPC cluster characterized, nowadays, by the context variables described in section 3.3.

After the calculation of these indices, we wanted to search for the most used solvers and queues by users, to identify, considering the law of 80/20 analysed in section 2.2.1, those that would allow solving most of the problems by acting in a targeted way on a small sample. Therefore, Pareto's graphs have been realized both for the software used by the users and for the queues used for the execution of the jobs.

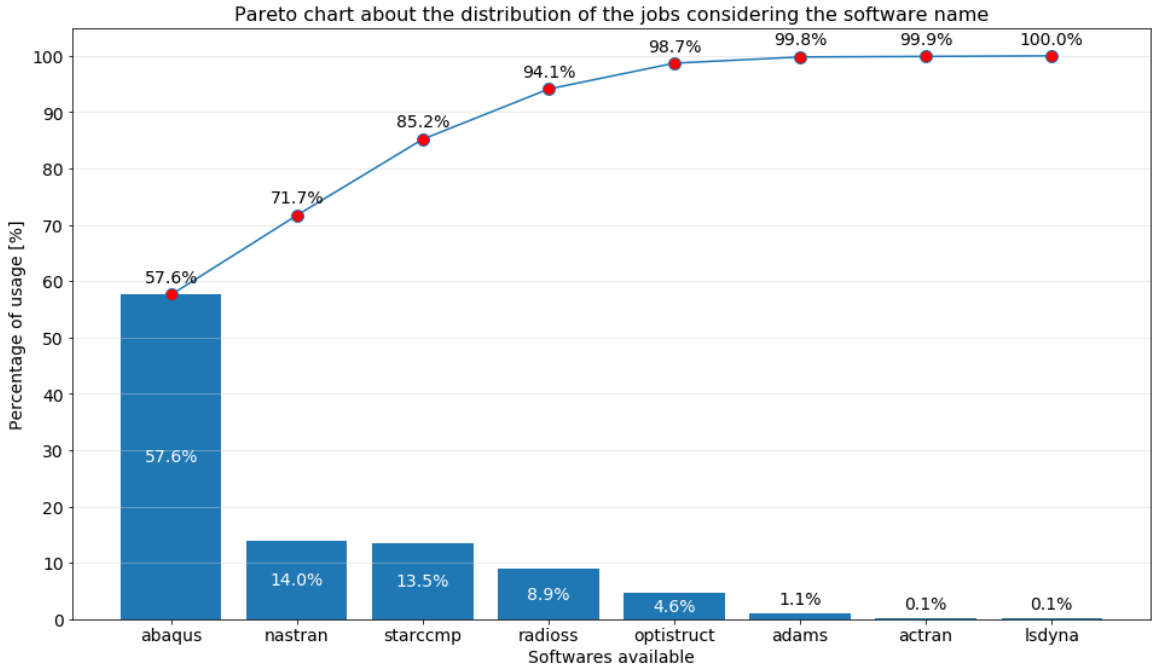



**Figure 3.6.** Pareto chart about submitted jobs considering the software employed for the execution of the operations.

In figure 3.6 it is possible to see the Pareto diagram specifically for the first case. From this representation we can notice that to satisfy about 80% of the problems we would have to act directly on the first three highlighted software, that are Abaqus, Nastran and Starccmp. With a similar reasoning we can also say that, always in this case, it would be pointless or unfruitful to solve problems for software such as Lsdyna, Actran and Adams which, together, are less than 2% of the performed work on the HPC cluster.

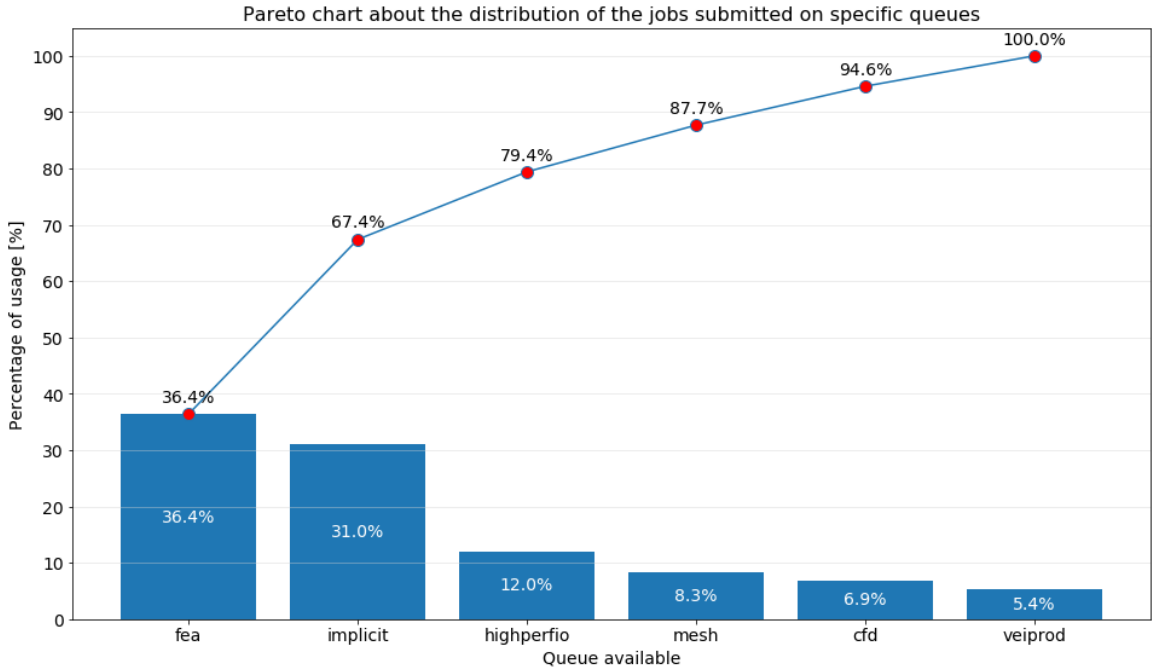



**Figure 3.7.** Pareto chart representing the distribution of the jobs considering the queue where they were submitted on.

About queues, however, the trend of the Pareto diagram, shown in figure 3.7, presents a quite

different situation from the one previously analysed. As can be seen, it is true that 80% of the cases are always concentrated in the first three classes, but, in this case, even the subsequent classes have an appreciable relevance compared to, instead, the result for the softname case. Later, when the implementation strategy of the prediction process was defined, it was decided to use all the jobs belonging to the cleaned dataset anyway, to avoid further resizing of the data.

After that, all variables that could provide useful information were selected, like ncpu, run_time, pend_time and whole_time, on which descriptive analyses were carried out to obtain the values of the most relevant position indices such as quartiles, mean, standard deviation, in addition to the minimum and maximum value of the variables.
This kind of analysis was performed both before and after data cleaning so that it is possible to make a comparison and make visible the changes made. In the table 3.4 are reported the statistics, associated with the selected attributes, computed on the data before performing the cleaning process.

**Table 3.4.** Results of the descriptive analysis using the accounting data before the execution of the data cleaning process

| Variable | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|
| ncpu | 23.5 | 25.5 | 1 | 16 | 16 | 32 | 240 |
| run_time | 9325.3 | 39618.8 | 0 | 64 | 660 | 2642 | 2144795 |
| pend_time | 2249.1 | 13671.9 | 0 | 6 | 14 | 41 | 867451 |
| whole_time | 11574.4 | 43771.1 | 0 | 104 | 846 | 3735 | 2144812 |

As mentioned in the section 3.3, it immediately catches the eye a value within the table 3.4 that it would be expected different. In fact, the minimum value equal to 0 associated to the run_time variable means that some jobs, present inside the dataset, have never entered in execution and, therefore, cannot be used for our purpose because they do not bring useful information. From this came the need to enter the $run\_time = 0$ condition to select all those records to be deleted. Another value that aroused, initially, some suspicion was the minimum value equal to 0 of the pend_time variable, because, at the time when the HPC cluster has the resources to run a job, PBS still needs a few seconds to select the nodes, initialize them and find the necessary licences. Despite this, users have reported that these cases are completely acceptable considering the cases when the PBS server is unloaded, thus performing these operations so quickly that can be considered as instantaneous.

**Table 3.5.** Results of the descriptive analysis using the accounting data after the execution of the data cleaning process

| Variable | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|
| ncpu | 26.3 | 27 | 1 | 16 | 16 | 32 | 240 |
| run_time | 19320.6 | 53059.7 | 900 | 1887 | 3743 | 12815 | 2144795 |
| pend_time | 3899.3 | 15812.8 | 2 | 9 | 24 | 582 | 313975 |
| whole_time | 23219.9 | 57263.4 | 904 | 2157 | 4883 | 16744.5 | 2144812 |

Comparing the results of the analyses obtained on the cleaned data, reported in the table 3.5, and those obtained from the starting data, reported in the table 3.4, shows how the values of some indices, concerning the ncpu variable, remain completely identical while two present

few changes. This means that data cleaning operations did not change the distribution of this attribute. It can also be noted that the minimum value of the run_time attribute respects the rules imposed, while, as for the pend_time, this value reflects the general situation in which the resources are immediately available, but that minimum amount of time is required to organize them and make them usable. An ulterior aspect to evidence is given from the fact that the quantiles referred to the three times change, but they are always distributed in the low part of the values associated to these variables, such that the 25°, 50° and 75° percentiles are smaller of the mean value (also low compared to the maximum value). Finally, we can see how the values of the indices for run_time and whole_time differ little, confirming therefore the hypothesis that, despite the cleaning process, the first variable contributes more in the final value of the second one.

To deepen the study of the distribution of the values of pending time a clustering technique has been applied. In fact, these data have been divided into nine separate clusters, visible in the figure 3.8, to facilitate the understanding of the knowledge contained inside.

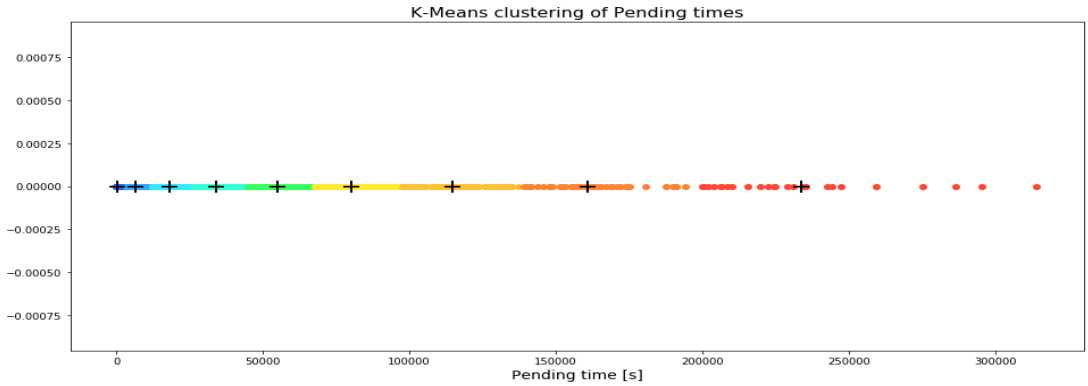
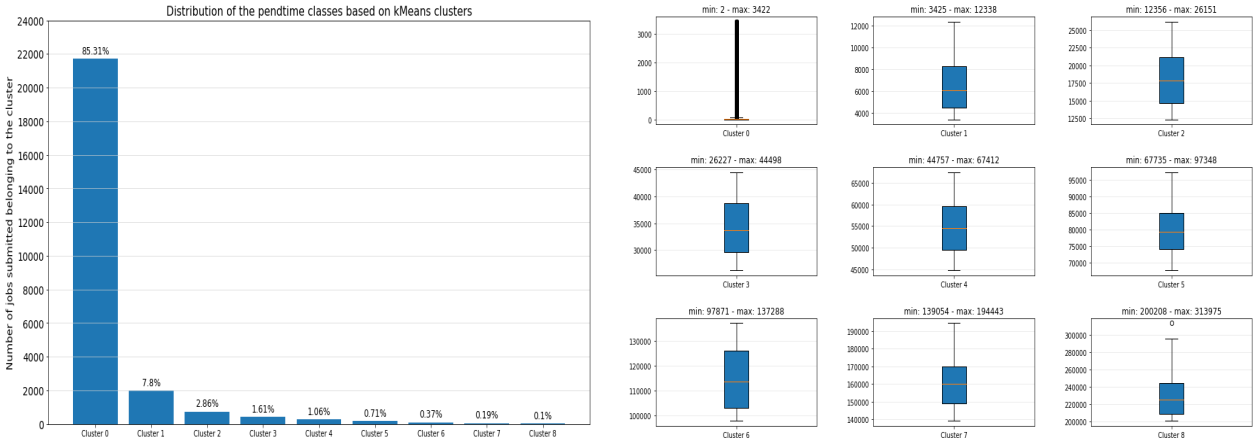



**Figure 3.8.** Visualization of the 9 clusters created to analyse the distributtion of the pending time values.

In the graph are represented the centroids (indicated with +) and, with different colours, all the records belonging to the different clusters created. Through this information it was possible to generate the bar chart and the boxplots shown in the figure 3.9, thus allowing a more in-depth analysis.

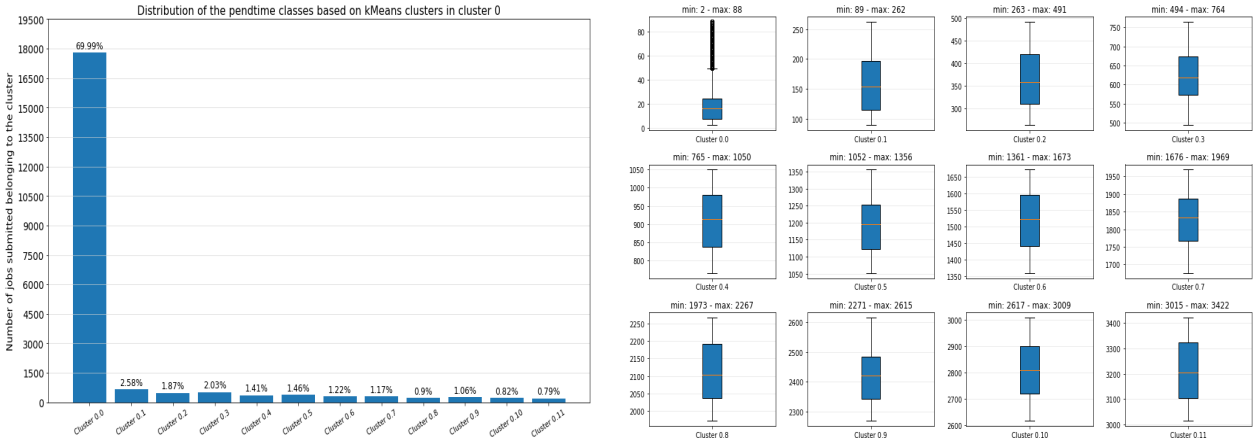



(a) Bar chart representing the absolute and relative frequency of each cluster.

(b) Boxplots representing the distribution of the values of each cluster.

**Figure 3.9.** Representations of the clustering process over the pend_time attribute.

As it is possible to see in the var chart shown in the figure 3.9(a), the vast majority (just over 85%) of the available data fall within the first cluster. From the relative boxplot, present in figure 3.9(b), we can notice another important aspect, that is the excessive presence of outliers inside the cluster, which means that the boxplot is completely crushed on the low values of the graph. From this it can be inferred that the descriptive indexes related to the cluster 0 are all concentrated on the bottom side of the range considered, which is confirmed by the data in the table 3.5.
About the other clusters, one information that can be derived is that most of the values belonging to each cluster tend to be concentrated all around the median. From the analysis of the boxplots, in addition to the dispersion of the data around the median, it is also clear that there is a higher frequency of the medium-low values within the clusters, which explains the downward displacement of the box. This confirms the presence of a positive asymmetry, as already highlighted by the bar chart in figure 3.9(a).
Following the data previously analysed, it was decided to carry out a further process of clustering specifically on the data belonging to cluster 0. This time, to try to obtain a better distribution, it was decided to use twelve clusters instead of nine. The results extracted from this further analysis are shown in figure 3.10.




(a) Bar chart representing the absolute and relative frequencies of each sub-cluster.

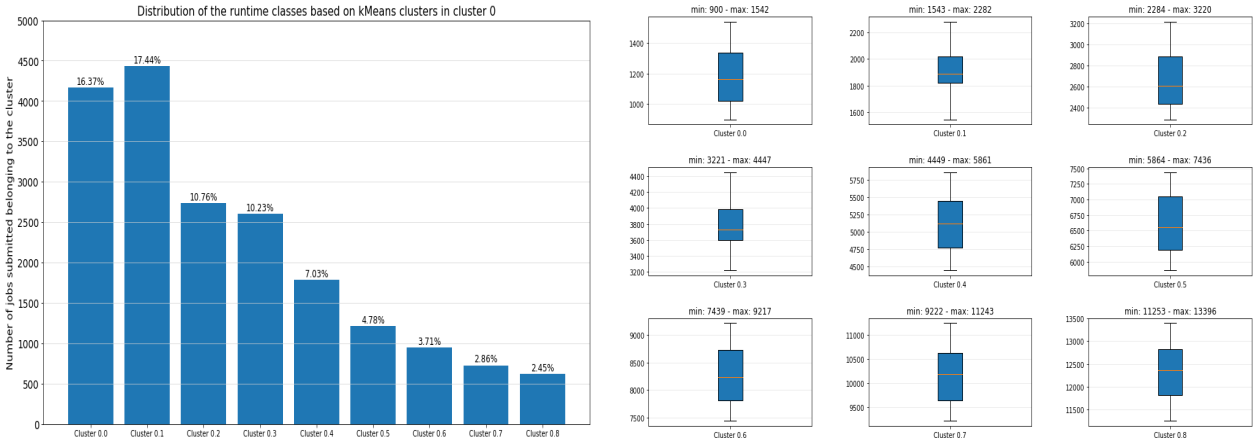
(b) Boxplots representing the distribution of the values of each sub-cluster.

**Figure 3.10.** Representations of the clustering process over the values belonging to cluster 0 of the pend_time attribute.

As can be seen from the bar chart in figure 3.10(a), the situation was quite similar to the previous case. In fact, about 70% of the total data are grouped in the lower part of the interval (belonging therefore to cluster 0.0 as represented in figure 3.10(a)). In addition, it is important to point out that the percentages reported on the bar chart columns refer to the relative frequency calculated with respect to the entire dataset and not only to the data belonging to cluster 0.
From the boxplots shown in the figure 3.10(b) it can be observed that, although the data have been further subdivided into sub-clusters, the cluster 0.0 still has outliers whose values are therefore higher than the descriptive index $Q_3$, as explained in section 2.2.1. As for the other clusters generated, they all have internal values to their descriptive indices and grouped around the median. In addition, the averages of these clusters tend to have values nearly to the medians.

The deepening of the study of the distribution through the clustering techniques has been successively applied also to the run_time attribute. Considering the general process applied to the entire dataset, the information found was practically identical to the information related to

the pend_time. Also for the run_time, the data were divided into nine clusters and just over 75% of them belonged to cluster 0. From this subdivision, the boxplot related to cluster 0 is not completely crushed, but it still has several outliers whose values are above the $Q_3$ threshold. As for the pend_time, also for the run_time, due to the high numerosity, it was decided to apply an additional clustering procedure only for the data belonging to cluster 0, but, compared to the analysis made for the first, in the second nine clusters seemed more than enough to recover some information quite accurate, as can be seen from the results reported in figure 3.11.




(a) Bar chart representing the absolute and relative frequencies of each sub-cluster.

(b) Boxplots representing the distribution of the values of each sub-cluster.

**Figure 3.11.** Representations of the clustering process over the values belonging to cluster 0 of the run_time attribute.

Also in the case of run_time, the percentages represented in the chart in figure 3.11(a) refer to the relative frequency calculated in relation to the total data available.
The scenario depicted by this further subdivision into nine sub-clusters is quite different from the situation previously seen for the pend_time. It is interesting to notice that the distribution of values is not concentrated entirely on a single group but divided into several clusters. This trend is confirmed by the relative boxplots, represented in the figure 3.11(b), which have no outliers and the data are grouped around the median. From the analysis of the boxplots it can be deduced not only the dispersion of the data around the median, but also how the distributions turn out to be asymmetrically positive, confirming what already highlighted by the histogram. In fact, there is a higher frequency of the medium-low values, which explains the downward of the box.

### 3.4.2 Analysis on context data

After the feature engineering procedure for the extrapolation of the context variables, statistical and graphical analyses were carried out on these data to obtain all possible information.
As previously done for accounting data, we wanted to analyse the level of correlation present between the numerical context attributes available within the dataset. Since a fundamental condition for the decision of the correlation index to use is the normality of the distribution of the variables, it was decided to apply the Shapiro-Wilk test first, discussed in section 2.2.3, thus obtaining the values of $W$ and p-value reported in the table 3.6.

**Table 3.6.** Results of the Shapiro-Wilk normality test for some context variables

| Variable | $W$ | p-value |
|---|---|---|
| ncpu | 0.6766 | 0 |
| jobs_in_queue | 0.2530 | 0 |
| jobs_running | 0.8920 | 0 |
| time_in_que_tot | 0.0977 | 0 |
| time_running_tot | 0.2376 | 0 |
| cores_available | 0.7733 | 0 |
| nodes_available | 0.7717 | 0 |
| pend_time | 0.2577 | 0 |

From the data found, looking only the value of the p-value for all the variables, it would be to say that the null hypothesis can be rejected since this value is inferior to the threshold value $\alpha$ (for convention equal to 0.05) that indicates the test significance level. However, this value cannot be taken into account as it is again violated one of the basic conditions associated with the Shapiro-Wilk test, that is the dimensionality of the dataset, which must not exceed the threshold of 5000 records. So, to overcome this problem, the distribution graphs of the context variables were executed, obtaining the representations shown in figure 3.12.

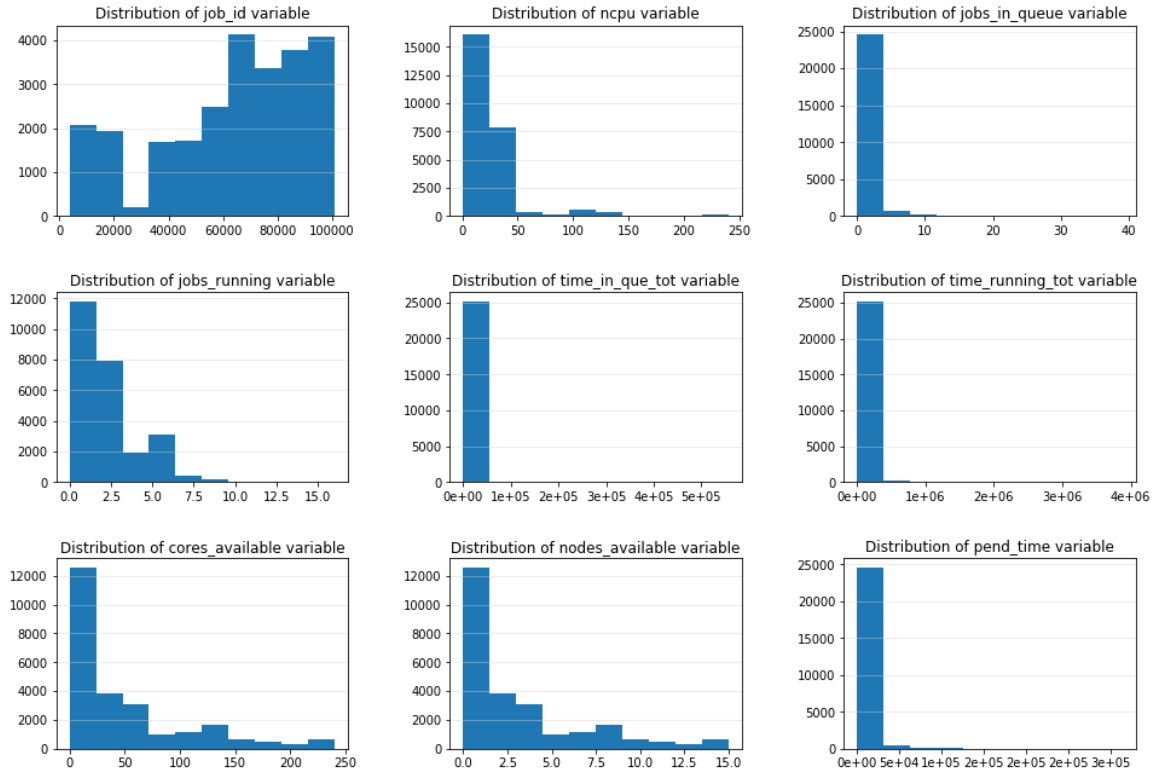



**Figure 3.12.** Distributions of the context variables.

As can be seen, these graphs confirm, to a large extent, the assumptions made previously through the p-value, showing that none of the variables of interest has a distribution that follows a normal trend. From this it was concluded that it was not possible to use the Pearson correlation index and, therefore, between the Kendall ranks correlation coefficient and the one of Spearman, it was decided to use the first one to maintain some continuity with the analyses

performed on the accounting data, in addition to the fact that it is the most reliable of the two. The correlation matrix obtained is shown in figure 3.13.

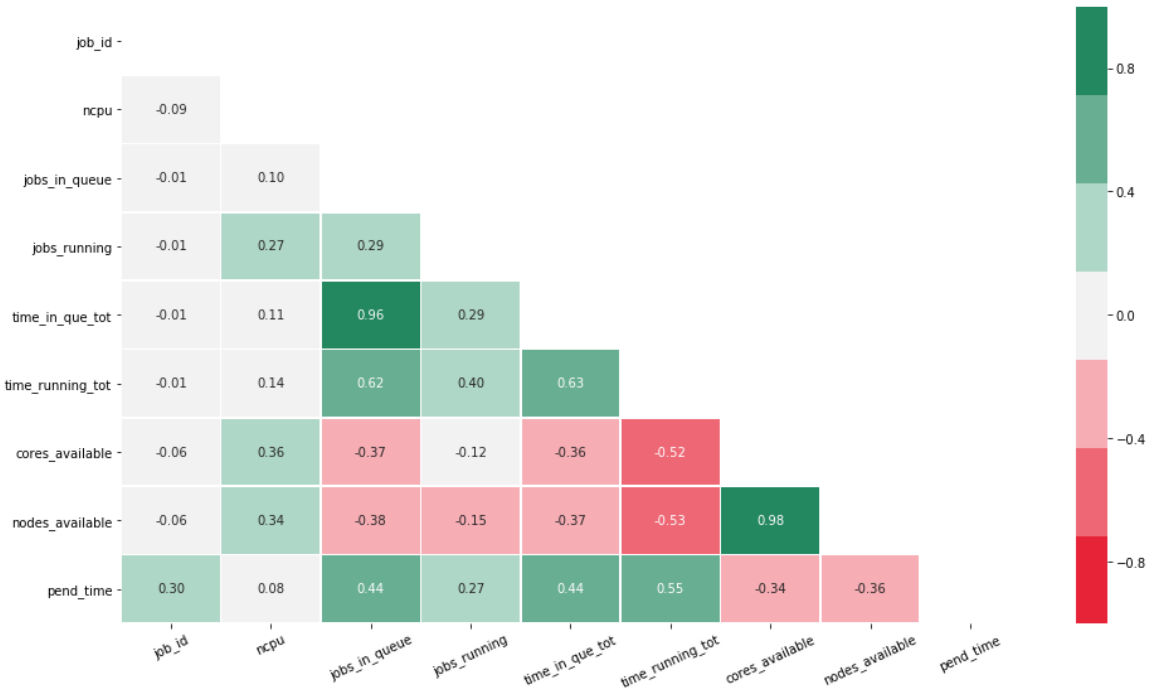



**Figure 3.13.** Correlation matrix representing the level of correlation between couples of context variables.

Within this representation, it is possible to see how the jobs_in_queue and time_in_que_tot variables are highly correlated with each other. Although this value may seem plausible, it is believed that this is altered by the presence of a bias, regarding the fact that many of the values of these two attributes are associated with 0. This is because, in feature engineering procedure analysed in section 3.3, at the time when a job was associated with the state "RUNNING", meaning that the resources necessary for its execution were immediately available, the jobs_in_queue and time_in_que_tot variables were assigned directly to 0. Since the number of jobs associated with the state "RUNNING" is about 19 thousand compared to the total, this may have caused this high correlation value between the two attributes.
Another extremely high value is the one present between the pair cores_available and nodes_available, which is entirely justified since, to compute both variables, a common feature is used based on the queue taken into analysis.

Since these matrices are used only in presence of continuous variables, in case of categorical attributes or discrete numerical one, $\chi^2$ (Chi-squared) is computed and, subsequently, the relative Cramer's V. the results of this two statistics are shown in the table 3.7 below.

**Table 3.7.** Results of the Chi-squared index and its relative Cramer's V for some couples of context variables

| Variable 1 | Variable 2 | $\chi^2$ | Cramer's V |
|---|---|---|---|
| que | cores_available | 49100.571 | 0.621 |
| que | state | 645.097 | 0.159 |
| que | jobs_in_queue | 999.697 | 0.089 |
| que | jobs_running | 16682.036 | 0.362 |

From these results it is possible to see that jobs_in_queue and jobs_running are not at all related to the queue on which the job is executed. The only medium relevant result is the one concerning the pair que and cores_available. This correlation is explained by the fact that the value of the second variable depends on the physical characteristics of the queue, shown in table 1.1, and therefore on the queue itself.

Subsequently, to highlight the level of relation present between mixed variables (containing a discrete numerical or a categorical variable and a continuous numerical one), the $\eta^2$ (Eta-squared) was computed, the results of which are shown in the table 3.8.

**Table 3.8.** Results of the Eta-squared index for some couples of context variables

| Variable 1 | Variable 2 | $\eta^2$ |
|---|---|---|
| que | time_in_que_tot | 0.011 |
| que | time_running_tot | 0.044 |
| que | time_tot | 0.047 |
| que | pend_time | 0.028 |

Through these values it can be noticed that all the four times, put in correlation with the type of queue on which the job is launched, have extremely low values of $\eta^2$, sign this of a high statistical independence between the attributes considered.

Finally, it was decided to better analyse the distributions of the two discrete numerical variables, namely jobs_in_queue and jobs_running, to try to extract useful information to improve the prediction process. The charts for these distributions are shown below in the figures 3.14 and 3.15.

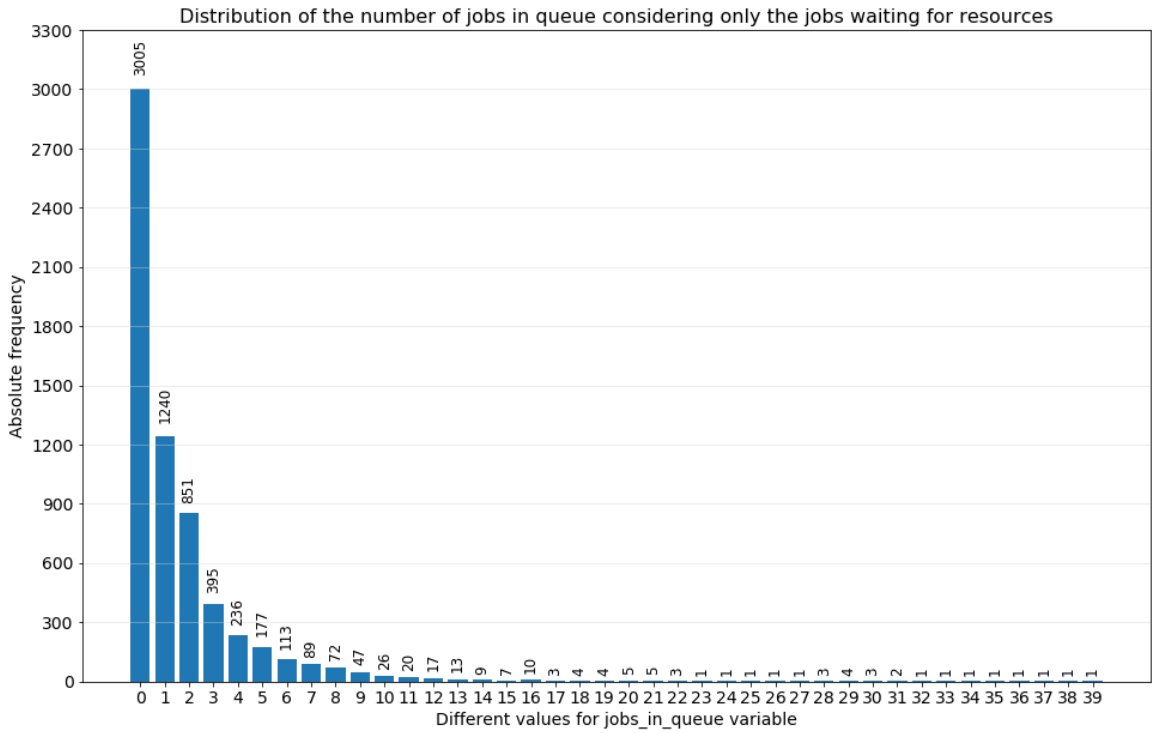



**Figure 3.14.** Distribution of the values of jobs in queue, representing how much records have a specific number of jobs waiting for resources when a new job is submitted.

From the data reported in figure 3.14 it can be noted that the frequency of appearance of a certain event (understood, in this case, like number of times in which it has been found the same number of jobs waiting for the resources) decreases to the increment of the number of job in standby.

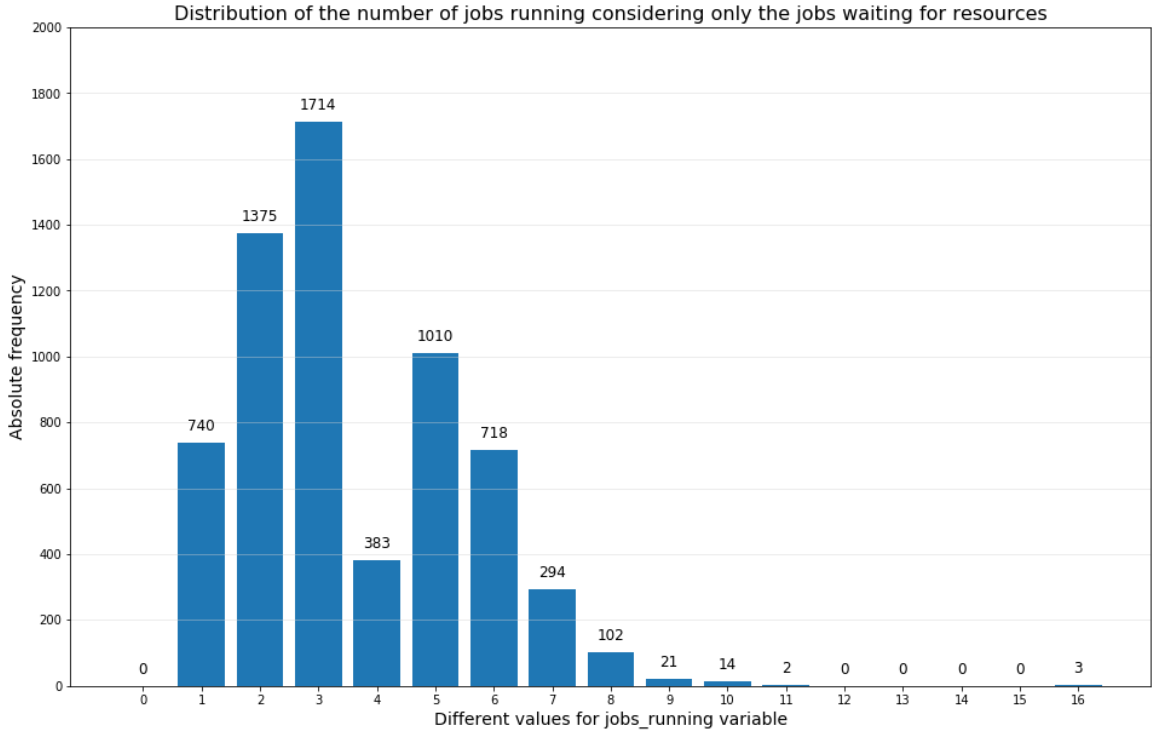



**Figure 3.15.** Distribution of the values of jobs running, representing how much records have a specific number of jobs using resources to execute specific operations when a new job is submitted.

Indeed, from the data reported in figure 3.15, it is possible to see a completely different trend from the one associated to the attribute previously considered. Through this representation, in fact, there is a more discontinuous trend that, compared to the first, is somewhat reminiscent of the one of a Gaussian, thing that is confirmed also by the relatively high value of $W$ inside the table 3.6.
The data used for these last two representations consider only those jobs that have associated to the variable state the value "QUEUE". This is because, for the jobs considered to be immediately in running ($state = "RUNNING"$), the value 0 has been assigned to the variables in question and to the time_in_que_tot, since the information made, in this case, for the prediction of pending time were entirely irrelevant. In fact, the pending time, for the jobs that have undergone the resources necessary for its execution, currently depends only on the variable cores_available (and, consequently, also nodes_available, since the two are highly correlated) and none of the other variables computed by the feature engineering process.

## 3.5 Classification Strategies

In this section, firstly, the classification model, implemented for the prediction of the time interval to which belongs the pending time of a job, is analysed.
Subsequently, the steps implemented in the classification field will be described and justified, as in the case of the discretization type (by frequency or by interval width) that are possible to be applied. In addition, particular attention will be paid to the selection process of classification

algorithms also emphasizing the limits that each of them presents to justify the choices made as part of the best solution among those analysed in the section 4.2.

### 3.5.1 The classification approach

The following is an analysis of the best approach identified as result of the various experiments carried out, which are reported in section 4.2.
The solution found is part of the category known as hierarchical classification in which more algorithms are used to predict the class of belonging. In fact, the system created, whose schema is shown in figure 3.16, is composed of several prediction levels in which the training of algorithms depends on the class predicted in the previous level. Only correctly classified records are reused for the algorithm training in later levels, resulting in a physiological decrease of records each time you move to a following level (since it is not possible to expect a 100% correct prediction). The classes' characterization for each classification level is experimental-driven, meaning that the choice of thresholds is based entirely on the results of experiments previously conducted, the results of which are reported in the section 4.2.
For each type of classification used to define this prediction system the following input variables have been used: que, ncpu, jobs_in_queue, jobs_running, time_in_que_tot, time_running_tot and cores_available. Obviously, the que, before being used for the classification process, has been numerically categorized.

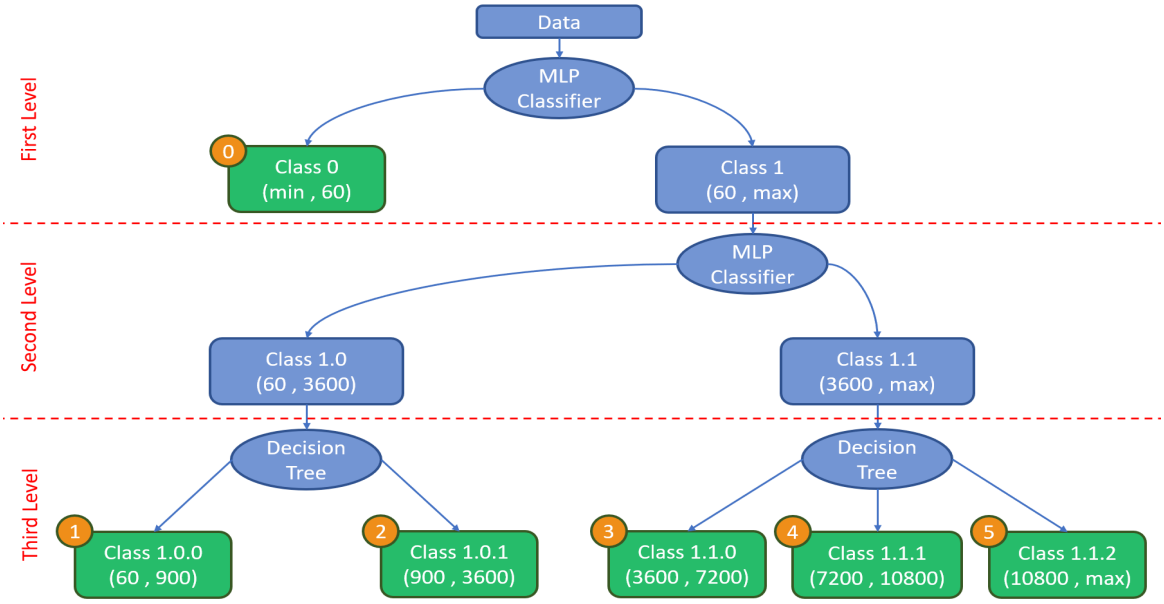



**Figure 3.16.** Schema of the best classification solution found.

In figure 3.16 we try to schematize, in the best way possible, the operation of the hierarchical process with which the best results have been obtained. Within this image are presented the algorithms selected at each level to compose the final system, the ranges used, so that at each level there are the best results, and, in the end, the final prediction classes associated with their ranges (green boxes with orange labels). In essence, are depicted all the steps that the system performs to get assign the most appropriate class, given the input variables provided.
The final result can be seen as the graph of a decision tree, in which, based on the values of certain variables, it is possible to choose the path to follow to reach the prediction class. In this case, the input attributes for this tree are the values of the predicted classes in the previous classification levels.

The first decision concerns the binary classification threshold applied in the first level. Indeed, the 60 seconds threshold was chosen considering that, among all the binary experiments performed and discussed in section 4.2.3, this was the one that presented the best results. After that, analysing the results obtained from the evaluation of the models, the values of which are reported in the table 4.28, the possible choices were the KNN or the MLP. With the same initialization time, the MLP allows a better prediction in real time even for very complex functions and, for this, it was chosen compared to the KNN.
As for the second level, first of all an evaluation of all the algorithms has been performed to understand how these worked with the data available in case there is the need to predict two intervals defined by the threshold of 3600 seconds, thus obtaining the data given in the table 4.11. This threshold has been designed in such a way that it can create specific classes that can give fairly precise feedback, at the level of timing, to users. From the obtained results it is evidenced that the MLP is the algorithm with the best values among all (confirmed also from the confusion matrices generated for each of them and brought back in the figure 4.4) which is why it was chosen for the prediction to be performed at the second level.
The third-level binary classification associated with records of class 1.0 was divided considering the results obtained in the prediction test where the threshold was set to 900 seconds. However, given that the boundary situation, in this case, was different from the experiment discussed in section 4.2.3.1, in order to decide which model was the most accurate in the assignment of records belonging to class 1.0, an evaluation of the algorithms has been carried out, obtaining both the results reported in table 4.12 and the matrices in figure 4.5. From the data obtained it is noted that the most effective model in the prediction of these two classes is the Decision Tree, whose choice is given by the fact that both classes were classified fairly well compared to the other algorithms, who tended to predict one class better that the other.
Finally, for the prediction of class 1.1 it was decided to apply a classification to 3 classes, to obtain 6 final classes of prediction, as happened in the experiment analysed in section 4.2.1.3 and also to not compromise too much the results related to this classification. All the algorithms selected to decide which was the most suitable for this last step of prediction were evaluated and the results reported in table 4.13 were obtained. Taking these values into account and studying the confusion matrices shown in figure 4.6, it was decided, also in this case, to use the Decision Tree for the third level classification of class 1.1, since it presents the best results in almost all indicators computed.

It is important to note that all the algorithms that make up the hierarchical model are interpretable, that is, starting from specific input values, it is clear the process for which a certain value is returned. This is implicit for the Decision Trees used in the third level as they are algorithms that can be interpreted by definition.
In the case, instead, of the two MLPs used for the classification of the first and second level, being these considered as black box, it was necessary an intermediate step of translation with the use of Decision Trees: the latter are provided with the input variables of each MLP, while as target variable, instead of using the real values, the predictions made by the MLP are used. Following these tests, the results obtained are extremely significant since the Decision Tree employed for the first level explains the MLP to 99%, while the one employed in the second level to 93%.

### 3.5.2 Discussion

Early evaluated classification algorithms included Logistic Regression, Perceptron, SVM, KNN, Decision Tree, Random Forest and MLP. The main characteristics that led to use of these models and also the problems encountered will be analysed below.

*Logistic Regression* is a linear classification algorithm that, although it is optimal in dichotomous classifications cases, it has been possible to use it by setting the multi_class parameter to multinomial, so that it can also be applied to predictions of more than two classes. As solver, "lbfgs" was selected. Usually, this is used for small datasets but comparing it with other types of solutors available, it was noticed that the values obtained did not show any significant changes. Therefore, it was decided to use it both for its robustness and for its speed of convergence even with medium-large datasets.
The *Perceptron* is a simple classification algorithm, also linear, which does not need a specific learning rate, does not use penalties and is updated only when it makes a mistake. For this, it is quite fast compared to other models but the results are usually not very accurate, as it is possible to see from the data in tables 3.9 and 3.10.
The *SVM* has been selected because it is a very effective nonlinear algorithm when dealing with multidimensional spaces and, in addition, requires little computing power and is easy to interpret the result. The problem is that it generates limited models in prediction capability despite the possibility of using various types of kernel, which does not make it usable in all situations.
The *KNN* is always a classification algorithm, selected for its robustness to outliers and for its effectiveness when dealing with large datasets. Moreover, it is a non-parametric tool, meaning it makes no assumptions about the distribution of the data it analyses. In other words, the structure of the model is determined by the data and is rather useful because, in the real word, most data does not obey to the typical theoretical assumptions (as happens, for example, in linear regression models). Despite this, it is very computational heavy and is difficult to implement as well as initialize. In fact, for this last operation, it is necessary to tune the main parameters, including the value $k$ of points to be considered as belonging to the neighbourhood.
*Decision Trees* are widely used models for their easy visualization and understandability but can create very large trees that are not able to generalize well the problem, as they are highly sensitive to small variations in the values of the input attributes.
*Random Forest* is an algorithm that is mainly based on decision trees and, compared to the these, tries to reduce overfitting, resulting, therefore, more accurate. However, being a combination of several trees is rather slow in real-time predictions.
Finally, The *MLP* has been selected for its ability to learn real-time models, regardless of their complexity and although the function may also be non-linear. However, given that its loss-function may have more local minimum, it is necessary to modify and adapt the weights of the hidden layers several times. Considering this problem and taking into account the fact that it is necessary a tuning phase for some parameters, such as the number of hidden layers and nodes that compose them, this kind of models presents a slow initialization process that however, once you find the right values for these features, it only runs once.

To be able to perform an accurate classification it was also necessary to decide the discretization method (also called binning) initially more appropriate to use. In fact, both the discretization for intervals of same width and for intervals of same frequency has been analysed, thus going to evaluate also the algorithms selected in such a way to use only those that had the best performance. This type of operation was applied by grouping the data into 20 different bins.
With the first type of binning (the one with intervals of same width) the diagram brought in figure 3.17 has been generated, to see the distribution of the various classes.

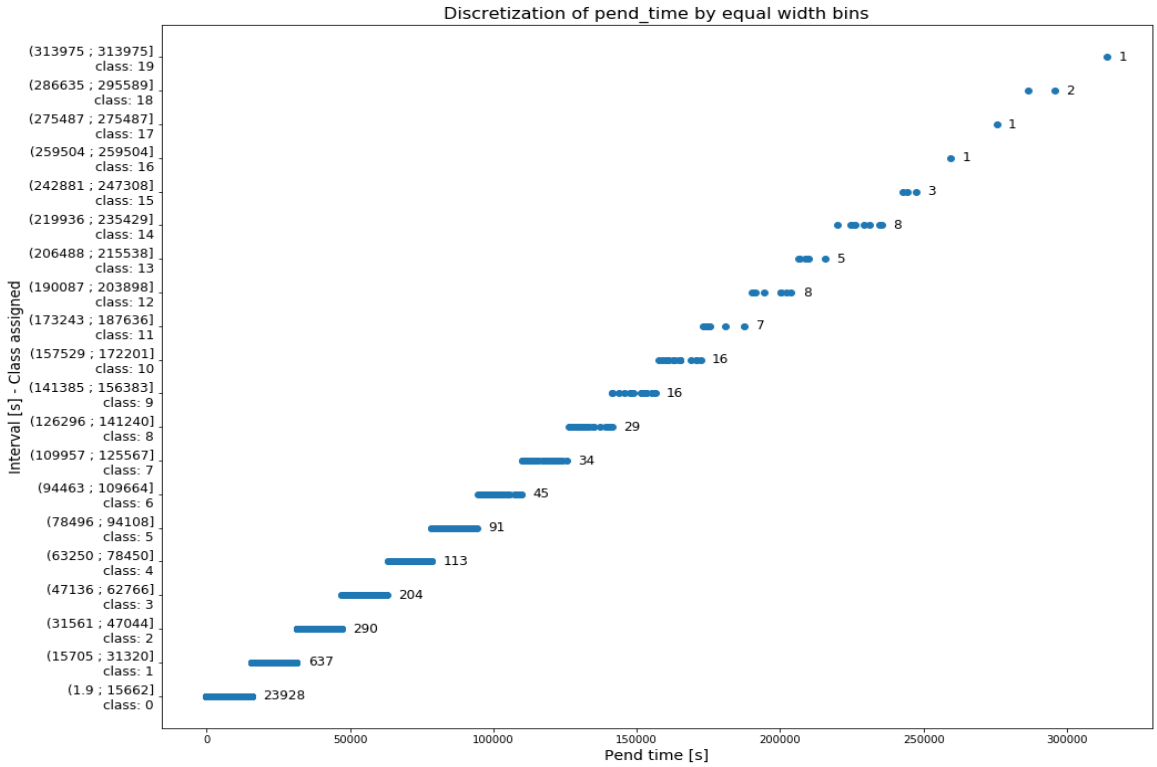



**Figure 3.17.** Distribution of the records in the 20 classes created with the discretization using intervals with same width.

In this diagram are reported, in fact, both the classes, with the minimum and maximum value of the interval, and the numerosity of each of them.
The table 3.9 shows the results obtained from the evaluation of the algorithms after the discretization process of the target variable in 20 ranges of same width.

**Table 3.9.** Results of the algorithms evaluation using a discretization with intervals of same width

| Algorithm | Accuracy | F-measure |
|---|---|---|
| Logistic Regression | 0.943789 | 0.086996 |
| Perceptron | 0.752752 | 0.070170 |
| SVM Classifier | 0.951650 | 0.142131 |
| KNN Classifier | 0.951651 | 0.266283 |
| Decision Tree | 0.946934 | 0.130014 |
| Random Forest | 0.946541 | 0.189119 |
| MLP Classifier | 0.948506 | 0.166967 |

Despite the excellent results, regarding the accuracy, obtained during the evaluation of the models, these do not respect the reality of the facts, as demonstrated by the low value of the F-measure. In fact, these high accuracy values are linked to the fact that, using this kind of discretization, the classification problem would not be balanced since class 0 contains the 94% of dataset records.
Therefore, it has been chosen the binning with intervals of same frequency, to have classes containing more or less the same number of records (as it is possible to see in figure 4.7). Using this other type of discretization and performing a further evaluation of the algorithms, the results in the table 3.10 have been obtained.

**Table 3.10.** Results of the algorithms evaluation using a discretization with intervals of same frequency

| Algorithm | Accuracy | F-measure |
|---|---|---|
| Logistic Regression | 0.115959 | 0.045421 |
| Perceptron | 0.042060 | 0.013653 |
| SVM Classifier | 0.176887 | 0.130552 |
| KNN Classifier | 0.218946 | 0.196251 |
| Decision Tree | 0.240959 | 0.208368 |
| Random Forest | 0.243711 | 0.223526 |
| MLP Classifier | 0.243711 | 0.206632 |

Based on the results of the latter assessment, it was finally decided to make a further selection on the classification algorithms to be used in the experiments reported in section 4.2, taking into account only the SVM, the KNN, the Decision Tree, the Random Forest and the MLP, substantially discarding all the linear models previously considered.

# 4. Experimental Results

All the results of the experiments carried out to find an acceptable solution for the prediction of pending time are reported in this chapter. In the first section, the data obtained by the regression techniques and the reason for which classification ones are then analysed in the second section are discussed. As regards classification, it was decided to subdivide the paragraph into three sub-paragraphs: hierarchical, multi-class and binary classification, to group the experiments and increase their comprehensibility.

## 4.1 Regression Experiments

As the pending time is a continuous numerical variable, the first approach related to the prediction process involved the use of regression techniques to try to identify a finite value able to get as close as possible to the final result, resulting in the error and the most accurate possible.

Initially, linear regression algorithms were evaluated to verify if there were assumptions for creating a linear prediction model. Therefore, the input data was randomly divided into two groups, one for the training and the other one for the testing, which were then used to perform a prediction test. From these, the results obtained are expressed in the table 4.1.

**Table 4.1.** Evaluation of some linear regression algorithms

| Algorithm | Accuracy |
|---|---|
| Linear Ridge Regressor | 0.191842 |
| Lasso | 0.191828 |
| Elastic Net | 0.188593 |
| Bayesian Ridge Regressor | 0.191310 |
| Polynomial Regressor | 0.316552 |

Later, non-linear regression algorithms were also selected to verify their performance with the available data. Considering the same training and testing data used with linear models, the results obtained from the evaluations carried out are reported in the table 4.2.

**Table 4.2.** Evaluation of some non-linear regression algorithms

| Algorithm | Accuracy |
|---|---|
| Support Vector Regressor | -0.059722 |
| KNN Regressor | 0.378152 |
| Decision Tree 1 | 0.270867 |
| Decision Tree 2 | 0.267371 |
| Decision Tree 3 | 0.238567 |
| MLP Regressor | 0.322663 |

The three different Decision Tree model differ only for the max-depth parameter which respectively, is set to 3,5 and 7 maximum depth levels.

Considering the results obtained from the evaluation of both the linear and non-linear regression algorithms, only those with a sufficiently high accuracy were selected, although the values obtained were still not acceptable. For the linear models, the only one selected was the polynomial one, while for non-linear models the only one to be discarded was the SVR, which even presented a negative accuracy.
To try to improve the performance of these algorithms and, therefore, obtain better results, a k-fold cross validation with $k = 10$ was applied. From this attempt, the results obtained are reported in the table 4.3.

**Table 4.3.** Results of the k-fold CV technique, with $k = 10$, using all the variables available

| Algorithm | Max Accuracy | Mean Accuracy |
|---|---|---|
| Polynomial Regressor | 0.533203 | 0.231085 |
| KNN Regressor | 0.400557 | 0.172242 |
| Decision Tree 1 | 0.453951 | 0.273126 |
| Decision Tree 2 | 0.495346 | 0.235226 |
| Decision Tree 3 | 0.467555 | 0.002513 |
| MLP Regressor | 0.550093 | 0.328904 |

These data are the result, as explained at the end of the preamble of the section 2.3, of ten separate tests using ten different train and test sets of which it was wanted to highlight the maximum and the mean value found.
From the final results, it can be noted that the Decision Tree with $max\_depth = 7$ is characterized by an extremely low mean value of accuracy. This means that, compared to the others, this algorithm does not work at all well with the data available and, therefore it was decided not to use it any more in experiments performed later.
In addition, regarding the correlation matrix related to context data, shown in figure 3.13, it was noted that the cores_available and nodes_available attributes are highly correlated and have a common variable in their calculation process. For this reason, since the first attribute provides more general and complete information, it was decided to remove the second from the group of input variables.
Following these changes, it was decided to repeat the operations of k-fold cross validation, again with $k = 10$, obtaining the results expressed in the table 4.4.

**Table 4.4.** Results of the k-fold CV technique, with $k = 10$, after having removed the nodes_available attribute

| Algorithm | Max Accuracy | Mean Accuracy |
|---|---|---|
| Polynomial Regressor | 0.532658 | 0.235069 |
| KNN Regressor | 0.400554 | 0.172238 |
| Decision Tree 1 | 0.453951 | 0.270740 |
| Decision Tree 2 | 0.493425 | 0.177651 |
| MLP Regressor | 0.532969 | 0.325323 |

As it can be noted, comparing with the data shown in the table 4.3, most of the values remain

more or less unchanged, a sign that this variable does not provide any useful information to improve the performance of the models.
To try to improve the results obtained, we tried to perform the k-fold cross validation again, but, in this case, with $k = 20$ to try to improve the performance of prediction models. From this attempt the values obtained are reported in the table 4.5.

**Table 4.5.** Results of the k-fold CV technique, with $k = 20$, after having removed the nodes_available attribute

| Algorithm | Max Accuracy | Mean Accuracy |
|---|---|---|
| Polynomial Regressor | 0.551733 | 0.006580 |
| KNN Regressor | 0.581727 | 0.080216 |
| Decision Tree 1 | 0.533837 | 0.200991 |
| Decision Tree 2 | 0.571285 | 0.173368 |
| MLP Regressor | 0.630281 | 0.220024 |

As it is possible to see, this technique has improved, even if slightly, the performance of models on specific data groups, but, for the most of them, these have worsened given the drastic decrease in average values.

Finally, as a last shot to try to improve the accuracy of prediction of the selected models, given the high number of records available, it was decided to apply the holdout technique, also analysed in the final part of the preamble of section 2.3. The results obtained from this technique can be viewed in the table 4.6.

**Table 4.6.** Results of the Holdout technique, using all the variable except the nodes_available

| Algorithm | Accuracy |
|---|---|
| Polynomial Regressor | 0.217887 |
| KNN Regressor | 0.119690 |
| Decision Tree 1 | 0.199433 |
| Decision Tree 2 | 0.149165 |
| MLP Regressor | 0.233375 |

Unfortunately, even this partitioning technique failed to improve the performance of the models, as can be seen from the comparison with the results obtained with the k-fold cross validation with $k = 10$ shown in the table 4.4.

## 4.2 Classification Experiments

Since the accuracy values obtained from linear and non-linear regression models were not at all acceptable, it has been decided to use classification techniques to predict the time interval of the real pending time when a new job is submitted. In this way, we try to define the time interval with the greatest accuracy possible, thus allowing, in the future, the execution of other prediction techniques to finally arrive at predicting the exact value of pending time.

Classification experiments were initially carried out with more than two prediction ranges, the results of which are given in section 4.2.2. Doing so, we studied the behaviour of the selected algorithms during the variation of the number of classes to be predicted and also the way these classes are generated. Subsequently, we moved on to binary classification experiments, reported in the section 4.2.3, to be able to identify time periods so that the algorithms were able to predict the class of belonging in the best way possible. Finally, on the basis of the knowledge acquired through these tests, a hierarchical approach has been adopted, meaning that several prediction levels are used, each of which depends on the results obtained from the classifications carried out in the previous levels. These latest experiments are reported in section 4.2.1 where also the results of the best approach analysed in section 3.5.1 are presented.

### 4.2.1 Hierarchical Classification Experiments

This kind of experiment was used as a last approach to try to implement a classification system as accurate as possible. Indeed, the information obtained from the various tests carried out, the result of which are reported in section 4.2.2 and 4.2.3, has been used to design useful experiments and implement their various prediction levels. Finally, with the foreground, the final system, presented in section 3.5.1, was defined with which the results discussed in section 4.2.1.4 were obtained.

#### 4.2.1.1 Usage of two hierarchy levels with four final classes

Starting from the binary classification with 900 seconds threshold, the results of which are presented in section 4.2.3.1, it was decided to use the MLP as the algorithm for the first level since, in addition to being one of the models with the best results in that test, it also has the ability to learn any type of function in real time. This is extremely useful since the final purpose of this project is precisely to create a prediction model that must give immediate answers, being within a chain of submission.
The second level, as can be seen from the diagram in figure 4.1, is characterized by two further binary classifications. In this case the thresholds imposed in the second level were designed to give realistic feedback to users. Considering that the second level uses for the training only those records predicted correctly in the previous level, in this case, looking at the MLP confusion matrix shown in figure 4.23, 22762 records were used. Of these 18597 were used to predict the classes 0.0 and 0.1, while 4165 for the other two.

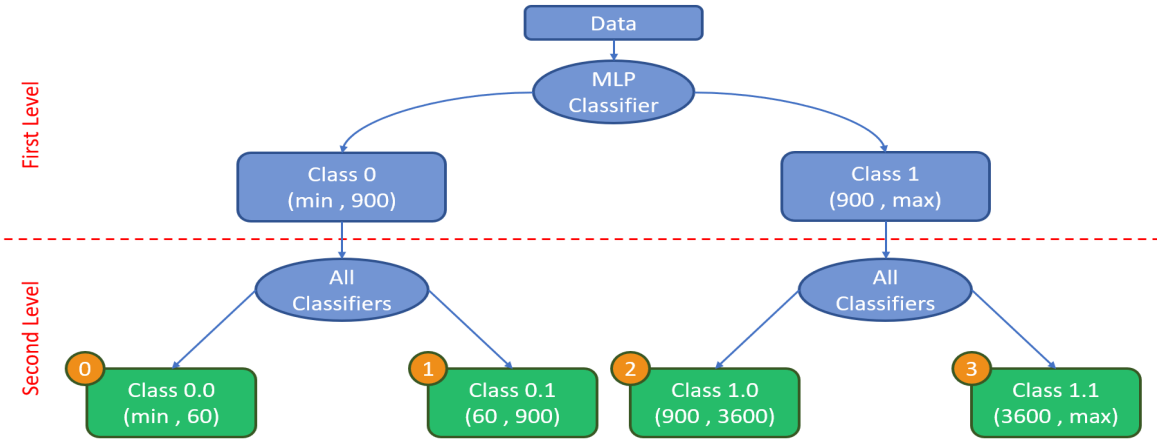



**Figure 4.1.** Schema of the classification strategy with 2 levels and 4 final classes.

To evaluate the performance of models in the prediction of classes 0.0 and 0.1, a stratified

20-fold cross validation has been performed and the accuracy, precision, recall and F-measure indices have been computed, the values of which are reported in the table 4.7.

**Table 4.7.** Results for the second level classification of the class 0 related to the esperiment with 2 levels and 4 final classes

| Algorithm | Mean Accuracy | Precision Class 0.0 | Precision Class 0.1 | Recall Class 0.0 | Recall Class 0.1 | F-measure Class 0.0 | F-measure Class 0.1 |
|---|---|---|---|---|---|---|---|
| SVM Classifier | 0.937893 | 0.941022 | 0.748344 | 0.995605 | 0.173180 | 0.967544 | 0.281269 |
| KNN Classifier | 0.941550 | 0.951167 | 0.670846 | 0.987856 | 0.327969 | 0.969164 | 0.440555 |
| Decision Tree | 0.942948 | 0.949737 | 0.721014 | 0.991094 | 0.304981 | 0.969975 | 0.428648 |
| Random Forest | 0.940259 | 0.948451 | 0.674460 | 0.989533 | 0.287356 | 0.968557 | 0.403009 |
| MLP Classifier | 0.944131 | 0.948309 | 0.782979 | 0.994101 | 0.281992 | 0.970665 | 0.414648 |

To calculate the performance of the models in the prediction of classes 1.0 and 1.1, instead, a stratified 50-fold cross validation was performed, since the number of records available was quite small, and the accuracy, precision, recall and F-measure indices were computed, the values of which are given in table 4.8.

**Table 4.8.** Results for the second level classification of the class 1 related to the esperiment with 2 levels and 4 final classes

| Algorithm | Mean Accuracy | Precision Class 0.0 | Precision Class 0.1 | Recall Class 0.0 | Recall Class 0.1 | F-measure Class 0.0 | F-measure Class 0.1 |
|---|---|---|---|---|---|---|---|
| SVM Classifier | 0.749698 | 0.650624 | 0.765538 | 0.302653 | 0.933672 | 0.412197 | 0.841287 |
| KNN Classifier | 0.712696 | 0.495717 | 0.768802 | 0.382645 | 0.840609 | 0.431903 | 0.803104 |
| Decision Tree | 0.738573 | 0.539700 | 0.781318 | 0.415702 | 0.854822 | 0.469654 | 0.816419 |
| Random Forest | 0.727449 | 0.527829 | 0.792355 | 0.470248 | 0.827750 | 0.497378 | 0.809666 |
| MLP Classifier | 0.761790 | 0.625798 | 0.787108 | 0.404959 | 0.900846 | 0.491721 | 0.840145 |

#### 4.2.1.2 Usage of two hierarchy levels with five final classes

To try to improve somewhat the classification performance of the system it was decided to slightly change the test schema discussed in section 4.2.1.1, transforming the binary classification implemented for class 1 into a multi-class classification with 3 distinct intervals. Since the method applied to the first level has not changed, the number of records used for the training of the various classes is the same. The general schema of this new test is shown in figure 4.2.

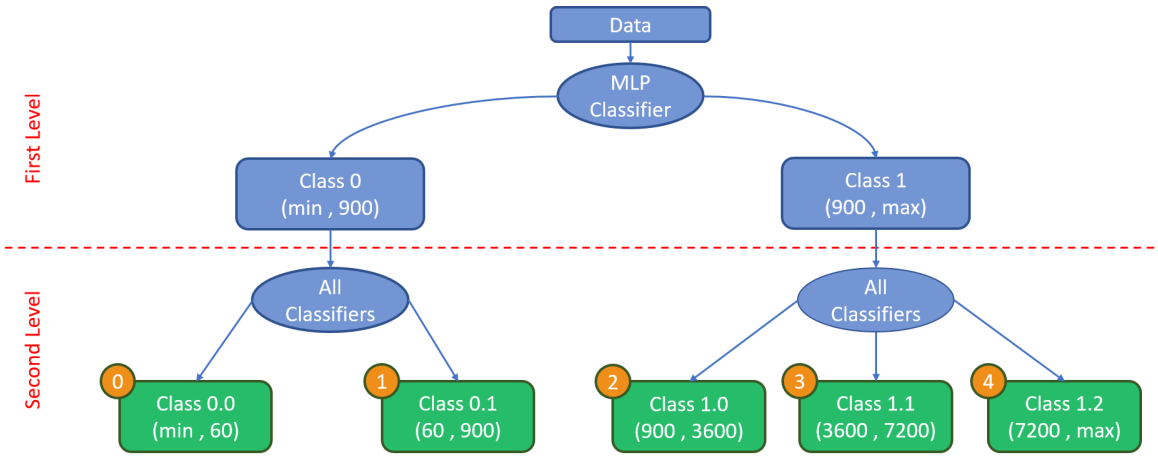



**Figure 4.2.** Schema of the classification strategy with 2 levels and 5 final classes.

Considering the fact that the second level binary classification applied to class 0 has not changed compared to the experiment analysed in the section 4.2.1.1, the results of the accuracy, precision, recall and F-measure indices are those shown in the table 4.7. Instead, with regard to the multi-class classification applied to class 1, to analyse the performance of the selected algorithms, a stratified 50-fold cross validation has been performed to then calculate the accuracy, precision, recall and F-measure indices, the mean values of which are given in the table 4.9.

**Table 4.9.** Results for the second level classification of the class 1 related to the esperiment with 2 levels and 5 final classes

| Algorithm | Mean Accuracy | Mean Precision | Mean Recall | Mean F-measure |
|---|---|---|---|---|
| SVM Classifier | 0.612092 | 0.396542 | 0.483656 | 0.434300 |
| KNN Classifier | 0.551632 | 0.484539 | 0.455373 | 0.455720 |
| Decision Tree | 0.593954 | 0.520826 | 0.514457 | 0.506964 |
| Random Forest | 0.558888 | 0.498518 | 0.495766 | 0.496920 |
| MLP Classifier | 0.632406 | 0.575030 | 0.531690 | 0.522985 |

From these results, however, it can be noted that performance tends to slightly decrease compared to the second level classification applied to class 1 performed in section 4.2.1.1. This is quite normal since there is the possibility to predict an extra class. The problem is that the overall performance of the system is still low to be accepted.

#### 4.2.1.3 Usage of three hierarchy levels with six final classes

To try to better characterize prediction intervals so that users can receive more truthful feedback, we have considered the diagram shown in figure 4.1 and a third level of classification, related to class 1.1, has been added. For the execution of this new prediction 2662 records were used (those correctly classified as belonging to class 1.1 in the second level). This process allowed us to obtain the diagram of the prediction system shown in figure 4.3.

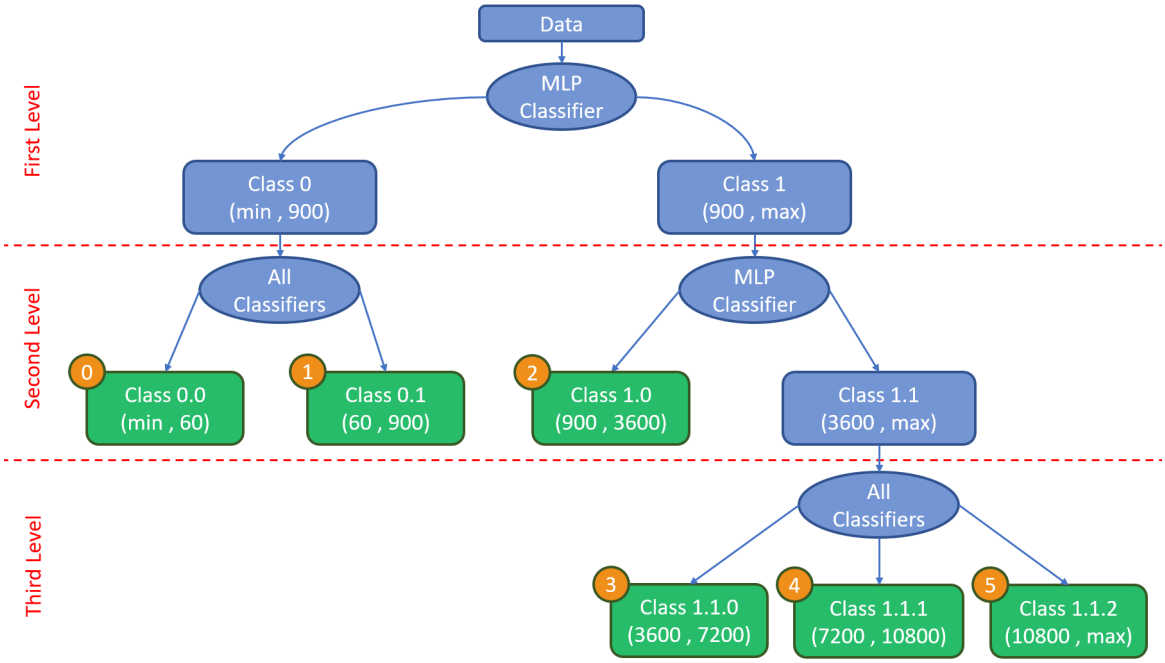



**Figure 4.3.** Schema of the classification strategy with 3 levels and 6 final classes.

Since this experiment is based on the schema analysed in section 4.2.1.1, the index results for classes 0.0 and 0.1 are those given in the table 4.7. For class 1.0, however, the results are already present in the table 4.8.
To analyse the performance of the classification algorithms used in the third level to subdivide the class 1.1, given the low availability of training records, a stratified 30-fold cross validation was performed and the accuracy, precision, recall and F-measure indices were computed, the mean values of which are reported in the table 4.10.

**Table 4.10.** Results for the third level classification of the class 1.1 related to the esperiment with 3 levels and 6 final classes

| Algorithm | Mean Accuracy | Mean Precision | Mean Recall | Mean F-measure |
|---|---|---|---|---|
| SVM Classifier | 0.617581 | 0.439337 | 0.370283 | 0.323296 |
| KNN Classifier | 0.596121 | 0.398759 | 0.395409 | 0.386362 |
| Decision Tree | 0.608189 | 0.473791 | 0.437790 | 0.435682 |
| Random Forest | 0.587153 | 0.464728 | 0.449098 | 0.452004 |
| MLP Classifier | 0.613073 | 0.497023 | 0.440774 | 0.438479 |

Based on the results obtained in the previous table, it can be noted that this type of structure for the hierarchical classification is little usable since the values obtained for some classes are too low significant. It is for this reason that a completely new schema has been devised which considers all the knowledge acquired. This is analysed in section 3.5.1 and its results will be reported in section 4.2.1.4.

#### 4.2.1.4 Results of the best solution proposed

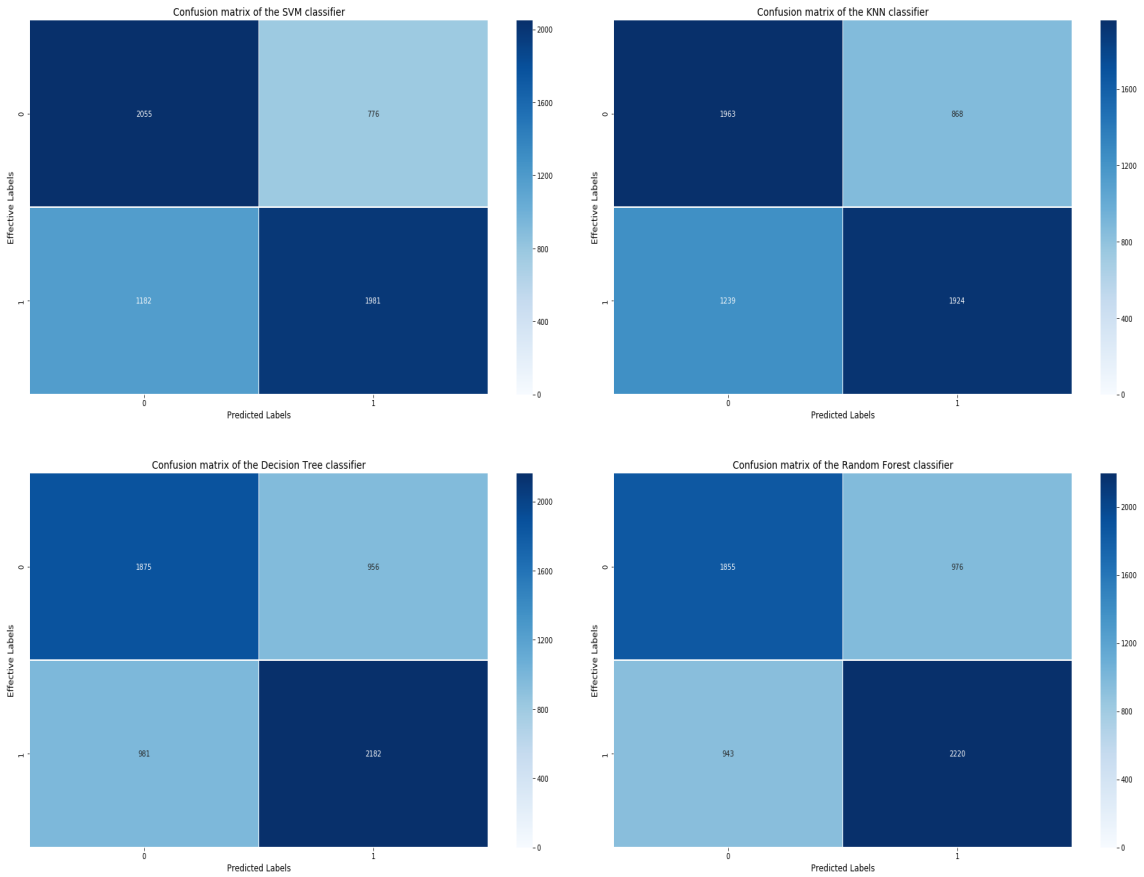
Since the first level of this experiment, the diagram of which is shown in figure 3.16, coincides with the binary classification experiment with 60 seconds threshold analysed in section 4.2.3.6,

the results of the first classification level are those given in the table 4.28.
For the prediction of the second level it was possible to use 5994 records that, as it can be noted in the MLP confusion matrix reported in figure 4.29, is the number of correctly predicted records belonging to class 1. Also in the second level a binary classification has been applied using a threshold of 3600 seconds. After discretizing the target variable a second time using the intervals generated with this threshold, a stratified 20-fold cross validation was performed to assess the performance of the second level algorithms and the accuracy, precision, recall and F-measure indices were computed, the values of which are shown in the table 4.11.

**Table 4.11.** Results for the second level classification of class 1 related to the best classification approach

| Algorithm | Mean Accuracy | Precision Class 1.0 | Precision Class 1.1 | Recall Class 1.0 | Recall Class 1.1 | F-measure Class 1.0 | F-measure Class 1.1 |
|---|---|---|---|---|---|---|---|
| SVM Classifier | 0.673340 | 0.634847 | 0.718535 | 0.725892 | 0.626304 | 0.677324 | 0.669257 |
| KNN Classifier | 0.648482 | 0.613054 | 0.689112 | 0.693395 | 0.608283 | 0.650754 | 0.646180 |
| Decision Tree | 0.676510 | 0.656513 | 0.695347 | 0.662310 | 0.689851 | 0.659398 | 0.692588 |
| Random Forest | 0.679847 | 0.662974 | 0.694618 | 0.655245 | 0.701865 | 0.659087 | 0.698223 |
| MLP Classifier | 0.707040 | 0.671890 | 0.745378 | 0.742141 | 0.675624 | 0.705270 | 0.708789 |

These results can also be obtained manually from the generated confusion matrices shown in figure 4.4.



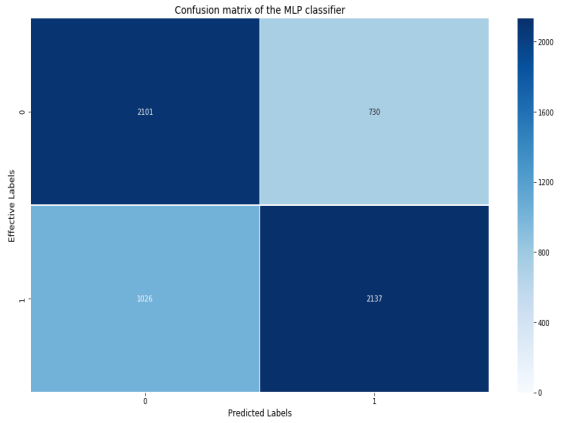


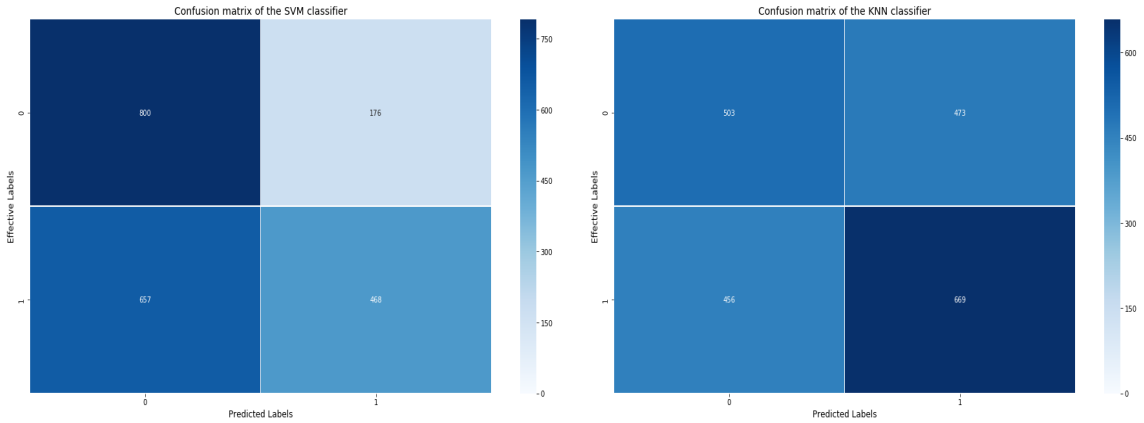

**Figure 4.4.** Confusion matrices generated for each model during the evaluation of the algorithms for the selection of the one to use in the second level of the best solution.

Considering the results of the MLP, 4238 records have been used for the training of the third level that are equivalent to those correctly predicted both as class 1.0 and as 1.1.
For the third classification applied to class 1.0, it was decided to perform a binary classification using a 900 seconds threshold, thus creating the two final classes 1.0.0 and 1.0.1. After the discretization of the target variable with the generated ranges, to evaluate prediction models for these two classes a stratified 30-fold cross validation was performed and, through the obtained results, the accuracy, precision, recall and F-measure indices have been computed, the values of which are given in the table 4.12.

**Table 4.12.** Results for the third level classification of class 1.0 related to the best classification approach

| Algorithm | Mean Accuracy | Precision Class 1.0.0 | Precision Class 1.0.1 | Recall Class 1.0.0 | Recall Class 1.0.1 | F-measure Class 1.0.0 | F-measure Class 1.0.1 |
|---|---|---|---|---|---|---|---|
| SVM Classifier | 0.603522 | 0.549073 | 0.726708 | 0.819672 | 0.416000 | 0.657624 | 0.529112 |
| KNN Classifier | 0.557830 | 0.524505 | 0.585814 | 0.515369 | 0.594667 | 0.519897 | 0.590207 |
| Decision Tree | 0.642075 | 0.598246 | 0.694069 | 0.698770 | 0.592889 | 0.644612 | 0.639501 |
| Random Forest | 0.582580 | 0.553863 | 0.604907 | 0.521516 | 0.635556 | 0.537203 | 0.619853 |
| MLP Classifier | 0.587815 | 0.545230 | 0.646328 | 0.679303 | 0.508444 | 0.604927 | 0.569154 |

In figure 4.5 it is possible to view the generated confusion matrices, related to this case, from which the results in the table 4.12 can be derived.



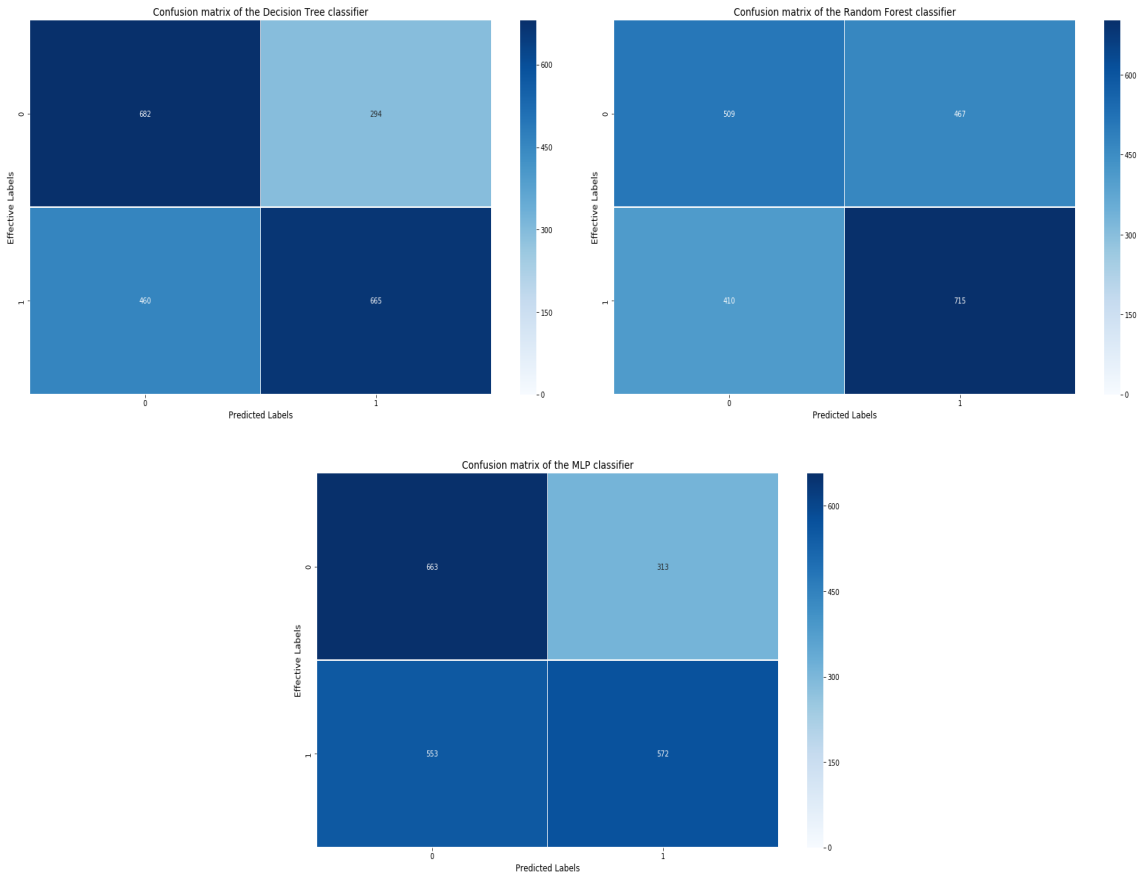



**Figure 4.5.** Confusion matrices generated for each model during the evaluation of the algorithms for the selection of the one to use in the third level for the class 1.0 of the best solution.

Finally, for the third level classification applied to class 1.1 it was decided to perform a 3-class classification to define the ranges $(3600, 7200]$, $(7200, 10800]$ and $(10800, \max]$, thus creating classes 1.1.0, 1.1.1 and 1.1.2 respectively. To evaluate the performance of the algorithms in the prediction of these 3 classes, after having discretized the target variable based on these generated ranges, a stratified 30-fold cross validation was performed and, subsequently, the accuracy, precision, recall and F-measure indices were computed, the mean values of which are reported in the table 4.13.

**Table 4.13.** Results for the third level classification of class 1.1 related to the best classification approach

| Algorithm | Mean Accuracy | Mean Precision | Mean Recall | Mean F-measure |
|---|---|---|---|---|
| SVM Classifier | 0.671034 | 0.518335 | 0.382680 | 0.352399 |
| KNN Classifier | 0.608797 | 0.402744 | 0.380130 | 0.373967 |
| Decision Tree | 0.666355 | 0.537390 | 0.459776 | 0.471808 |
| Random Forest | 0.632195 | 0.490659 | 0.459686 | 0.468268 |
| MLP Classifier | 0.662611 | 0.531875 | 0.445390 | 0.455680 |

Also in this case has been generated the confusion matrices, one for each model, relative to this specific step that are reported in figure 4.6.

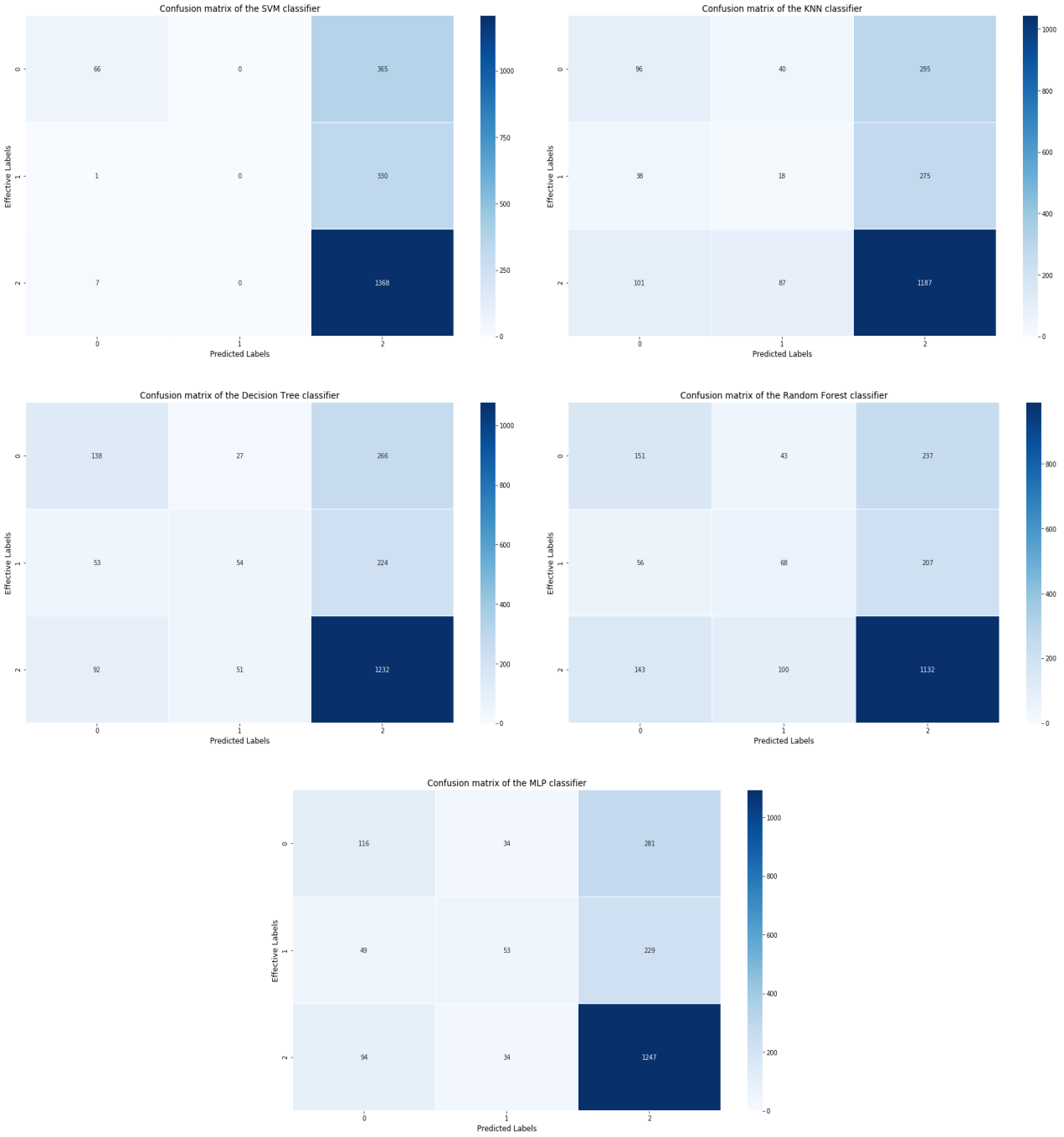



**Figure 4.6.** Confusion matrices generated for each model during the evaluation of the algorithms for the selection of the one to use in the third level for the class 1.1 of the best solution.

As can be seen from these matrices, the poor results obtained by the last classification are due to the low number of correct predictions regarding the first two classes. In addition, it can be noted that these two have a significantly lower number of associated records than the ones associated with the third class, leading to limitations in model training.

### 4.2.2 Multi-class Classification Experiments

The multi-class experiments were the first to be carried out in such a way to define a reasonable number of classes both in terms of meaning and of correctness of classification. The first tests consider the subdivision of the target variable into several equal frequency ranges, since those

of the same width have been discarded (as explained in section 3.5). In addition, in the last four experiments described in this section, the ranges have been defined in a way to provide acceptable feedback to the user, also considering the information obtained from the previous tests.

#### 4.2.2.1 Classification experiments discretizing the target variable in 20 classes of same frequencies

The first classification experiment was carried out by trying to discretize the values of the pending time in 20 different classes, the distribution of which is shown in figure 4.7.

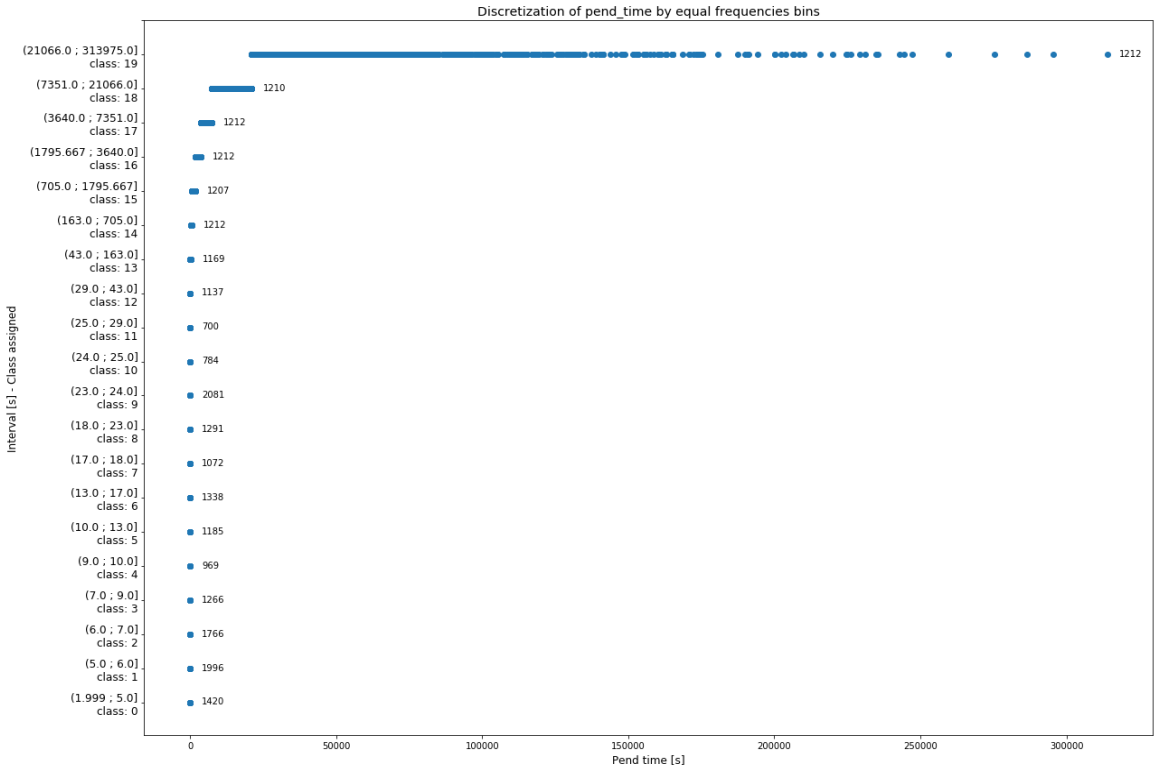



**Figure 4.7.** Visualization of the 20 classes generated and the distribution of the records inside them.

Using these intervals, a stratified 20-fold cross validation was performed so that each group had more or less the same frequency of records belonging to the various classes. With the results of this process, to evaluate the performance of the classifiers selected, the accuracy, precision, recall and F-measure indices have been computed and their mean values are reported in the table 4.14.

**Table 4.14.** Results of the experiment with 20 classes with equal frequencies

| Algorithm | Mean Accuracy | Mean Precision | Mean Recall | Mean F-measure |
|---|---|---|---|---|
| SVM Classifier | 0.170879 | 0.124497 | 0.159318 | 0.120080 |
| KNN Classifier | 0.166713 | 0.157180 | 0.158591 | 0.143910 |
| Decision Tree | 0.213963 | 0.195345 | 0.197367 | 0.177636 |
| Random Forest | 0.196391 | 0.181803 | 0.182780 | 0.170767 |
| MLP Classifier | 0.215417 | 0.202553 | 0.199668 | 0.179786 |

In addition to these values, to better visualize how the records were classified during the training process, the confusion matrices were also produced for each individual model and they are shown in figure 4.8.

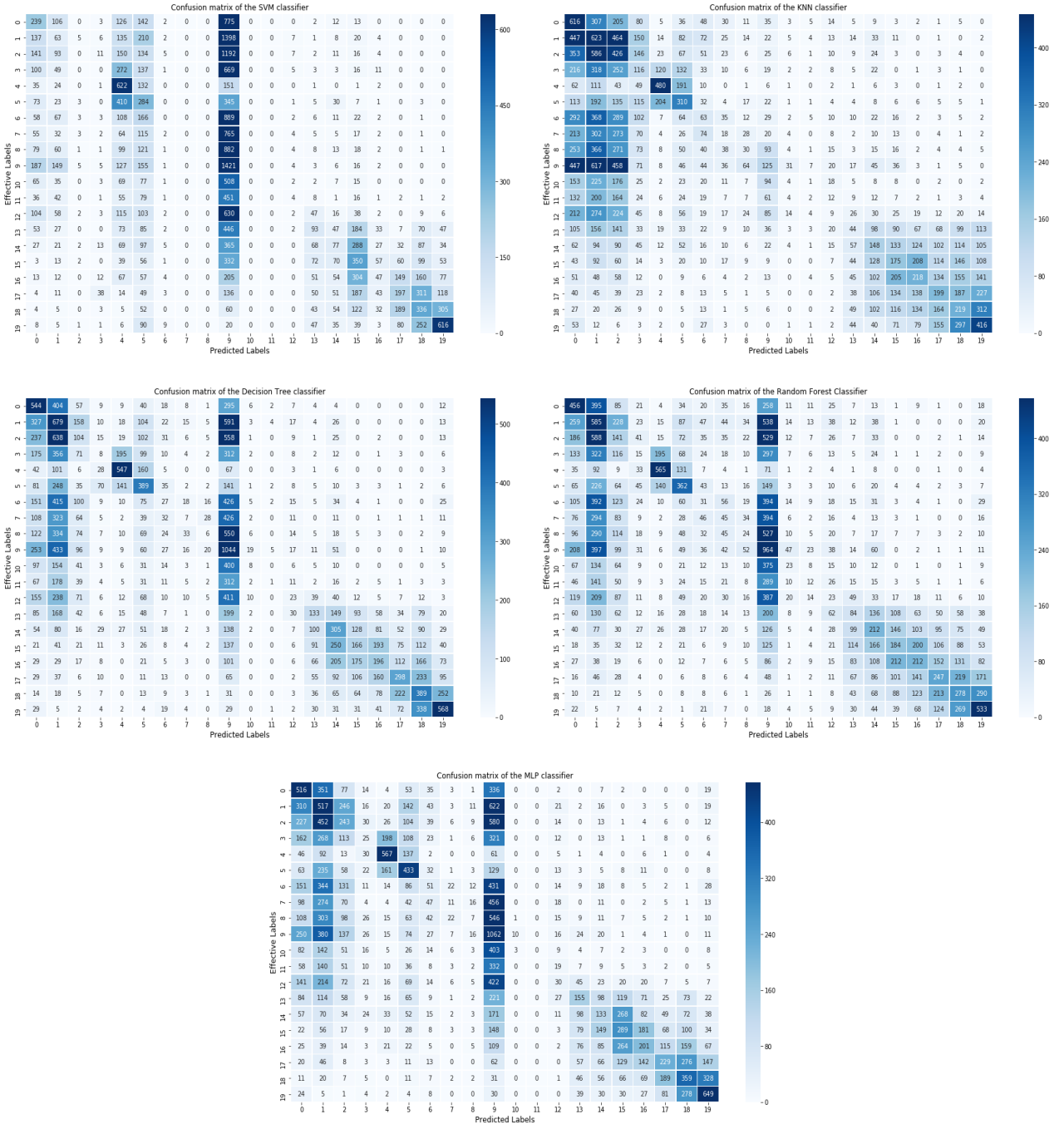



**Figure 4.8.** Confusion matrices generated for each model during the experiment with 20 classes with same frequency.

These matrices confirm the results obtained. As it is possible to see, all classifiers have problems on the prediction of the classes. In fact, a lot of mistakes are made as demonstrated by the fact that the cells of the various diagonals do not contain values close to the numerosity of the specific class. In addition, from these matrices it is clear that models tend to go wrong more or less in the same areas and that is why it was decided to do other experiments decreasing the number of predictable classes.

#### 4.2.2.2 Classification experiment discretizing the target variable in 10 classes of same frequencies

Subsequently, it was decided to try a training process in which the target variable was divided into 10 different classes with same frequency. The intervals created and the respective absolute frequencies are shown in figure 4.9.

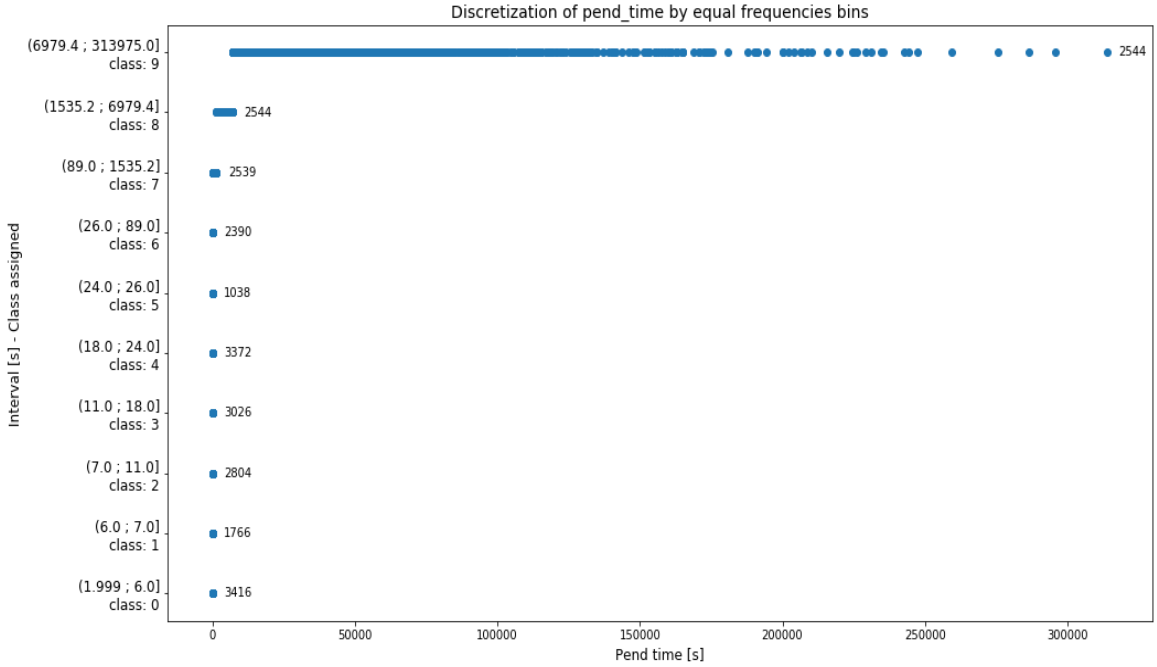



**Figure 4.9.** Visualization of the 10 classes generated and the distribution of the records inside them.

As in the previous case, a stratified 20-fold cross validation was performed and the accuracy, precision, recall and F-measure indices were computed, the mean values of which are given in the table 4.15.

**Table 4.15.** Results of the experiment with 10 classes with equal frequencies

| Algorithm | Mean Accuracy | Mean Precision | Mean Recall | Mean F-measure |
|---|---|---|---|---|
| SVM Classifier | 0.307088 | 0.272931 | 0.271401 | 0.250410 |
| KNN Classifier | 0.275600 | 0.260910 | 0.250677 | 0.230243 |
| Decision Tree | 0.326821 | 0.310865 | 0.287534 | 0.272162 |
| Random Forest | 0.312119 | 0.289264 | 0.276104 | 0.265845 |
| MLP Classifier | 0.341090 | 0.326092 | 0.300946 | 0.281455 |

As it is possible to see, the results have improved somewhat compared to the case analysed previously in section 4.2.2.1. To show how the records have been classified, also in this case have been generated the confusion matrices reported in figure 4.10.

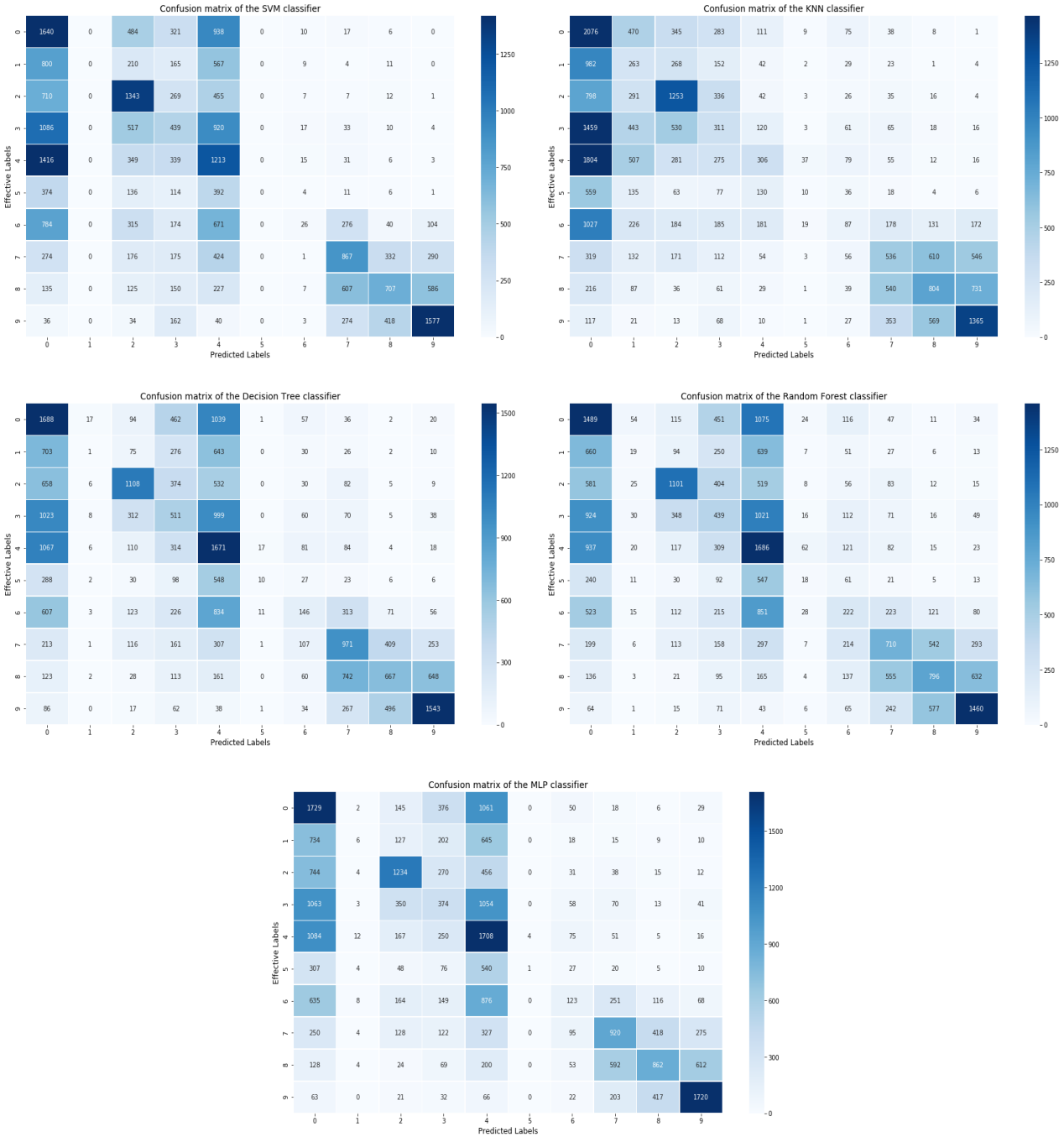



**Figure 4.10.** Confusion matrices generated for each model during the experiment with 10 classes with same frequency.

From these matrices it can be noted that, with these changes, the algorithms tend to make errors less than in the previous case but not enough to make accurate prediction levels. In fact, even here can be seen, regardless of the model considered, the areas of classification denser than the others. A clear example is the prediction of the last three classes, in which models struggle to correctly assign records.

#### 4.2.2.3 Classification experiment discretizing the target variable in 7 classes of same frequencies

Since results tend to improve by decreasing the number of predictable classes, it has been tried to see how models behave in predicting 7 distinct classes with more or less the same frequency. The generated intervals and their numerosity are represented in figure 4.11.

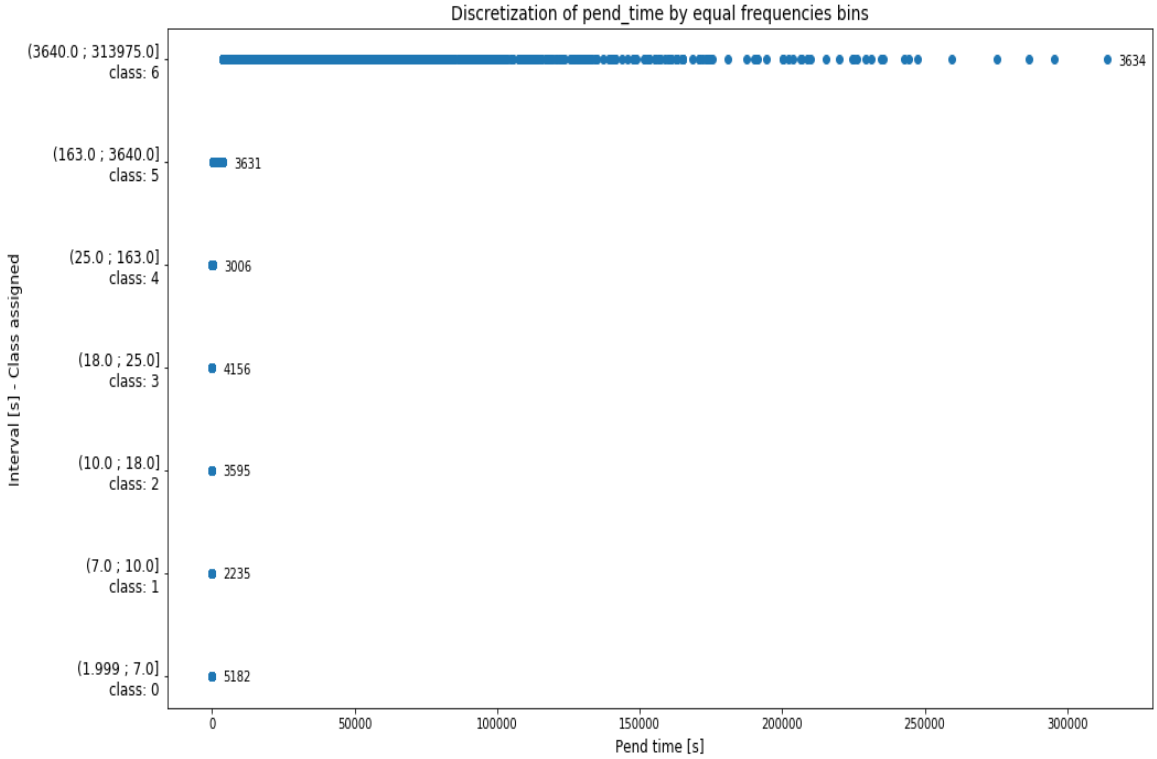



**Figure 4.11.** Visualization of the 7 classes generated and the distribution of the records inside them.

As above, a stratified 20-fold cross validation was performed and the accuracy, precision, recall and F-measure indices were computed, the mean values of which are reported in the table 4.16 to assess the performance of the various classifiers selected.

**Table 4.16.** Results of the experiment with 7 classes with equal frequencies

| Algorithm | Mean Accuracy | Mean Precision | Mean Recall | Mean F-measure |
|---|---|---|---|---|
| SVM Classifier | 0.382602 | 0.378283 | 0.360221 | 0.335620 |
| KNN Classifier | 0.368882 | 0.375971 | 0.336808 | 0.318225 |
| Decision tree | 0.405244 | 0.409913 | 0.374159 | 0.365250 |
| Random Forest | 0.399269 | 0.403519 | 0.373465 | 0.370206 |
| MLP Classifier | 0.418452 | 0.435021 | 0.390245 | 0.385226 |

It can be noted that, also in this case, the performance of the models has improved compared to the previous cases but not enough. Below, in figure 4.12, are shown the confusion matrices generated to try to understand where the classification errors were made, considering the poor results obtained, and how to fix them.

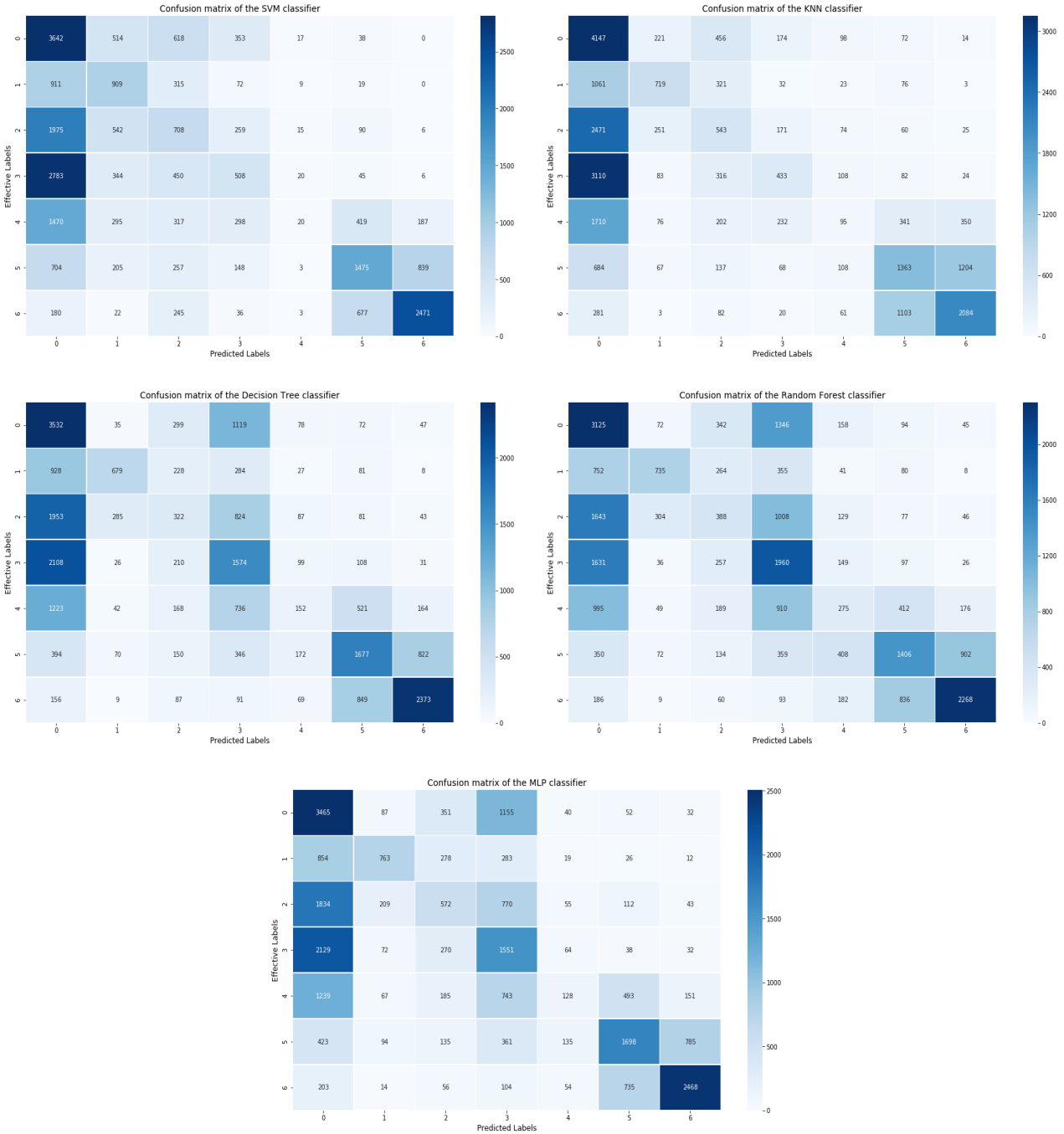



**Figure 4.12.** Confusion matrices generated for each model during the experiment with 7 classes with same frequency.

In all the generated matrices it can be noted that the selected models have problems distinguishing the first five classes, since all five are mostly predicted as class 0. Beyond that, it is possible also to highlight another difficulty of prediction between the last two labels.

#### 4.2.2.4 Classification experiment discretizing the target variable in 4 classes of same frequencies

Finally, it was decided to make a final try with 4 classes of the same frequency and, as in the previous cases, a useful graph, shown in figure 4.13, to display the distribution of the records was generated.

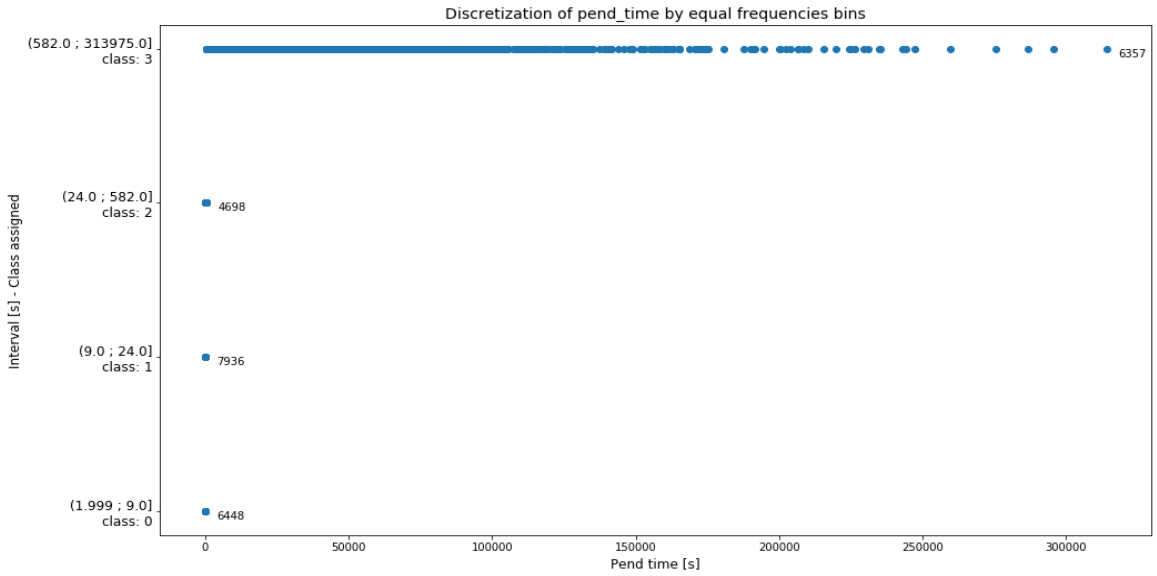



**Figure 4.13.** Visualization of the 4 classes generated and the distribution of the records inside them.

Immediately after performing a stratified 20-fold cross validation, the accuracy, precision, recall and F-measure indices were computed for all the chosen algorithms, to evaluate their performance. The mean values of these indices are given in the table 4.17.

**Table 4.17.** Results of the experiment with 4 classes with equal frequencies

| Algorithm | Mean Accuracy | Mean Precision | Mean Recall | Mean F-measure |
|---|---|---|---|---|
| SVM Classifier | 0.508235 | 0.537935 | 0.459793 | 0.426678 |
| KNN Classifier | 0.476041 | 0.467252 | 0.464447 | 0.419915 |
| Decision Tree | 0.514525 | 0.487938 | 0.480335 | 0.461583 |
| Random Forest | 0.497976 | 0.485055 | 0.469455 | 0.465651 |
| MLP Classifier | 0.524863 | 0.514528 | 0.490659 | 0.470602 |

The results of this test, as it is possible to see from the data previously reported, improve a little compared to those of the case analysed in the section 4.2.2.3, but, despite there has been a significant decrease in the number of the predictable classes (the number of these, in fact, has been almost halved), these values, in most cases, do not exceed 50% of correctness in the classification execution of the records. To try understanding the reasons for what happened, it is necessary to see the confusion matrices generated and reported in figure 4.14.


**Figure 4.14.** Confusion matrices generated for each model during the experiment with 4 classes with same frequency.

From these matrices it can be found more or less the same problem evidenced also in the other tests, that is the models struggle to predict the classes in the extremities. Indeed, in this case, class 3 is detected pretty well by the algorithms while the first two (0 and 1) are still confused between them. Finally, it should be noted that class 2, compared to the others, is rarely recognized.

#### 4.2.2.5 Classification experiment considering the discretization of the target variable in 7 custom intervals

Since experiments with classes of the same frequency led to results that were not entirely accurate and did not meet our expectations, it was decided to proceed by creating specific

prediction intervals, considering the foreground. The classes and the respective distributions of the records for this specific case are reported in figure 4.15.

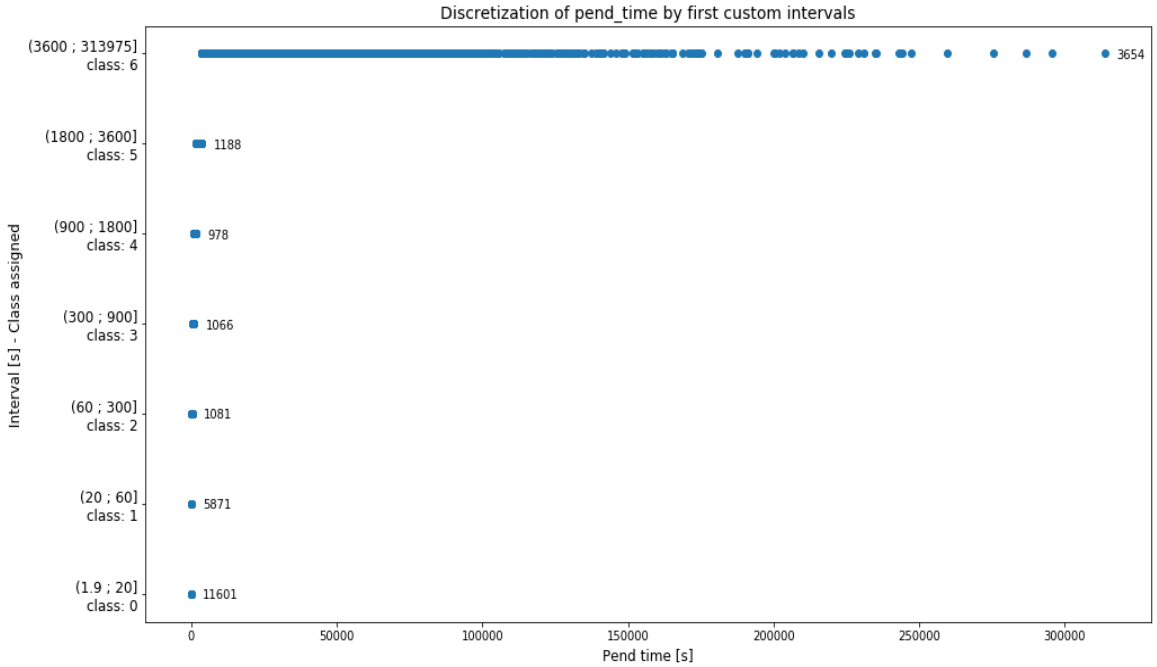



**Figure 4.15.** Visualization of the 7 custom classes generated and the distribution of the records inside them.

It can be immediately pointed out that class 0 is extremely populous compared to the others. To better evaluate the behaviour of the selected algorithms, a stratified 20-fold cross validation has been performed in order to compute accuracy, precision, recall and F-measure indices, whose mean values are shown in the table 4.18.

**Table 4.18.** Results of the experiment with 7 custom classes

| Algorithm | Mean Accuracy | Mean Precision | Mean Recall | Mean F-measure |
|---|---|---|---|---|
| SVM Classifier | 0.567593 | 0.229631 | 0.255881 | 0.204707 |
| KNN Classifier | 0.573686 | 0.326979 | 0.293869 | 0.275410 |
| Decision Tree | 0.588427 | 0.369179 | 0.322930 | 0.319689 |
| Random Forest | 0.570816 | 0.352834 | 0.319275 | 0.323275 |
| MLP Classifier | 0.586187 | 0.356230 | 0.312460 | 0.305662 |

From the obtained values it can be noted that the difference between accuracy and F-measure is increased compared to what happened in the experiments discussed in sections 4.2.2.1, 4.2.2.2, 4.2.2.3 and 4.2.2.4. To try to understand why this difference is increase, the confusion matrices generated for this specific case are shown in figure 4.16.


**Figure 4.16.** Confusion matrices generated for each model during the experiment with 7 custom classes.

From these representations it is clear why certain results have been obtained. In fact, all classifiers manage to predict fairly well the first and partly even the last class. The same cannot be said, however, for the remaining five central classes where the errors are many and therefore lower the prediction indices shown in the table 4.18. Comparing with the case analysed in section 4.2.2.3 in which there were also 7 prediction classes, the most marked difference is given by the ranges associated with these and, consequently, the presence or lack of homogeneity within them. In fact, it is entirely plausible that a model tends to predict more frequently the class that has the most records associated with it.

#### 4.2.2.6 Classification experiment considering the discretization of the target variable in 6 custom intervals

Later, attempts were made to reduce the number of classes to be predicted, from seven to six, also trying to make them more homogeneous. The new intervals and new distributions of records used in the experiment are shown in figure 4.17.

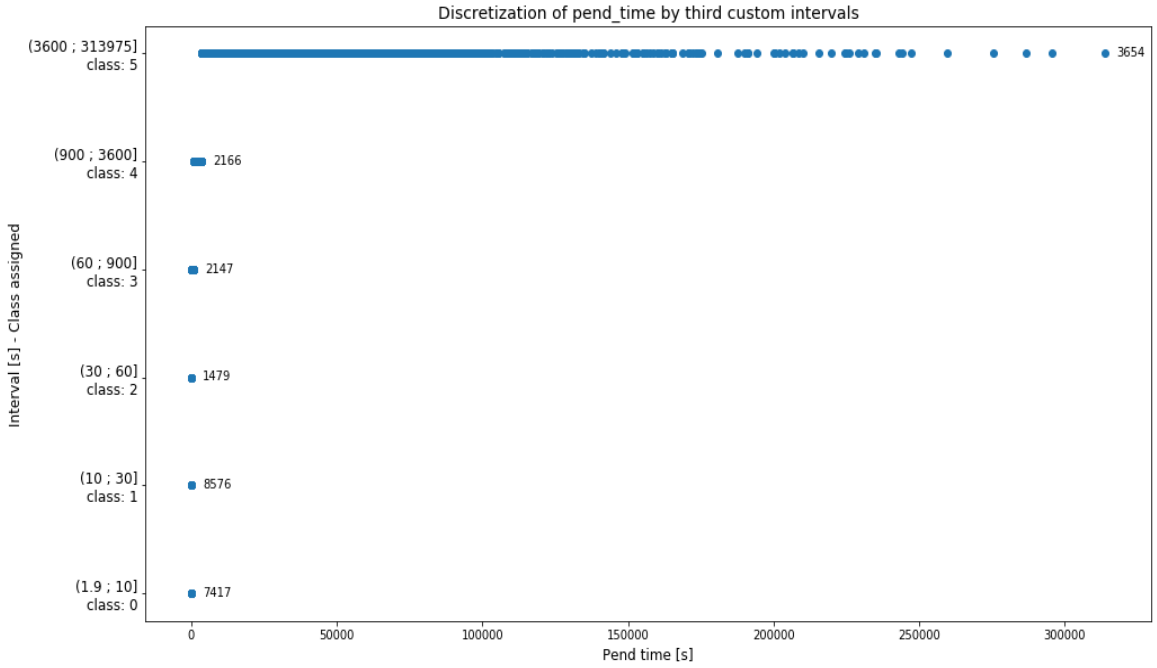



**Figure 4.17.** Visualization of the 6 custom classes generated and the distribution of the records inside them.

A stratified 20-fold cross validation was performed for each model available and the accuracy, precision, recall and F-measure indices were computed, the mean values of which are shown in the table 4.19.

**Table 4.19.** Results of the experiment with 6 custom classes

| Algorithm | Mean Accuracy | Mean Precision | Mean Recall | Mean F-measure |
|---|---|---|---|---|
| SVM Classifier | 0.492865 | 0.406168 | 0.369085 | 0.365455 |
| KNN Classifier | 0.418295 | 0.349598 | 0.340961 | 0.323907 |
| Decision Tree | 0.502064 | 0.410488 | 0.386844 | 0.385265 |
| Random Forest | 0.490782 | 0.404485 | 0.377412 | 0.380419 |
| MLP Classifier | 0.503164 | 0.407425 | 0.390228 | 0.390795 |

Through these data it is possible to say that the situation has improved compared to the previous case analysed in section 4.2.2.5, but not enough since the results are not yet satisfactory. Figure 4.18 shows the confusion matrices related to this experiment, used to define other case studies to improve the prediction performance of models.


**Figure 4.18.** Confusion matrices generated for each model during the experiment with 6 custom classes.

From these matrices it is evident that all models have problems in predicting class 2 while, as for the other classes, errors are made considering the neighbouring classes. In fact, dense areas are created at the first two and last three classes.

#### 4.2.2.7 Classification experiment considering the discretization of the target variable in 5 custom intervals

To try to improve the results obtained in previous experiments, an attempt was made to reduce the number of classes even further, also modifying the ranges of associated intervals. In this way we created five distinct classes, whose intervals and distributions of records are shown in figure 4.19.

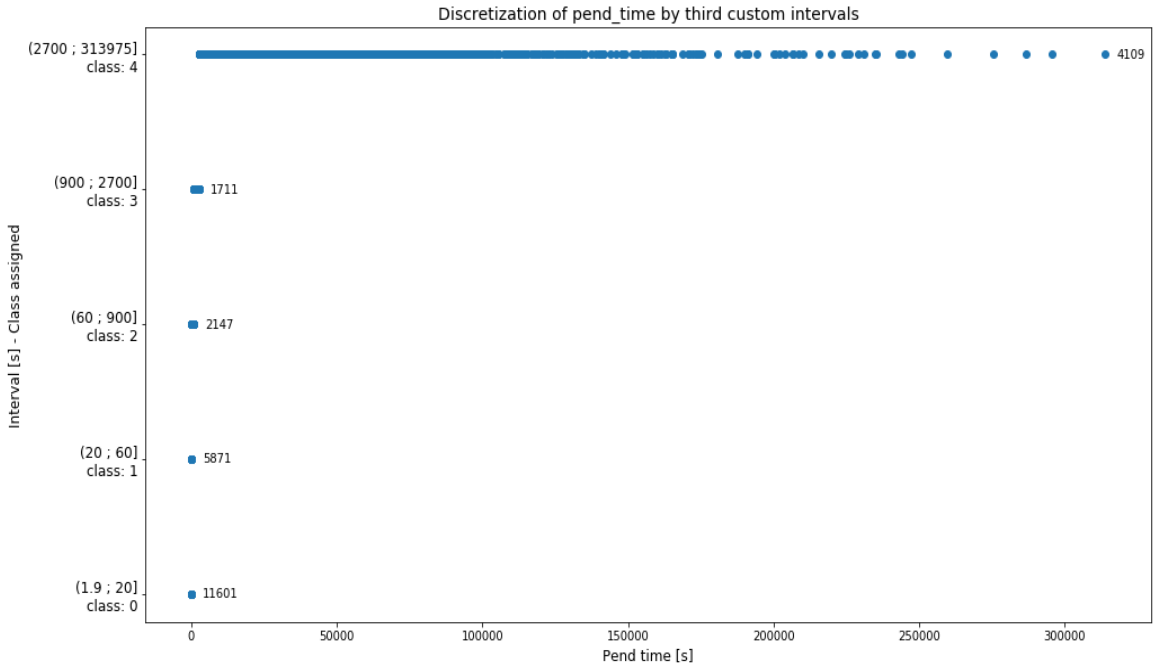



**Figure 4.19.** Visualization of the 5 custom classes generated and the distribution of the records inside them.

From this graph we can immediately notice that the first two classes are the most populous and this could lead the algorithms to make mistakes during the classification. To test the performance of the models a stratified 20-fold cross validation has been performed, to be able to compute the accuracy, precision, recall and F-measure indices whose mean values are reported in the table 4.20.

**Table 4.20.** Results of the experiment with 5 custom classes

| Algorithm | Mean Accuracy | Mean Precision | Mean Recall | Mean F-measure |
|---|---|---|---|---|
| SVM Classifier | 0.579347 | 0.431264 | 0.363045 | 0.317228 |
| KNN Classifier | 0.588270 | 0.447292 | 0.399713 | 0.378983 |
| Decision Tree | 0.610794 | 0.511985 | 0.445486 | 0.444375 |
| Random Forest | 0.593380 | 0.475918 | 0.431792 | 0.436247 |
| MLP Classifier | 0.606746 | 0.508933 | 0.440080 | 0.436944 |

The results, as it is possible to see, always tend to improve, but not enough. In fact, the value of the average F-measure for each model is always below 50% which is still low as threshold. Figure 4.20 shows the confusion matrices generated for this specific case in order to better analyse the situation.


**Figure 4.20.** Confusion matrices generated for each model during the experiment with 5 custom classes.

As already mentioned, it can be noted that algorithms have difficulty distinguishing between class 0 and class 1, which implies that these two classes often have the same values in the input variables used. In addition, central labels tend to be always the worst predicted, which leads to the lowering of the general performance statistics.

#### 4.2.2.8 Classification experiment considering the discretization of the target variable in 3 custom intervals

Finally, three distinct classes were generated, summing the ranges used in the experiment discussed in section 4.2.2.7, to use the information provided by the models. The ranges and their distributions are shown in figure 4.21.

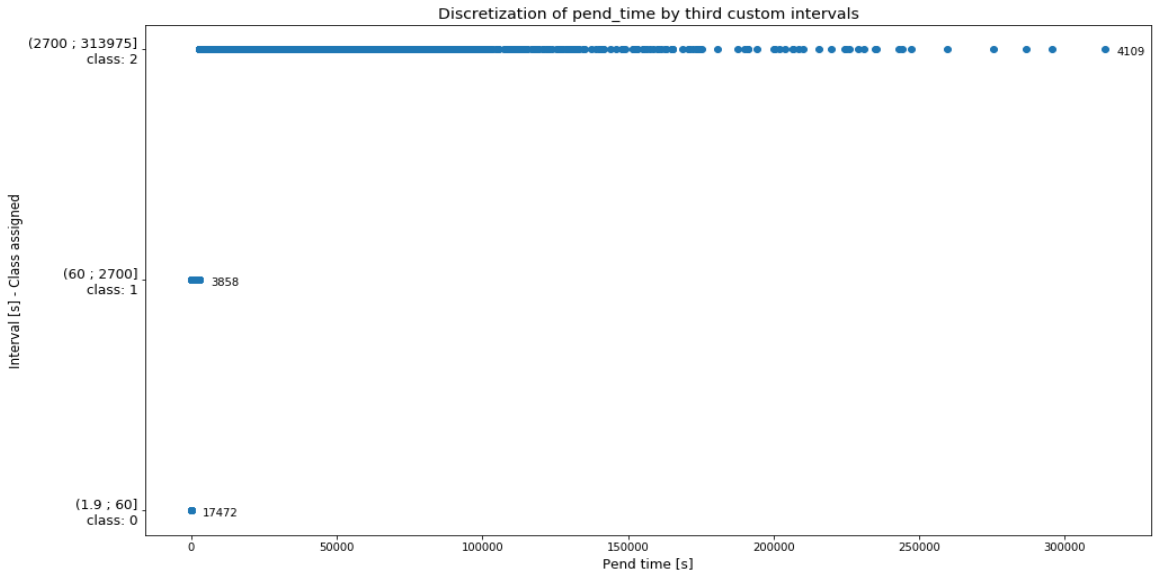



**Figure 4.21.** Visualization of the 3 custom classes generated and the distribution of the records inside them.

From this graph it can be noted that class 0 is the sum of the first two classes of the test analysed in section 4.2.2.7, class 1 is the sum of class 2 and 3, while the last one remains the same. To evaluate the performance of the models, a stratified 20-fold cross validation has been performed to compute the accuracy, precision, recall and F-measure indices, whose mean values are shown in the table 4.21.

**Table 4.21.** Results of the experiment with 3 custom classes

| Algorithm | Mean Accuracy | Mean Precision | Mean Recall | Mean F-measure |
|---|---|---|---|---|
| SVM Classifier | 0.814655 | 0.713570 | 0.611150 | 0.632459 |
| KNN Classifier | 0.811431 | 0.656376 | 0.627206 | 0.637632 |
| Decision Tree | 0.841149 | 0.732950 | 0.688357 | 0.706892 |
| Random Forest | 0.825937 | 0.702244 | 0.655989 | 0.673945 |
| MLP Classifier | 0.848972 | 0.759848 | 0.695855 | 0.721240 |

The results, as can be seen, are greatly improved even if they still do not meet the desired levels of correctness of prediction. Figure 4.22 shows the confusion matrices generated to try to understand the reason for the results obtained.


**Figure 4.22.** Confusion matrices generated for each model during the experiment with 3 custom classes.

What you can see from these matrices is that class 0 is predicted correctly in about 98% of cases. Class 1 is the most confusing. In fact, about a third is correctly predicted, while the two remaining are divided between the other two classes. Finally, the last class is correctly classified more than 60% of the time and is more confused with class 1. This means that prediction intervals are not yet suitable to distinguish significantly all generated classes.

### 4.2.3 Binary Classification Experiments

Whereas the accuracy values obtained from the multi-class classification experiments reported in section 4.2.2 have not yielded the desired results, it was decided to study the performance of selected classifiers using them for binary classifications. Doing so, possible pending time values

were identified so that the prediction indices of the two classes were meaningful.
These studies also gave rise to the idea of using hierarchical approaches to improve the overall performance of the prediction system. Indeed, as can be seen from the section 4.2.1, the first level of each hierarchical approach discussed always starts from a binary classification.

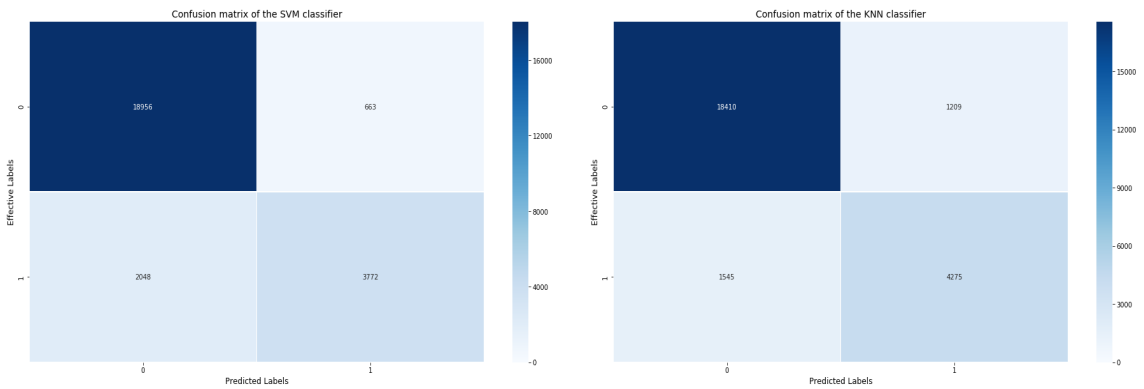
#### 4.2.3.1 Experiment using a threshold at 900 seconds

A threshold of 900 seconds was chosen as the first binary classification experiment. From this subdivision two ranges have been derived: the first has 19619 associated records, while the second the 5820 remaining.
To evaluate the performance of the models with these classes, a stratified 20-fold cross validation has been applied to compute the accuracy, precision, recall and F-measure indices, whose values, for each class, are reported in the table 4.22.

**Table 4.22.** Results of the experiment with a threshold at 900 seconds

| Algorithm | Mean Accuracy | Precision Class 0 | Precision Class 1 | Recall Class 0 | Recall Class 1 | F-measure Class 0 | F-measure Class 1 |
|---|---|---|---|---|---|---|---|
| SVM Classifier | 0.893431 | 0.902495 | 0.850507 | 0.966206 | 0.648110 | 0.933264 | 0.735641 |
| KNN Classifier | 0.891741 | 0.922576 | 0.779540 | 0.938376 | 0.734536 | 0.930409 | 0.756369 |
| Decision Tree | 0.899800 | 0.928292 | 0.796770 | 0.942912 | 0.754467 | 0.935545 | 0.775042 |
| Random Forest | 0.894925 | 0.918875 | 0.801957 | 0.947398 | 0.718041 | 0.932919 | 0.757683 |
| MLP Classifier | 0.894768 | 0.918280 | 0.802969 | 0.947908 | 0.715636 | 0.932859 | 0.756791 |

From these results it is evident that the first class is predicted better than the last. Most probably this situation is due to the fact that class 0 is more numerous and therefore it has been possible to do a more in-depth training compared to that for class 1. This aspect can also be noticed from the confusion matrices, shown in figure 4.23, generated for each individual model.



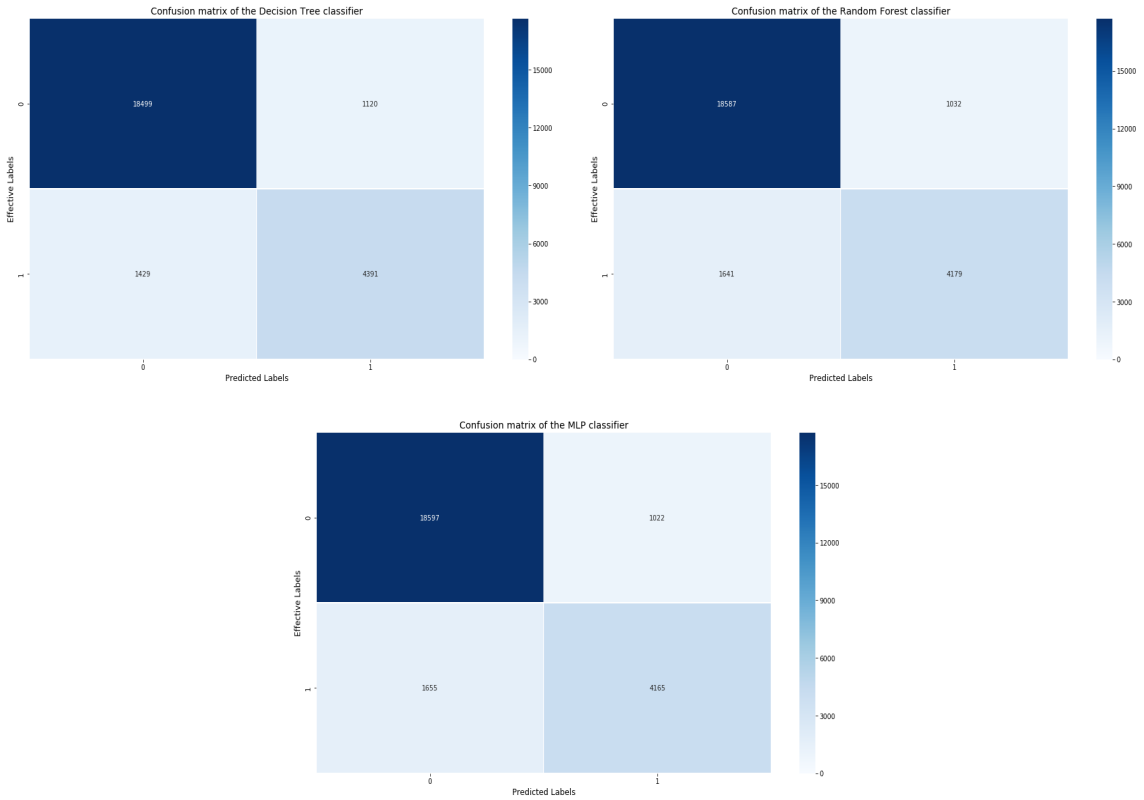



**Figure 4.23.** Confusion matrices generated for each model during the experiment with the threshold at 900 seconds.

#### 4.2.3.2 Experiment using a threshold at 1800 seconds

Another binary classification experiment performed was the one with the threshold at 1800 seconds and two intervals were created: the first with 20597 records associated, while the second with the remaining 4842.
To evaluate the selected algorithms, a stratified 20-fold cross validation was performed to compute the accuracy, precision, recall and F-measure indices. The values of these indices, some also related to the class, are reported in the table 4.23.

**Table 4.23.** Results of the experiment with a threshold at 1800 seconds

| Algorithm | Mean Accuracy | Precision Class 0 | Precision Class 1 | Recall Class 0 | Recall Class 1 | F-measure Class 0 | F-measure Class 1 |
|---|---|---|---|---|---|---|---|
| SVM Classifier | 0.901490 | 0.918637 | 0.804802 | 0.963684 | 0.636927 | 0.940621 | 0.711091 |
| KNN Classifier | 0.873580 | 0.922217 | 0.667422 | 0.921591 | 0.669352 | 0.921904 | 0.668386 |
| Decision Tree | 0.896183 | 0.920515 | 0.769137 | 0.954168 | 0.649525 | 0.937039 | 0.704289 |
| Random Forest | 0.889736 | 0.921651 | 0.734623 | 0.944070 | 0.658612 | 0.932726 | 0.694544 |
| MLP Classifier | 0.902197 | 0.923365 | 0.790474 | 0.958780 | 0.661504 | 0.940739 | 0.720261 |

These results show excellent performance of models in predicting class 0, performance that worsens when dealing with class 1. This difference in behaviour can be associated with the inhomogeneity of classes, meaning that the number of records associated with them are clearly different. Figure 4.24 shows the confusion matrices generated for this specific case.

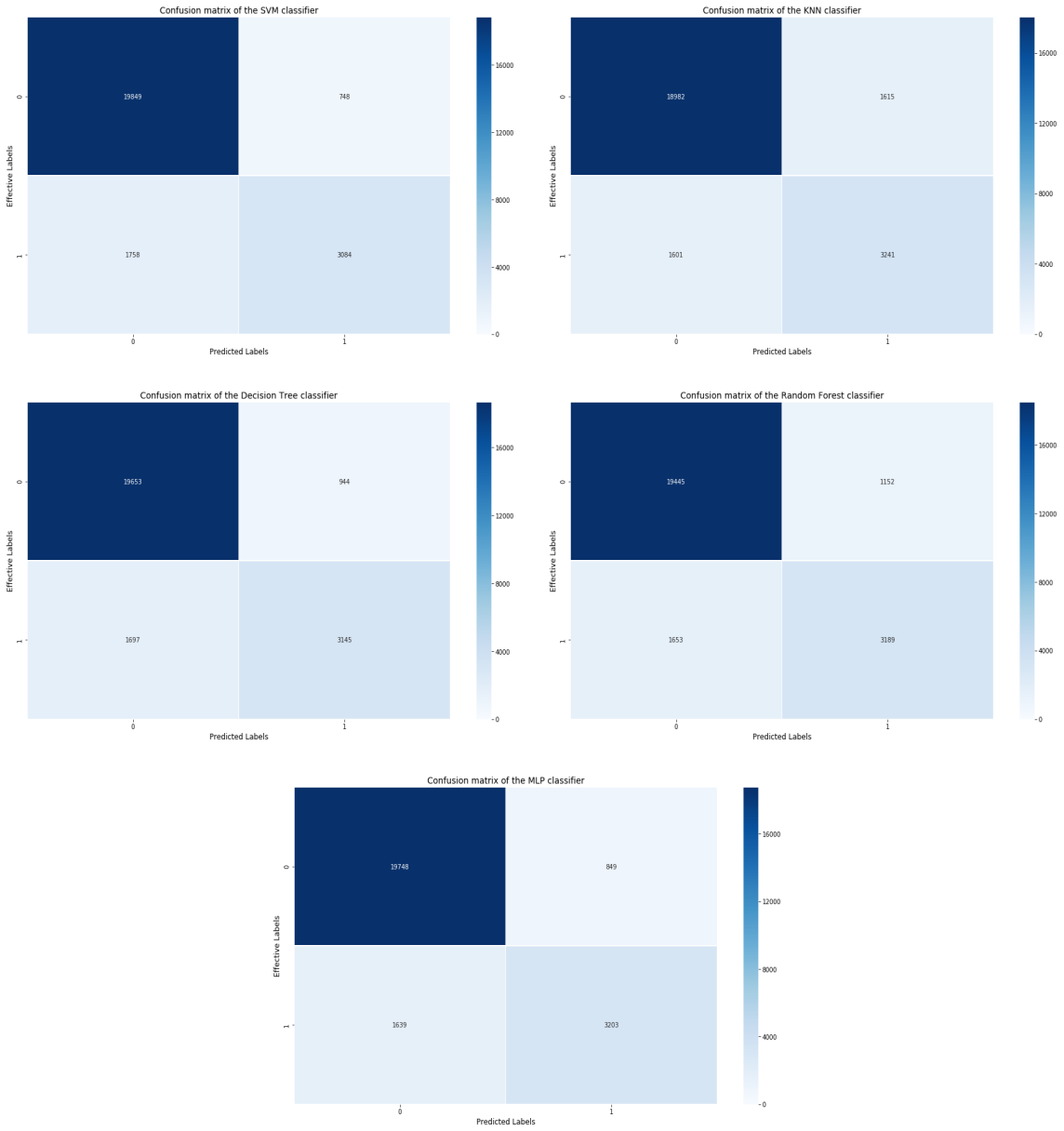



**Figure 4.24.** Confusion matrices generated for each model during the experiment with the threshold at 1800 seconds.

#### 4.2.3.3 Experiment using a threshold at 3600 seconds

To confirm what noticed in the cases analysed in section 4.2.3.1 and 4.2.3.2, it was decided to bring the limit to 3600 seconds by creating two intervals: the first containing 21785 records and the second the 3654 remaining. This accentuates the unevenness between the classes to understand how models react to these kinds of events.
For the evaluation of the algorithms, a stratified 20-fold cross validation has been used to be able to compute the accuracy, precision, recall and F-measure indices, whose values are brought back in the table 4.24.

**Table 4.24.** Results of the experiment with a threshold at 3600 seconds

| Algorithm | Mean Accuracy | Precision Class 0 | Precision Class 1 | Recall Class 0 | Recall Class 1 | F-measure Class 0 | F-measure Class 1 |
|---|---|---|---|---|---|---|---|
| SVM Classifier | 0.919612 | 0.933047 | 0.803929 | 0.976176 | 0.582373 | 0.954124 | 0.675447 |
| KNN Classifier | 0.874091 | 0.916786 | 0.571656 | 0.938123 | 0.492337 | 0.927332 | 0.529040 |
| Decision Tree | 0.911710 | 0.938586 | 0.722503 | 0.959697 | 0.625616 | 0.949024 | 0.670578 |
| Random Forest | 0.898817 | 0.934422 | 0.662260 | 0.948405 | 0.603175 | 0.941362 | 0.631338 |
| MLP Classifier | 0.915759 | 0.940285 | 0.741143 | 0.962773 | 0.635468 | 0.951396 | 0.684249 |

As can be seen from these data, comparing them with those obtained in the experiments analysed in sections 4.2.3.1 and 4.2.3.2, the unevenness of the classes increases the results of the indices with regard to the first class, while those of the second tend to decrease. Figure 4.25 shows the confusion matrices generated specifically for this experiment.

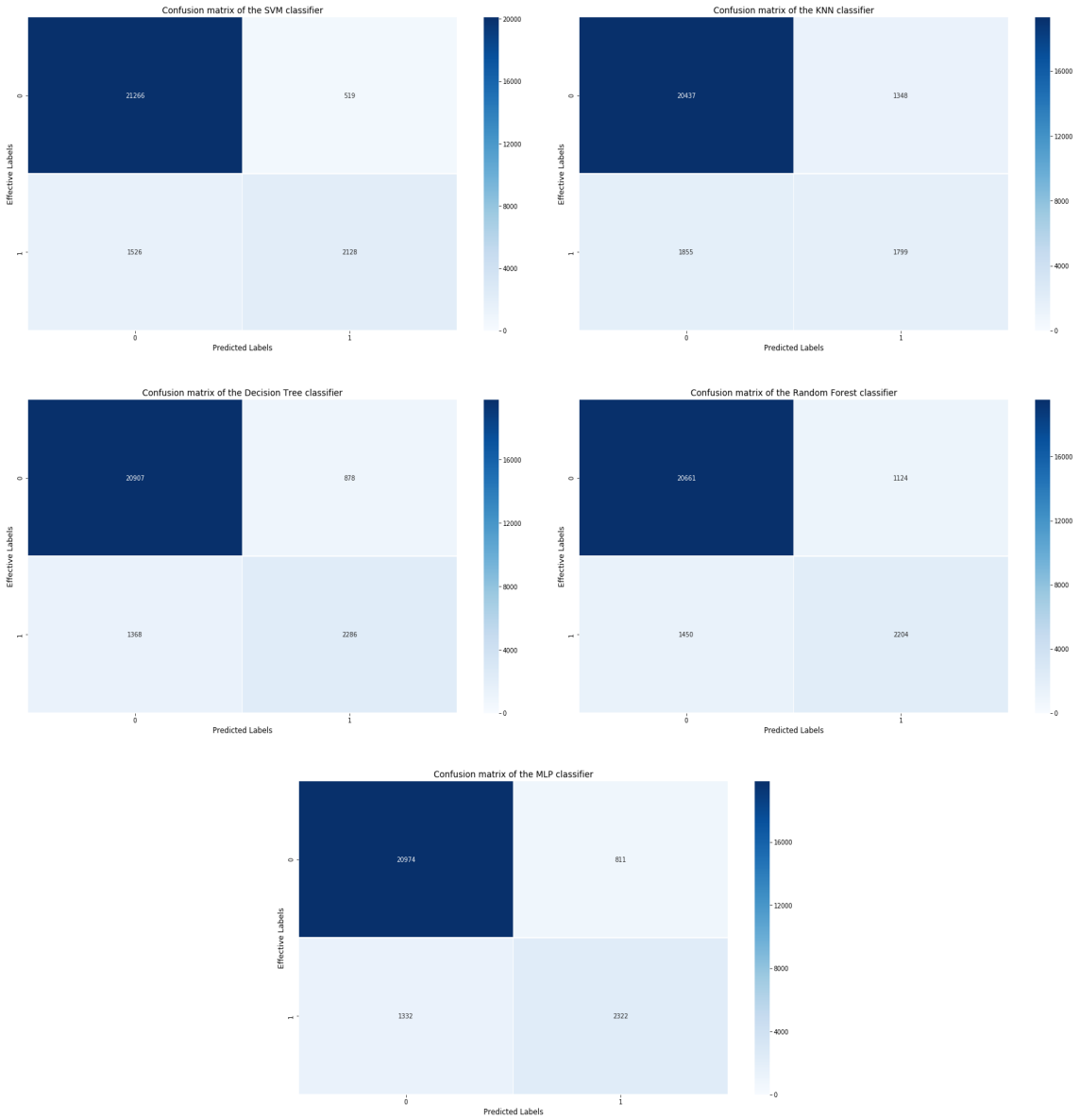



**Figure 4.25.** Confusion matrices generated for each model during the experiment with the threshold at 3600 seconds.

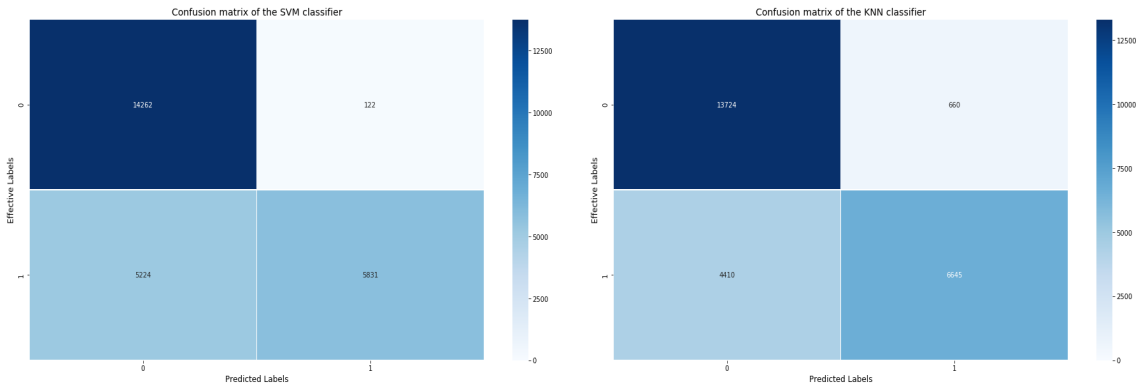
#### 4.2.3.4 Experiment using a threshold at 24 seconds or 582 seconds

To better identify the reliable thresholds, it was decided to use the classes and the confusion matrices of experiments discussed earlier in this chapter, summing the ranges of the various classes, to obtain two which to apply a binary classification.
The ranges of the first experiment of this type were generated from those used in the case analysed in section 4.2.2.4. The first class is given by the sum of the first two intervals while the second by the last two. Doing so, the threshold applied for this experiment is 24 seconds. The first class has associated 14384 records while the second has the remaining 11055.
To evaluate the selected algorithms a stratified 20-fold cross validation has been applied to compute the accuracy, precision, recall and F-measure indices, the values of which are shown in the table 4.25.

**Table 4.25.** Results of the experiment with a threshold at 24 seconds

| Algorithm | Mean Accuracy | Precision Class 0 | Precision Class 1 | Recall Class 0 | Recall Class 1 | F-measure Class 0 | F-measure Class 1 |
|---|---|---|---|---|---|---|---|
| SVM Classifier | 0.789850 | 0.731910 | 0.979506 | 0.991518 | 0.527454 | 0.842161 | 0.685678 |
| KNN Classifier | 0.800700 | 0.756810 | 0.909651 | 0.954116 | 0.601085 | 0.844086 | 0.723856 |
| Decision Tree | 0.805220 | 0.756935 | 0.930183 | 0.965587 | 0.596563 | 0.848624 | 0.726922 |
| Random Forest | 0.793231 | 0.754037 | 0.887315 | 0.941393 | 0.600670 | 0.837363 | 0.716383 |
| MLP Classifier | 0.806164 | 0.754647 | 0.945188 | 0.973790 | 0.588060 | 0.850326 | 0.725032 |

In this case it can be noted that, although the threshold makes the classes more or less homogeneous, many of the records predicted as belonging to class 0 actually belong to class 1. This can be seen through the confusion matrices generated and reported in the figure 4.26.



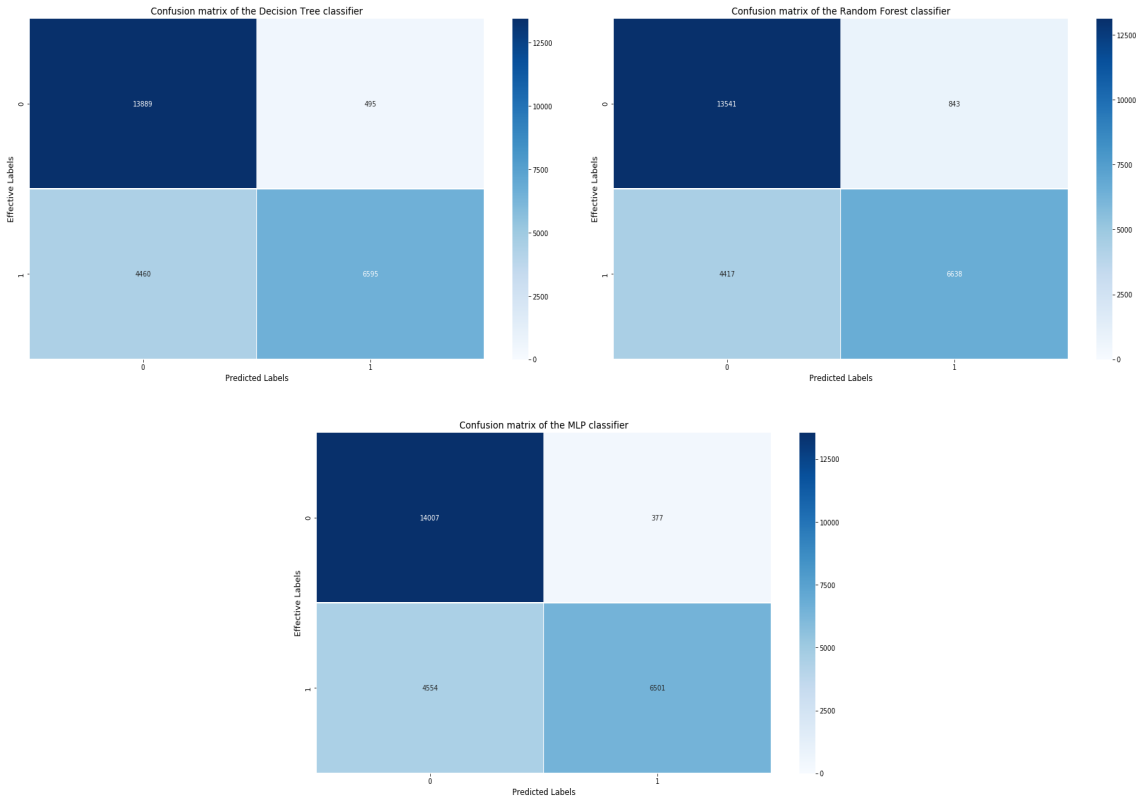



**Figure 4.26.** Confusion matrices generated for each model during the experiment with the threshold at 24 seconds.

Again, starting from the case analysed in section 4.2.2.4, it was decided to make a second attempt, summing the first three classes to create the first interval while the second corresponds to the last class. Doing so, the threshold used was placed at 582 seconds. The first interval has 19082 records while the other 6357.
After having carried out a training of the algorithms through a stratified 20-fold cross validation, the accuracy, precision, recall and F-measure indices have been generated and reported in the table 4.26.

**Table 4.26.** Results of the experiment with a threshold at 582 seconds

| Algorithm | Mean Accuracy | Precision Class 0 | Precision Class 1 | Recall Class 0 | Recall Class 1 | F-measure Class 0 | F-measure Class 1 |
|---|---|---|---|---|---|---|---|
| SVM Classifier | 0.883447 | 0.885408 | 0.874393 | 0.970181 | 0.623093 | 0.925858 | 0.727657 |
| KNN Classifier | 0.878502 | 0.930585 | 0.800605 | 0.934388 | 0.790782 | 0.932483 | 0.795663 |
| Decision Tree | 0.900900 | 0.927116 | 0.816920 | 0.941935 | 0.777725 | 0.934467 | 0.796841 |
| Random Forest | 0.890287 | 0.906888 | 0.828967 | 0.951420 | 0.706780 | 0.928620 | 0.763013 |
| MLP Classifier | 0.899406 | 0.920197 | 0.828660 | 0.948119 | 0.753185 | 0.933949 | 0.789122 |

These data were found to be better than those obtained with the limit put to 24 seconds. In this case, however, the problem is slightly more uneven, having the class 0 more records for the training. Figure 4.27 shows the confusion matrices generated by this analysis case with the limit to 582 seconds.

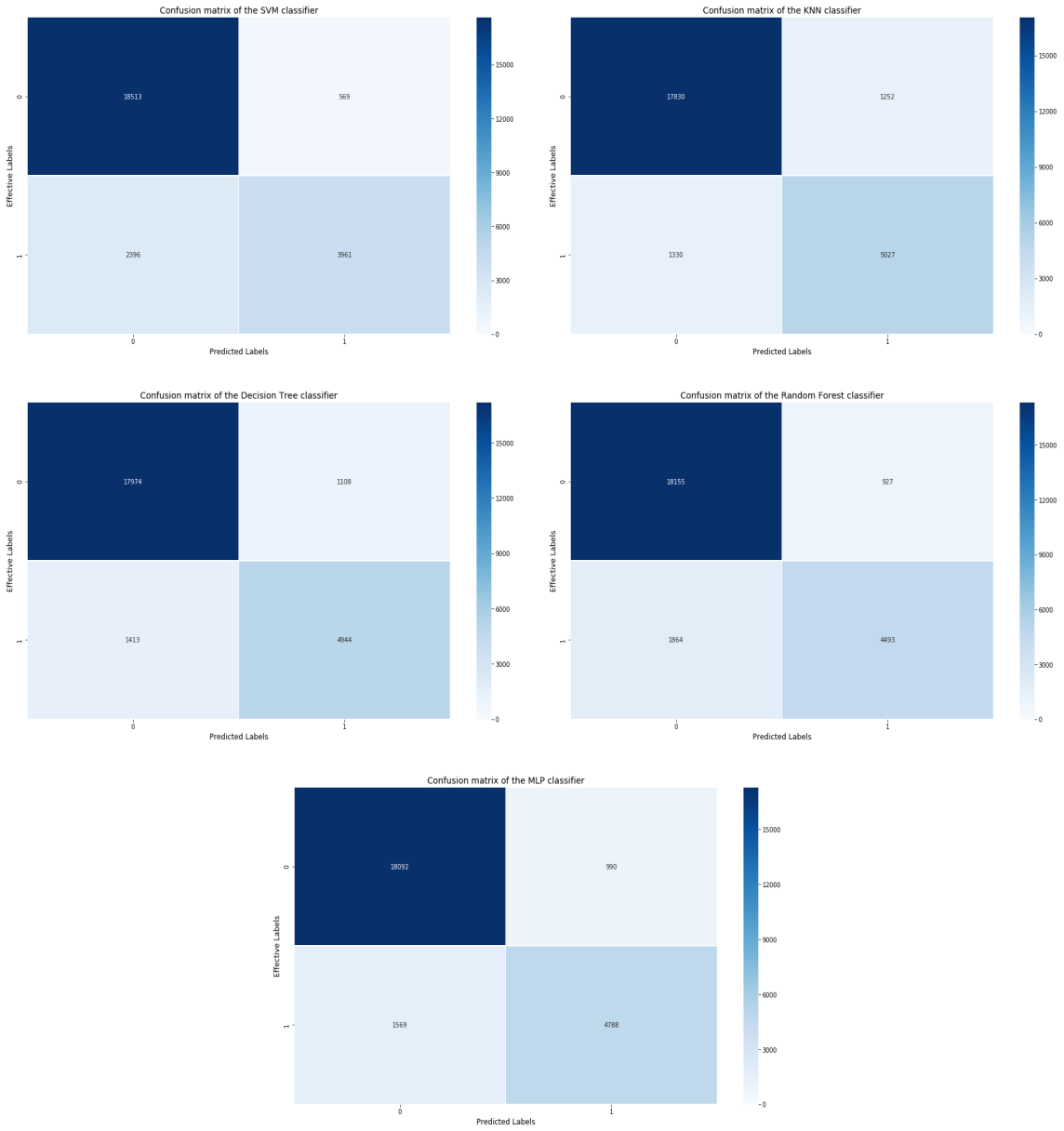



**Figure 4.27.** Confusion matrices generated for each model during the experiment with the threshold at 582 seconds.

#### 4.2.3.5 Experiment using a threshold at 30 seconds

Considering the results obtained in the experiments analysed in section 4.2.3.4, since the first case had a certain homogeneity of classes, it was decided to look for a threshold that was close to 24 seconds. Therefore, the new ranges were generated from the test analysed in section 4.2.2.6. In this case, the first class is the sum of the first two of the previous experiment, while the remaining two compose the second class. Doing so, the threshold is put to 30 seconds, creating a first class with 15993 records and a second with the remaining 9446.
After performing a stratified 20-fold cross validation, the accuracy, precision, recall and F-measure indices were computed to evaluate the performance of the models in this new situation, obtaining the results reported in the table 4.27.

**Table 4.27.** Results of the experiment with a threshold at 30 seconds

| Algorithm | Mean Accuracy | Precision Class 0 | Precision Class 1 | Recall Class 0 | Recall Class 1 | F-measure Class 0 | F-measure Class 1 |
|---|---|---|---|---|---|---|---|
| SVM Classifier | 0.837572 | 0.796599 | 0.988060 | 0.995936 | 0.569447 | 0.885184 | 0.722498 |
| KNN Classifier | 0.860254 | 0.834751 | 0.929311 | 0.969674 | 0.674995 | 0.897168 | 0.781996 |
| Decision Tree | 0.869059 | 0.834655 | 0.968870 | 0.987307 | 0.668855 | 0.904586 | 0.791383 |
| Random Forest | 0.863910 | 0.833502 | 0.949790 | 0.979116 | 0.668855 | 0.900460 | 0.784943 |
| MLP Classifier | 0.868666 | 0.834002 | 0.969688 | 0.987682 | 0.667161 | 0.904360 | 0.790468 |

These data allow us to say that, in this way, the classes are predicted better than the case with the limit to 24 seconds (analysed in section 4.2.3.4), trying to maintain a minimum of homogeneity of the classes, although in this case the two differ by about 6000 units. In addition to generating the values of the indices shown in the table 4.27, the confusion matrices, shown in figure 4.28, were also created for each individual algorithm used in the training process.

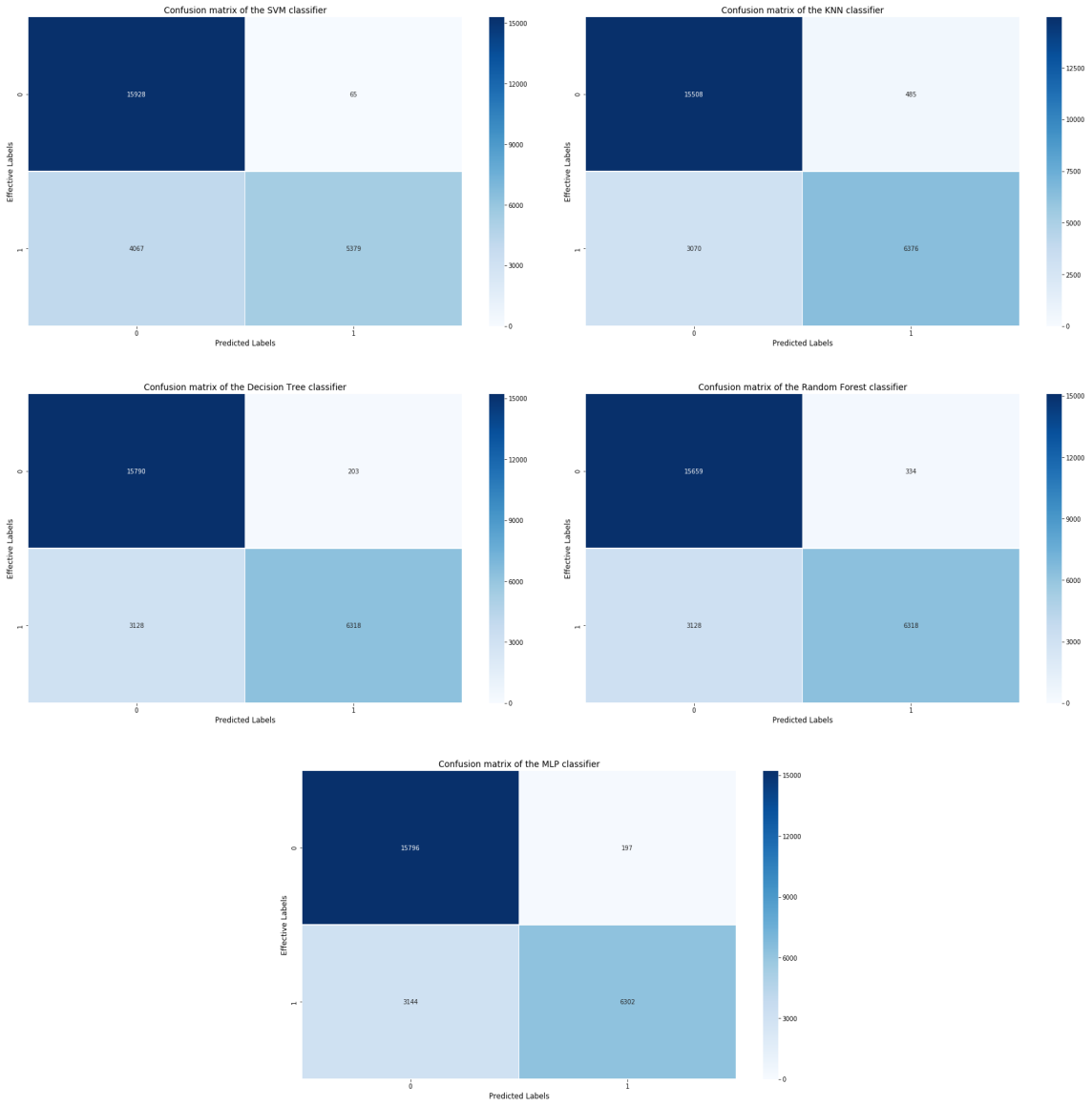



**Figure 4.28.** Confusion matrices generated for each model during the experiment with the threshold at 30 seconds.

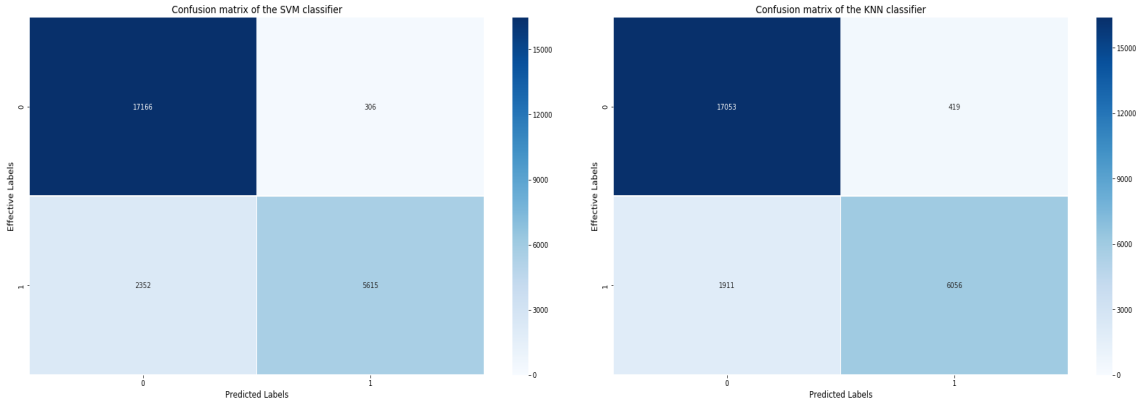
#### 4.2.3.6 Experiment using a threshold at 60 seconds

The classes of the last binary classification experiment consider those used in the case discussed in section 4.2.2.8. In fact, the first interval is identical to the first one generated in that test, while the second is given by summing the last two. Doing so, the threshold was set to 60 seconds, thus creating a first class containing 17472 records, while the second contains the remaining 7967.
To evaluate the performance of the chosen algorithms, a stratified 20-fold cross validation was applied and the accuracy, precision, recall and F-measure indices were computed, the values of which are shown in the table 4.28.

**Table 4.28.** Results of the experiment with a threshold at 60 seconds

| Algorithm | Mean Accuracy | Precision Class 0 | Precision Class 1 | Recall Class 0 | Recall Class 1 | F-measure Class 0 | F-measure Class 1 |
|---|---|---|---|---|---|---|---|
| SVM Classifier | 0.895515 | 0.879496 | 0.948320 | 0.976763 | 0.704782 | 0.925581 | 0.808612 |
| KNN Classifier | 0.908408 | 0.899230 | 0.935290 | 0.976019 | 0.760136 | 0.936052 | 0.838665 |
| Decision Tree | 0.904910 | 0.894905 | 0.934796 | 0.976190 | 0.748588 | 0.933782 | 0.831393 |
| Random Forest | 0.902551 | 0.891687 | 0.935556 | 0.976763 | 0.739802 | 0.932288 | 0.826243 |
| MLP Classifier | 0.908959 | 0.896712 | 0.945873 | 0.980369 | 0.752353 | 0.936676 | 0.838087 |

From these data we can see a remarkable improvement in the prediction of class 1, in which we get about 83% of the correct predictions, while the one of the class 0 is stable around 93%. Considering the above, this proved to be the best experiment among those performed. To confirm the data reported in the table 4.28, in figure 4.29 are reported the confusion matrices for each model, to be able to better understand how the errors are distributed.



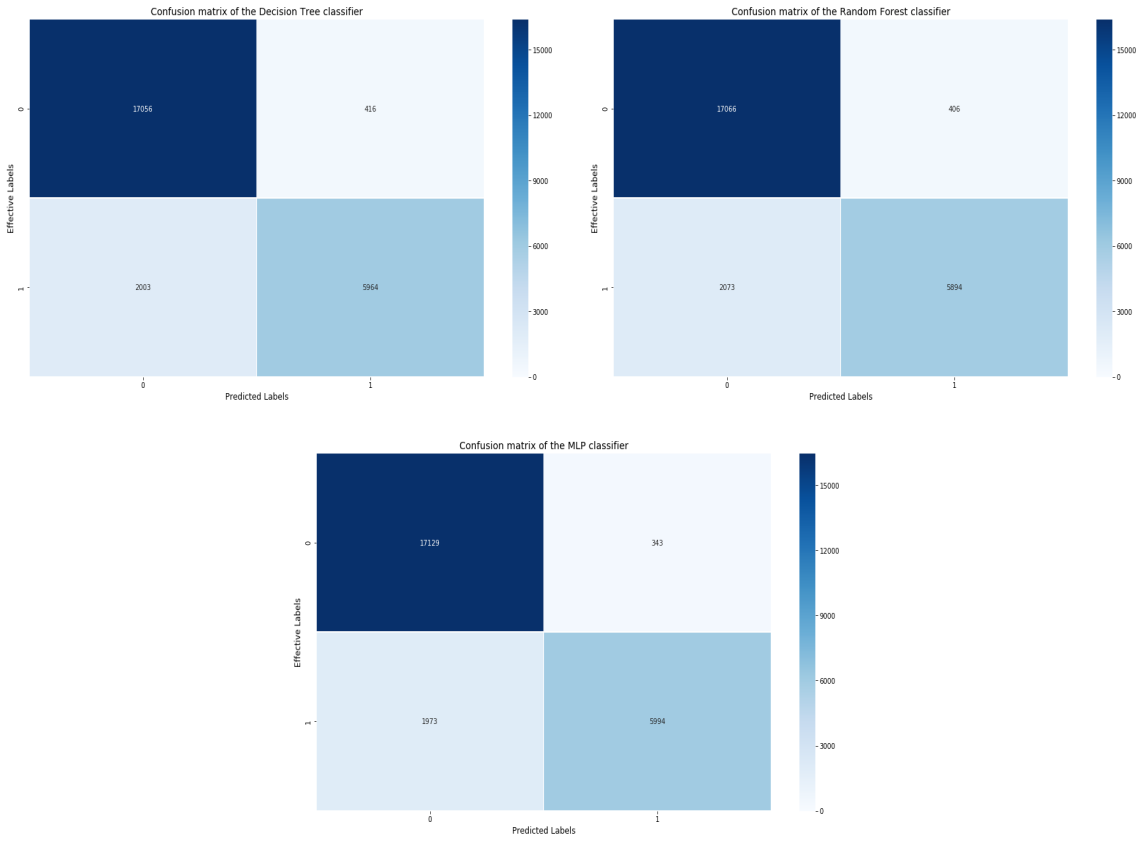



**Figure 4.29.** Confusion matrices generated for each model during the experiment with the threshold at 60 seconds.

From these matrices it is more noticeable that prediction problems are mainly present in class 1. In fact, all algorithms still struggle to distinguish clearly the records belonging to this class, classifying them as belonging to class 0.

# 5. Conclusions and Future Work

From the accuracy, precision, recall and F-measure indices obtained during the implementation of the solution analysed in section 3.5.1, it was noted that some classes are predicted better than others. In fact, through the data reported in the tables 4.12, 4.28 and those obtainable from the Decision Tree confusion matrix reported in figure 4.6, class 0 is predicted correctly about 94% of the time, class 1.0.0 about 62%, class 1.0.1 about 63%, class 1.1.0 about 39%, class 1.1.1 about 23% and class 1.1.2 about 80% of the time. It is extremely clear that classes 1.1.0 and 1.1.1 have unacceptable prediction results as they do not even reach the 50% threshold. Classes 1.0.0 and 1.0.1 are also not acceptable as they do not exceed 65% of correct classifications. Therefore, as already found in some classification experiments discussed in section 4.2.2, only the classes at the extremes (class 0 and 1.1.2) have optimal performances in the first case and medium in the second.
Additional information, which can help you understand why there are switches between excellent performance and poor ones, is the number of records used for the prediction of each class:

- Class 0, predicted through the first classification level of the system implemented, presents 17472 records associated with it, allowing the system to understand how to correctly distinguish jobs that have a pending time belonging to it.

- For the prediction of classes 1.0.0 and 1.0.1 only 2101 records are available, which must then be further subdivided into the two classes. In fact, class 1.0.0 has 966 records associated while the remaining 1125 are part of the other one.

- Finally, for the three 1.1.0, 1.1.1 and 1.1.2 it was only possible to use 2137 records. Of these 574 belong to the first, 331 to the second and the remaining 1375 to the third.

As it is possible to see from the above data, it is clear that one of the main problems that led to this disparity of performance between classes is the heterogeneity of these compared to the total of records used for their training. There is, in fact, a lack of data availability, as the records that were used to predict the last five classes are about a fifth compared to those used to predict only the first.
Another aspect that has negatively affected the classification capabilities of the system, regarding specific classes, is the presence of a bias inside the data, found following interviews done to users when trying to understand the nature of the data available. In fact, the main problem is that, often, users performed checks on the HPC cluster status before submitting a new job, which was subdued only if the resources needed to run it were available, with the result that rarely the architecture was overloaded, i.e. many jobs are waiting to be executed. Doing so, therefore, job waiting times for most cases turn out to be extremely low.
Moreover, it has been noted that the variables used for the multiple prediction levels, that make up the system, fail to perfectly describe the situation and therefore the problem appears to be under-modeled. This means that the variables used, for the prediction of job ranges, are not sufficient to describe optimally the problem and therefore to the algorithms lack the essential information to improve the classification indices of the system. In this regard it was noted that there are little data available that could describe the complexity of the job. From this, a lack of information has arisen, for those jobs submitted subsequently, regarding the resources time

of use.

Considering the prediction system implemented and analysed in section 3.5.1, it is possible that in the future changes will applied to improve the performance of each individual class.
More specifically, the problem will be better analysed together with users and the architecture technicians to identify additional variables able to describe both the jobs submitted, to understand how long specific resources will be occupied, and the state of the HPC cluster. In these meetings will be analysed also the way how these new data can be found, understanding if these are extractable from the already available variables, through other feature engineering processes, or by searching inside PBS raw files or from input and output files of the jobs. As analysed in the article [29], in fact, it is possible to define clustering methods able to unite the various jobs, regardless the number of operations that make up the simulation. By doing so, it is possible to define other common variables that can characterize the records, thus trying to use this information to modify and improve the prediction system discussed in this thesis. Afterwards, we will also attempt to describe the processes of data collection to save the information in the most optimal formats.
In addition, in this period, users were asked no longer to check the resources availability and submit the jobs immediately. Doing so, can be possible to try to overload the architecture by increasing the variability of the attributes currently used for prediction. These data will be added to those still used, enriching the dataset information, allowing to improve the performance of those classes that the implemented system fails to classify correctly.
We will also try to perform other tuning procedures to provide algorithms with the optimal parameters able to make the most of their potential, trying to improve the prediction system also from this point of view.
Finally, when the desired results are obtained, tests will be carried out by applying the prediction process directly to the jobs submissions one. In this way, the differences resulting from the employment of this system can be analysed, to understand whether or not this will bring significant benefits.

# General References

## Bibliography


[2] R. Artusi, P. Verderio, and E. Marubini. "Bravais-Pearson and Spearman correlation coefficients: meaning, test of hypothesis and confidence interval". In: *The International Journal of Biological Markers* (2002).

[3] V Athira, Kichu John, and Nitha Leeladharan. "Outranking for Better Decision Making: A Case Study of Two-wheeler Crashes in Cochin City". In: Jan. 2018.

[4] Elena Baralis and Silvia Chiusano. *Database Management System*. Course Notes. Politecnico di Torino, IT, 2017–2018.

[6] Raouf Boutaba et al. "A comprehensive survey on machine learning for networking: evolution, applications and research opportunities". In: *Journal of Internet Services and Applications* (2018). URL: https://jisajournal.springeropen.com/articles/10.1186/s13174-018-0087-2.

[7] R. Bower. *Graph it!* Ed. by Prentice-Hall. New Jersey, 1992.

[9] L. W. Buckalew and W. H. Pearson. "Critical factors in the chi-squared test of independence: A technique for exploratory data analysis". In: *Bulletin of the Psychonomic Society* (1982).

[10] Danilo Bzdok, Martin Krzywinski, and Altman Naomi. *Machine Learning: Supervised Methods*. Jan. 2018.

[11] G. Cicchitelli. *Statistica: Principi e Metodi*. Ed. by Pearson-Education. 2008.

[12] W. S. Cleveland. *The elements of graphing data*. Ed. by Hobart Press. New Jersey, 1994.

[14] Ayon Dey. "Machine Learning: A Review". In: *International Journal of Computer Science and Information Technologies* (2016).

[16] A. Durio and E. D. Isaia. *Elementi di statistica descrittiva per l'analisi dei dati*. Course notes. Facoltà di Economia università di Torino, IT, 2009–2010.

[17] Fabio Fabris, Joao Pedro de Magalhaes, and Alex A. Freitas. "A review of supervised machine learning applied to ageing research". In: *Biogerontology* (Feb. 2017).

[18] Usama Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. "Knowledge Discovery and Data Mining: Towards a Unifying Framework". In: *AAAI www.aaai.org* (1996).

[21] Ashar Ghasemi and Saleh Zahediash. "Normality tests for statistical analysis: A guide for non-statisticians". In: *International journal of endocrinology metabolism* (2012).

[23] Kim Hae-Young. "Statistical notes for clinical researchers: Chi-Squared test and Fisher's exact test". In: *RDE* (Mar. 2017).

[24] Larry L. Havlicek and Nancy L. Peterson. "Robustness of the Pearson Correlation against Violations of assumptions". In: *Perceptual and Motor Skills* (1976).

[28] Vandana Korde. "Text Classification and Classifiers: a survey". In: *IJAIA Journal* (Mar. 2012).

[29] Chunhong Liu et al. "Failure prediction of tasks in the cloud at an earlier stage: a solution based on domain information mining". In: *Springer Link* (Feb. 2020). URL: https://link.springer.com/article/10.1007/s00607-020-00800-1?utm_source=toc.

[31] Mary L. McHugh. "The Chi-squared test of indipendence". In: *Hrčak* (May 2013).

[33] Anthony M. Middleton and Arjuna Chala. *HPCC Systems®: Introduction to HPCC (High-Performance Computing Cluster*. Dec. 2015. URL: http://cdn.hpccsystems.com/whitepapers/wp_introduction_HPCC.pdf.

[36] M. M. Mukaka. "Statistics corner: A guide to appropriate use of correlation coefficient in medical research". In: *Malawi Medical Journal* (Sept. 2012).

[39] P. Newbold, W. Carlson, and B. Thorne. *Statistica*. Ed. by Pearson-Prentice Hall. 2007.

[40] Derya Oztuna, Elhan Atilla Halil, and Ersoz Tüccar. "Investigation of Four Different Normality Tests in Terms of Type 1 Error Rate and Power under Different Distributions". In: *Tübitak* (2006).

[41] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[42] Thair Nu Phyu. *Survey of classification Techniques in Data Mining*. 2009.

[43] Roberto Piccolo. "Multiple HPC clusters optimization through peer scheduling". Bachelor. Politecnico di Torino, 2015.

[46] Hassan Ramchoun et al. "Multilayer Perceptron: Architecture Optimization and Training". In: *International Journal of Interactive Multimedia and Artificial Intelligence* (Jan. 2016).

[47] Sebastian Raschka and Vahid Mirjalili. *Python Machine Learning, Machine Learning and Deep Learning with Python, scikit-learn and TensorFlow 2*. Birmingham, UK: Packt Publishing Ltd, Dec. 2019.

[48] Alfred Rayner. *Knowledge Discovery: Enhancing Data Mining and Decision Support Integration*. Qualifying Dissertation. University of York, UK, July 2005.

[49] Nornadiah Mohd Razali and Bee Wah Yap. "Power comparisons of Shapiro-Wilk, Kolmogorov-Smirnov, Lilliefors and Anderson-Darling tests". In: *Journal of Statistical Modeling and Analytics* (2011). URL: https://www.researchgate.net/publication/267205556.

[52] Veridian Systems. *Portable Batch System - Administration guide*. Version Open Release 2.3. Mountain View, CA, Aug. 2000. URL: https://euler.phys.cmu.edu/cluster/openpbs-v2.3-admin.pdf.

[53] E. R. Tufte. *The visual display of quantitative information*. Ed. by Graphics Press. Connecticut, 1983.

[56] B. W. Yap and C. H. Sim. "Comparisons of various types of normality tests". In: *Journal of Statistical Computation and Simulation* (2011). URL: https://doi.org/10.1080/00949655.2010.520163.

[57] Minhaz Fahim Zibram. *CHI-squared Test of Indipendence*. 2007.

## Sitography

[1] Abhay. *A hands-on tutorial on the Perceptron learning algorithm*. June 2016. URL: http://abhay.harpale.net/blog/machine-learning/a-hands-on-tutorial-on-the-perceptron-learning-algorithm/.

[5] Volodymyr Bilyk. *Guide to unsupervised machine learning: 7 real life examples*. 2019. URL: https://theappsolutions.com/blog/development/unsupervised-machine-learning/.

[8] Jason Brawnlee. *Supervised and Unsupervised Machine Learning Algorithms*. Mar. 2016. URL: https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms/.

[13] CWI.it. *Cos'è l'High Performance Computing (HPC)*. Jan. 2016. URL: https://www.cwi.it/data-center/high-performance-computing-hpc/cose-lhigh-performance-computing-hpc-83881.

[15] K. Dhiraj. *Support Vector Regression in Python using SciKit-learn*. Mar. 2020. URL: https://heartbeat.fritz.ai/support-vector-regression-in-python-using-scikit-learn-89cc18e933b7.

[19] AI & Games. *Artificial Neural Network*. Nov. 2017. URL: http://volanairbsc.blogspot.com/2017/11/artificial-neural-network.html.

[20] Rohit Garg. *7 Types of Classification Algorithms*. Jan. 2018. URL: https://analyticsindiamag.com/7-types-classification-algorithms/.

[22] Prashant Gupta. *Cross-Validation in Machine Learning*. June 2017. URL: https://towardsdatascience.com/cross-validation-in-machine-learning-72924a69872f.

[25] Martin Heller. *Il machine learning, spiegato bene: cos'è, come funziona e quali strumenti si usano*. Jan. 2020. URL: https://www.cwi.it/tecnologie-emergenti/intelligenza-artificiale/machine-learning-124626.

[26] Martin Heller. *Unsupervised Learning Explained*. Aug. 2019. URL: https://www.infoworld.com/article/3429017/unsupervised-learning-explained.html.

[27] *Iveco website*. 2018. URL: https://www.iveco.com/Pages/welcome.html.

[30] Samaya Madhavan and Mark Sturdevant. *Learning regression algorithms using Python and SciKit-learn*. Dec. 2019. URL: https://developer.ibm.com/technologies/data-science/tutorials/learn-regression-algorithms-using-python-and-scikit-learn/.

[32] MeeTheSkilled. *Test chi quadro o test di Pearson: Goodness of fit*. Jan. 2018. URL: https://meetheskilled.com/chi-quadro-goodness-of-fit-test-pearson/.

[34] Sanatan Mishra. *Unsupervised Learning and Data Clustering*. May 2017. URL: https://towardsdatascience.com/unsupervised-learning-and-data-clustering-eeecb78b422a.

[35] Michelle Moyer et al. *PBS exit codes*. NASA. Aug. 2017. URL: https://www.nas.nasa.gov/hecc/support/kb/pbs-exit-codes_185.html.

[37] Daniel Nelson. *What is a Confusion Matrix?* Dec. 2019. URL: https://www.unite.ai/what-is-a-confusion-matrix/.

[38] NetApp. *What Is High-Performance Computing?* 2020. URL: https://www.netapp.com/us/info/what-is-high-performance-computing.aspx.

[44] Upasana Priyadarshiny. *Introduction to Classification Algorithms*. Oct. 2019. URL: https://dzone.com/articles/introduction-to-classification-algorithms.

[45] Judy T. Raj. *A beginner's guide to dimensionality reduction in Machine Learning*. Mar. 2019. URL: https://towardsdatascience.com/dimensionality-reduction-for-machine-learning-80a46c2ebb7e.

[50] Margaret Rouse. *Supervised Learning*. Sept. 2019. URL: https://searchenterpriseai.techtarget.com/definition/supervised-learning.

[51] Alakh Sethi. *Support Vector Regression Tutorial for Machine Learning*. Mar. 2020. URL: https://www.analyticsvidhya.com/blog/2020/03/support-vector-regression-tutorial-for-machine-learning/.

[54] Gowthamy Vaseekaran. *Machine Learning: Supervised Learning vs Unsupervised Learning*. Sept. 2018. URL: https://medium.com/@gowthamy/machine-learning-supervised-learning-vs-unsupervised-learning-f1658e12a780.

[55] Niruhan Viswarupan. *K-Means Data Clustering*. July 2017. URL: https://towardsdatascience.com/k-means-data-clustering-bce3335d2203.