POLITECNICO DI TORINO

Collegio di Ingegneria Gestionale

Master of Science Degree in Engineering and Management

Master of Science Thesis

Artificial Intelligence in Manufacturing Industry: Application of machine learning algorithms for Cutting-tool Wear Monitoring and Predictive Maintenance

Advisor:
Prof. Franco Lombardi

Co-Advisor:
Dott. Emiliano Traini

Candidate:
Chiara Morresi

24th October, 2018

Index

1. Introduction

In the Industry 4.0 era, artificial intelligence is transforming the manufacturing industry. With the advent of Internet of Things (IoT) and machine learning methods, manufacturing systems are able to monitor physical processes and make smart decisions through real-time communication and cooperation with humans, machines, sensors, and so forth. Artificial intelligence enables manufacturers to reduce equipment downtime, spot production defects, improve the supply chain, and shorten design times by using machine learning technologies which learn from experiences. Companies are able to leverage these innovations in order to monitor and gain deeper insight into their operations, turning a typical manufacturing facility into a smart factory. One of the last application of these technologies is the development of Predictive Maintenance systems. Predictive maintenance combines Industrial IoT technologies with machine learning to forecast the exact time in which manufacturing equipment will need maintenance, allowing problems to be solved and adaptive decisions to be made in a timely fashion.

1.1 Thesis objective

This Thesis will discuss the implementation of a milling Cutting-tool Wear Monitoring and Predictive Maintenance solution, built in the Microsoft Azure Machine Learning Studio platform.

Predictive maintenance (PdM) is a method to monitor the status of machinery in order to prevent expensive failures form occurring and to perform maintenance when it is really required. Traditionally, maintenance put organizations in front of a trade-off situation where they had to choose between maximizing the useful life of a part at the risk of machine downtime (run-to-failure) or attempting to maximize uptime through early replacement of potentially good parts (time-based preventive maintenance), which has been demonstrated to be ineffective for most equipment components. Predictive maintenance aims to break these trade-offs by empowering companies to eliminate both failures and wasteful maintenance by forecasting it ahead of time. Adoption of PdM allows to maximize useful life of assets by reducing the frequency of maintenance activities, avoiding unplanned breakdowns and eliminating unnecessary preventive maintenance. This results in substantial time and cost savings and higher system reliability.

Condition Monitoring (CM) is a major component of Predictive Maintenance. It is defined as the process of monitoring parameters of condition in machinery in order to identify a significant change which is indicative of a developing fault. In CM, Tool Condition Monitoring (TCM) investigates how these parameters affect tool wear. Tool wear describes the gradual failure of tools due to regular operation. It is a key characteristic of machining processes as it adversely affects the tool life, which is of foremost importance in metal cutting owing to its direct impact on the surface quality of the machined surface, its dimensional accuracy and, consequently, the economics of machining. In manufacturing industry, it is often a disadvantage to use the tool until its breakage occurs, since this failure reduces the work surface quality and leads to a significant loss of dimensional accuracy, which result in defective parts and waste of workpiece material. On the other hand, an excessive preventive replacement of tools involves higher costs and production time: it will not only require additional tools to be purchased, which are generally expensive, but also considerable time to change the tool. Hence, a Tool Wear

Monitoring system is indispensable for better machining productivity: by replacing worn tools in time, it is possible to improve the quality of machining parts and to reduce production cost and idle time, with a precise exploitation of a tool's lifetime.

In summary, the objective of the Thesis is to create a ML model to predict when the cutting-tool used in milling operations must be replaced in order to maximize its useful life.

1.2 Thesis structure

The Thesis is structured in three main parts: overview of the concept of artificial intelligence in manufacturing, description of use-case problem and implementation of cutting-tool wear monitoring model.

The first part will start introducing the concepts of Industry 4.0 and Industrial Internet of Things, tackling the benefits and opportunities associated to Smart Factories. Then, it will present the notion of artificial intelligence and will provide an outline of Machine learning methods, focusing on the Learning algorithms used in the model. Finally, it will describe the applications of artificial intelligence in manufacturing industry, in particular detailing the role and importance of Predictive Maintenance and Tool Condition Monitoring systems.

The second part will discuss the use-case problem. First, it will define the milling process and the cutting-tool wear measurement approaches and tool life criteria. Consequently, experiment and dataset description will follow: since machinery datasets are not so easy to find as many companies prefer not to give their equipment information to other parties, the "Milling Data Set" collected by the Prognostic Center of Excellence (NASA – PCoE) is chosen for the purpose of the Thesis. This part will end describing the platform used to elaborate the solution, Microsoft Azure Machine Learning Studio.

The third part will focus on the implementation of the TCM model for the milling cutter, which takes as input cutting parameters and sensors signals and it returns the tool wear condition as output. The implementation involves data preparation, model training and testing, and model improvement. The data preparation phase consists of feature extraction, data cleaning, data normalization, feature selection and data transformation. The phase of training is the learning phase of the model: it consists of training various machine learning algorithms with a training set and tested with the testing set to sort out the most performant among them for this particular case. Consequently, the model is subject to further improvement to refine its predictions using hyperparameter tuning and cross-validation techniques. Finally, this part will proceed with the interpretation of the results and will draw the conclusion of the work, focusing on the future of the smart manufacturing.

2. Artificial intelligence in Manufacturing Industry

In the Industry 4.0 era, artificial intelligence is transforming the manufacturing industry. With the advent of Internet of Things (IoT) and machine learning methods, manufacturing systems are able to monitor physical processes and make smart decisions through real-time communication and cooperation with humans, machines, sensors, and so forth. Artificial intelligence enables manufacturers to reduce equipment downtime, spot production defects, improve the supply chain, and shorten design times by using machine learning technologies which learn from experiences. Companies are able to leverage these new technologies in order to monitor and gain deeper insight into their operations, turning a typical manufacturing facility into a smart factory. One of the last application of these technologies is the development of predictive maintenance systems. Predictive maintenance combines intelligent production system technologies with machine learning to forecast the exact time in which manufacturing equipment will need maintenance, allowing problems to be solved and adaptive decisions to be made in a timely fashion.

2.1 Industry 4.0

The name Industry 4.0 refers to the forth Industrial Revolution, with the first three coming about through mechanization, electricity, and IT (***Figure 1***). From the first industrial revolution (mechanization through water and steam power) to the mass production and assembly lines using electricity in the second, the fourth industrial revolution will take what was started in the third with the adoption of computers and automation and enhance it with smart and autonomous systems fuelled by data and machine learning [1].

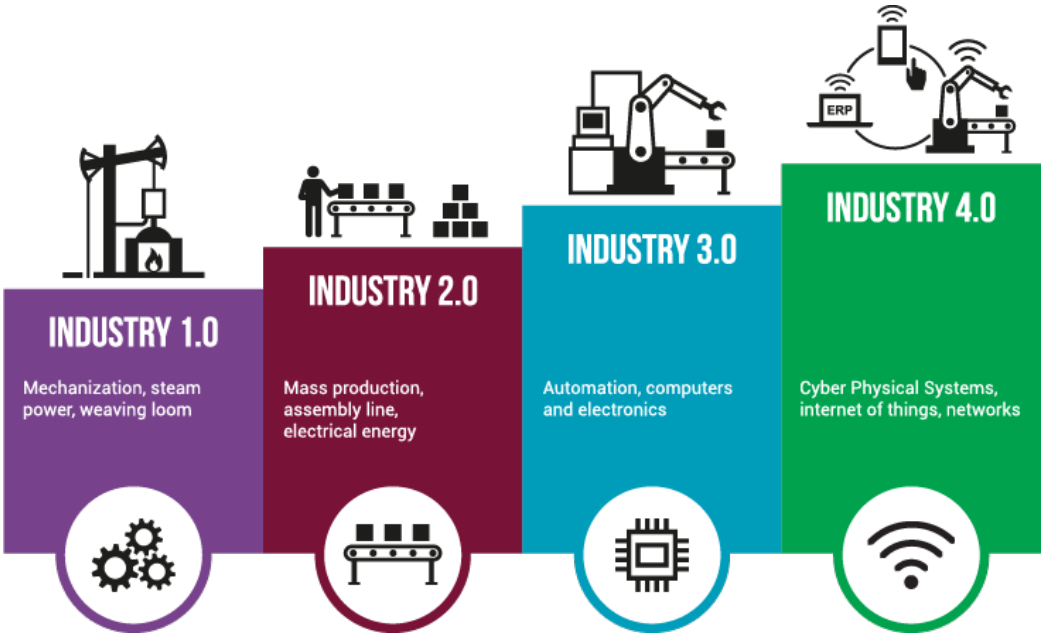


Figure 1*: Industrial revolutions and their driven forces*

The definition for Industry 4.0 was first introduced in 2011 at the Hannover Messe trade fair by the Germany Trade and Invest (GTAI) as: "*A paradigm shift made possible by technological advances which constitute a reversal of conventional production process logic. Simply put, this*

means that industrial production machinery no longer simply "processes" the product, but that the product communicates with the machinery to tell it exactly what to do" [2] GTAI further adds that Industry 4.0 represents "*the technological evolution from embedded systems to cyber-physical systems*," an approach that "*connects embedded production technologies and smart production processes*."

In other words, Industry 4.0 is a state in which manufacturing systems and the objects they create are not simply connected, drawing physical information into the digital realm, but also communicate, analyse, and use that information to drive further intelligent action back in the physical world to execute a physical-to-digital-to-physical transition. Now, and into the future as Industry 4.0 unfolds, computers are connected and communicate with one another to ultimately make decisions without human involvement. As a result of the support of smart machines that keep getting smarter as they get access to more data, our factories will become more efficient and productive and less wasteful.

To illuminate its definition of Industry 4.0, GTAI invokes the concept of Cyber-Physical Systems (CPS), which are technologies that marry the digital and physical worlds, typically via sensors affixed to physical devices and networking technologies that collect the resulting data. The vision of Industry 4.0 is that industrial businesses will build global networks to connect their machinery, factories, and warehousing facilities as cyber-physical systems, which will connect and control each other intelligently by sharing information that triggers actions. CPS monitor the physical processes of the factory and make decentralized decisions; they enable new capabilities in areas such as product design, prototyping and development, remote control, services and diagnosis, condition monitoring, pro-active and predictive maintenance, track and trace, structural health and systems health monitoring, planning, innovation capability, agility, real-time applications and more [3]. This concept is remarkably similar to the more commonly referenced Internet of Things (IoT). The physical systems become Internet of Things, communicating and cooperating both with each other and with humans in real time via the wireless web [4]. It's the network of these machines that are digitally connected with one another and create and share information that results in the true power of Industry 4.0.

2.1.1 Industrial Internet of Things

A key component of Industry 4.0 is the Internet of Things (IoT). IoT is the network of physical connected devices and other "things" embedded with electronics, software, sensors, actuators, and network connectivity which enable these objects to collect and exchange data. Each thing is uniquely identifiable through its embedded computing system but is able to interoperate within the existing Internet infrastructure. The IoT allows objects to be sensed or controlled remotely across existing network infrastructure, creating opportunities for more direct integration of the physical world into computer-based systems, and resulting in improved efficiency, accuracy and economic benefit in addition to reduced human intervention [5].

An IoT ecosystem consists of web-enabled smart devices that use embedded processors, sensors and communication hardware to collect, send and act on data they acquire from their environments. IoT devices share the sensor data they collect by connecting to an IoT gateway or other edge device where data is either sent to the cloud to be analysed or analysed locally. Sometimes, these devices communicate with other related devices and act on the information they get from one another [6].

The IoT concept has gained traction in recent years as the importance of connectivity has become better understood. Currently, a host of connected technologies is advancing rapidly, including high-quality sensors, more reliable and powerful networks, high-performance computing (HPC), robotics, artificial intelligence and cognitive technologies, and augmented reality. Taken together, these technologies can change manufacturing in profound ways. Increased connectivity and ever more sophisticated data-gathering and analytics capabilities enabled by the Internet of Things (IoT) have led to a shift toward an information-based economy. With the IoT, data, in addition to physical objects, are a source of value — and connectivity makes it possible to build smarter supply chains, manufacturing processes, and even end-to-end ecosystems.

Industry 4.0 represents an integration of IoT and relevant physical technologies to create stages of the physical-to-digital-to-physical (P-D-P) cycle (***Figure 2***). IoT translates physical actions from machines into digital signals using sensors, completing the first half of the P-D-P loop.

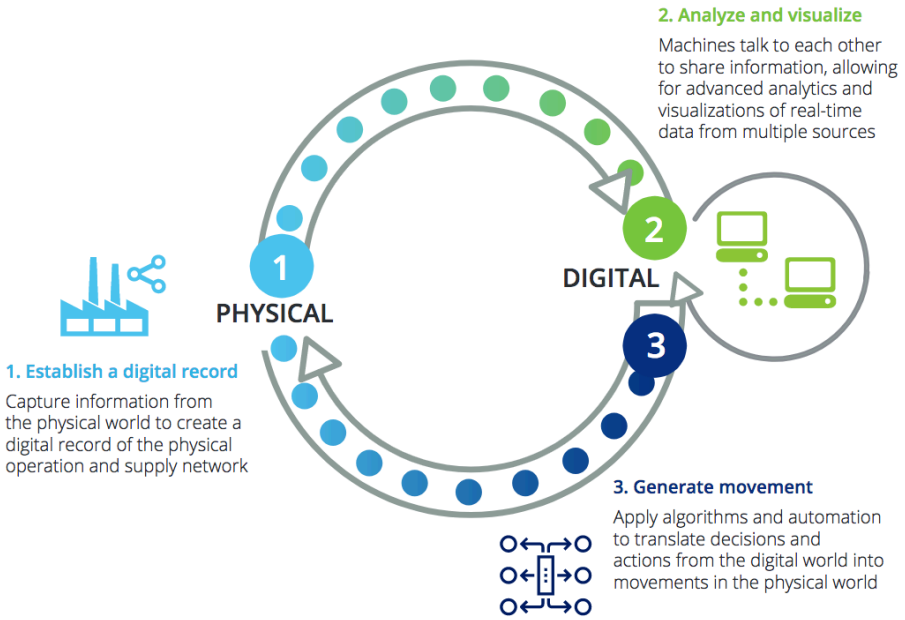


***Figure 2**: Physical-to-digital-to-physical cycle*

Once the physical actions have been translated into digital signals via sensors, they are processed, aggregated, and analysed. The second step in the P-D-P loop is to analyse and visualize the digital signals using advanced analytics and predictive algorithms. Advanced business intelligence (BI) tools are no longer only for data scientists. Many analytics platforms are beginning to incorporate high-level solutions for unstructured data, cognitive technologies, machine learning, and visualization. Finally, after the signals have been processed, analysed, and visualized, it's time to turn those insights back into physical action. In some cases, the digital conclusions drawn may instruct robots or machines to alter their functions. In other cases, maintenance alerts will spur a technician into action [7].

There are numerous real-world applications of the internet of things, ranging from consumer IoT and enterprise IoT to manufacturing and industrial IoT. The Industrial Internet of Things (IIoT) is the use of internet of things technologies to enhance manufacturing and industrial processes.

IIoT is a network of devices connected via communications technologies to form systems that monitor, collect, exchange and analyse data, delivering valuable insights that enable industrial companies to make smarter business decisions faster. It incorporates machine learning and big data technologies to harness the sensor data, machine-to-machine (M2M) communication and automation technologies that have existed in industrial settings for years [6]. An Industrial IoT system consists of:

- intelligent assets, i.e. applications, controllers, sensors and security components -- that can sense, communicate and store information about themselves;
- data communications infrastructure, e.g. the cloud;
- analytics and applications that generate business information from raw data; and
- people.

The IIoT provides a way to get better visibility and insight into the company's operations and assets though integration of machine sensors, middleware, software, and backend cloud compute and storage systems. Therefore, it provides a method of transforming business operational processes by using as feedback the results gained from interrogating large data sets though advanced analytics. The business gains are achieved though operational efficiency gains and accelerated productivity, which results in reduced unplanned downtime and optimized efficiency, and thereby profits [8]. The IIoT will revolutionize manufacturing by enabling the acquisition and accessibility of far greater amounts of data, at far greater speeds, and far more efficiently than before. It can greatly improve connectivity, efficiency, scalability, time savings, and cost savings for industrial organizations.

One of the top touted benefits the industrial internet of things affords businesses is Predictive Maintenance (PdM). This involves organizations using real-time data generated from IIoT systems to predict defects in machinery and tooling before they occur, enabling companies to take action to address those issues before a part fails or a machine goes down.

2.1.2 Smart factories

The heart of Industry 4.0 in conceptual terms is the Smart Factory and everything revolves around this central entity that makes up the business model. The smart factory represents a leap forward from more traditional automation to a fully connected and flexible system, which can use a constant stream of data from connected operations and production systems to learn and adapt to new demands [9]. A true smart factory can integrate data from system-wide physical, operational, and human assets to drive manufacturing, maintenance, inventory tracking, digitization of operations through the digital twin, and other types of activities across the entire manufacturing network. The result can be a more efficient and agile system, less production downtime, and a greater ability to predict and adjust to changes in the facility or broader network, possibly leading to better positioning in the competitive marketplace.

Through the application of artificial intelligence (AI) and increasing sophistication of cyber-physical systems that can combine physical machines and business processes, high levels of automation come as standard in the smart factory, including complex optimization decisions that humans typically make. The smart factory is a flexible system that can self-optimize performance across a broader network, self-adapt to and learn from new conditions in real or near-real time, and autonomously run entire production processes. Smart factories can operate

within the four walls of the factory, but they can also connect to a global network of similar production systems, and even to the digital supply network more broadly.

Data are the lifeblood of the smart factory. Through the power of algorithmic analyses, data drive all processes, detect operational errors, provide user feedback, and, when gathered in enough scale and scope, can be used to predict operational and asset inefficiencies or fluctuations in sourcing and demand. Data can take many forms and serve many purposes within the smart factory environment, such as discrete information about environmental conditions including humidity, temperature, or contaminants. How data are combined and processed, and the resulting actions, are what make them valuable. To power the smart factory, manufacturers should have the means to create and collect ongoing streams of data, manage and store the massive loads of information generated, and analyse and act upon them in varied, potentially sophisticated ways. Data might also represent a digital twin, a feature of an especially sophisticated smart factory configuration. At a high level, a digital twin provides a digital representation of the past and current behaviour of an object or process. The digital twin requires cumulative, real-world data measurements across an array of dimensions, including production, environmental, and product performance. The powerful processing capabilities of the digital twin may uncover insights on product or system performance that could suggest design and process changes in the physical world.

For a smart factory to function, assets—defined as plant equipment such as material handling systems, tooling, pumps, and valves—should be able to communicate with each other and with a central control system. These types of control systems can take the form of a manufacturing execution system or a digital supply network stack. The latter is an integrated, layered hub that functions as a single point of entry for data from across the smart factory and the broader digital supply network, aggregating and combining information to drive decisions. However, organizations will need to consider other technologies as well, including transaction and enterprise resource planning systems, IoT and analytics platforms, and requirements for edge processing and cloud storage, among others.

Process	Sample digitization opportunities
Manufacturing operations	• **Additive manufacturing** to produce rapid prototypes or low-volume spare parts • **Advanced planning and scheduling** using real-time production and inventory data to minimize waste and cycle time • **Cognitive bots and autonomous robots** to effectively execute routine processes at minimal cost with high accuracy • **Digital twin** to digitize an operation and move beyond automation and integration to predictive analyses
Warehouse operations	• **Augmented reality** to assist personnel with pick-and-place tasks • **Autonomous robots** to execute warehouse operations
Inventory tracking	• **Sensors** to track real-time movements and locations of raw materials, work-in-progress and finished goods, and high-value tooling • **Analytics** to optimize inventory on hand and automatically signal for replenishment
Quality	• In-line quality testing using **optical-based analytics** • **Real-time equipment monitoring** to predict potential quality issues
Maintenance	• **Augmented reality** to assist maintenance personnel in maintaining and repairing equipment • **Sensors** on equipment to drive predictive and cognitive maintenance analytics
Environmental, health, and safety	• **Sensors** to geofence dangerous equipment from operating in close proximity to personnel • **Sensors** on personnel to monitor environmental conditions, lack of movement, or other potential threats

Table 1 *- Smart factory production processes and Sample digitalization opportunities*

These technologies power the digital supply network and, by extension, the smart factory, creating new opportunities to digitize production processes. ***Table 1*** depicts a series of core smart factory production processes along with a series of sample opportunities for digitization enabled by various digital and physical technologies.

2.3 Artificial intelligence

Together with IIoT, artificial intelligence is one of the major element of Industry 4.0. Artificial intelligence (AI) is a branch of computer science that emphasizes the creation of intelligent machines that work and react like humans.

John McCarthy, who coined the term in 1956, defines it as "*the science and engineering of making intelligent machines*". The modern definition of artificial intelligence (or AI) is "*the study and design of intelligent agents*" where an intelligent agent is a system that perceives its environment and takes actions which maximizes its chances of successfully achieving its goals [10]. Colloquially, the term "artificial intelligence" is applied when a machine is programmed to think or act like a person and mimics cognitive functions that humans associate with other human minds. The goals of artificial intelligence include learning, reasoning and problem solving, and machines are wired using a cross-disciplinary approach based in mathematics, computer science, linguistics, psychology and more [11].

Artificial intelligence has become an essential part of the technology industry and research associated with it is highly technical and specialized. The overall research goal of AI is to create technology that allows computers and machines to function in an intelligent manner. The general problem of simulating (or creating) intelligence has been broken down into sub-problems. The core problems of artificial intelligence include programming computers for certain traits or capabilities that researchers expect an intelligent system to display, such as:

- Knowledge
- Reasoning
- Problem solving
- Perception
- Learning
- Planning
- Speech recognition
- Ability to manipulate and move objects

This paper focuses on machine learning methods through which a system is able to learn from data and predict future trends.

2.3.1 Machine learning methods

Machine learning (ML) is a core part of artificial intelligence which provides systems the ability to automatically learn and improve from experience without being explicitly programmed. Machine learning explores the study and construction of computer algorithms that can learn from and make prediction on data, where such algorithms overcome following strictly static program instructions by making data-driven predictions or decisions through building a model from sample inputs. Within the field of data analytics, machine learning is a method used to

devise complex models that lend themselves to prediction; in commercial use, this is known as predictive analytics. These analytical models allow researchers, data scientists, engineers, and analysts to "produce reliable, repeatable decisions and results" and uncover "hidden insights" through learning from historical relationships and trends in the data [12].

The process of learning begins with observations or data, such as examples, direct experience, or instruction, in order to look for patterns in data and make better decisions in the future based on the examples that are provided [13]. The basic premise of machine learning is to build algorithms that can receive input data and use statistical analysis to predict an output while updating outputs as new data becomes available, with the aim of allowing computers to learn automatically without human intervention or assistance and adjust actions accordingly.

The core objective of a machine learning algorithm, i.e. a learner, is to generalize from its experience. Generalization in this context is the ability of a learning machine to perform accurately on new, unseen examples after having experienced a learning data set. The training examples come from some generally unknown probability distribution (considered representative of the space of occurrences) and the learner has to build a general model about this space that enables it to produce sufficiently accurate predictions in new situations. For the best performance in the context of generalization, the complexity of the hypothesis should match the complexity of the function underlying the data. If the hypothesis is less complex than the function, then the model has under-fit the data. If the complexity of the model is increased in response, then the training error decreases. But if the hypothesis is too complex, then the model is subject to overfitting and generalization will be poorer.

ML is employed in a range of computing tasks where designing and programming explicit algorithms with good performance is difficult or infeasible Common machine learning use cases include personalized marketing, fraud detection, spam filtering, network security threat detection, predictive maintenance and building news feeds [14].

There are many different types of machine learning algorithms and they are typically classified into two groups based on the way they learn about data to make predictions: supervised and unsupervised learning (***Figure 3***).

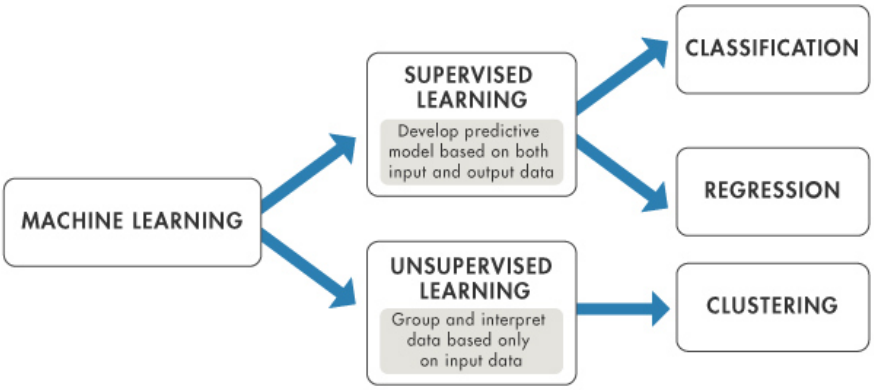


***Figure 3**: Classification of machine learning algorithms*

Choosing to use either a supervised or unsupervised machine learning algorithm typically depends on factors related to the structure and volume of your data and the use case of the issue at hand. In our use-case, supervised Machine Learning algorithms will be applied, both classification and regression task.

2.3.1.1 Supervised learning

In supervised learning, the computer is presented with example inputs and their desired outputs, given by a "teacher", and the goal is to learn a general rule that maps inputs to outputs. Supervised machine learning algorithms can apply what has been learned in the past to new data using labelled examples to predict future events. Starting from the analysis of a known training dataset, the learning algorithm produces an inferred function to make predictions about the output values. The system is able to provide targets for any new input after sufficient training. The learning algorithm can also compare its output with the correct, intended output and find errors in order to modify the model accordingly [13].

Supervised learning requires that the algorithm's possible outputs are already known and that the data used to train the algorithm is already labelled with correct answers. This happen when you have input variables (x) and an output variable (Y) and you use an algorithm to learn the mapping function from the input to the output [Y = f(X)]. The goal is to approximate the mapping function so well that when you have new input data (x) that you can predict the output variables (Y) for that data [15].

Supervised learning is so called because the data scientist acts as a guide to teach the algorithm what conclusions it should come up with. Data scientists determine which variables, or features, the model should analyse and use to develop predictions. The correct answer is known and the algorithm iteratively makes predictions on the training data and is corrected by the teacher. Learning stops when the algorithm achieves an acceptable level of performance. Once training is complete, the algorithm will apply what was learned to new data.

Supervised learning problems can be further grouped into classification and regression tasks.

- Classification: inputs are divided into two or more classes, and the learner must produce a model that assigns unseen inputs to one or more (multi-label classification) of these classes. It is used to determine what category something belongs in, after seeing a number of examples of things from several categories;
- Regression: consider the same problem as classification, but rather than considering discrete outputs, it returns continuous outputs (real number). It attempts to produce a function that describes the relationship between inputs and outputs and predicts how the outputs should change as the inputs change.

2.3.1.2 Unsupervised learning

Unsupervised learning studies how systems can infer a function to describe a hidden structure from unlabelled data. No labels are given to the learning algorithms, leaving it on its own to explores the data and find a structure in its inputs.

Unsupervised algorithms do not need to be trained with desired outcome data. Instead, they use an iterative approach called deep learning to review data and arrive at conclusions. Unsupervised learning algorithms are used for more complex processing tasks. They work by combing through millions of examples of training data and automatically identifying often subtle correlations between many variables. Once trained, the algorithm can use its bank of associations to interpret new data. These algorithms have only become feasible in the age of big data, as they require massive amounts of training data [14].

These are called unsupervised learning because, unlike supervised learning above, there is no correct answers and there is no teacher. Algorithms are left to their own devises to discover and present the interesting structure in the data. Unsupervised machine learning is more closely aligned with what some call true artificial intelligence, i.e. the idea that a computer can learn to identify complex processes and patterns without a human to provide guidance along the way. Although unsupervised learning is prohibitively complex for some simpler enterprise use cases, it opens the doors to solving problems that humans normally would not tackle [16].

Unsupervised learning problems can be further classified into clustering and association problems.

- Clustering: task of grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar to each other than to those in other groups;
- Association: discover frequent patterns, correlations, associations, or causal structures between variables in a dataset.

2.4 Intelligent manufacturing

Artificial intelligence found practical application in manufacturing industry. Intelligent manufacturing, also known as smart manufacturing, is a broad concept of manufacturing with the purpose of enhancing production and product transactions by making full use of advanced information and manufacturing technologies. Intelligent manufacturing means using the combined intelligence of people, processes and machines, to impact the overall economics of manufacturing. Its purpose is to optimize manufacturing resources, improve business value and safety, and reduce waste. It is regarded as a new manufacturing model based on artificial intelligent science and technology that greatly upgrades the design, production, management, and integration of the whole life cycle of a typical product. The entire product life cycle can be facilitated using various smart sensors, adaptive decision-making models, advanced materials, intelligent devices, and data analytics [17].

One form of realization of this concept is the intelligent manufacturing system (IMS), which is considered to be the next-generation manufacturing system that is obtained by adopting new models, new forms, and new methodologies to transform the traditional manufacturing system into a smart system. All IMS systems make use of the so called machine intelligence, i.e. sensor equipment, combined with machine learning techniques [18].

One of the most important research topics in the area of Intelligence Manufacturing is Predictive Maintenance (PdM) and in particular the implementation of Tool Condition Monitoring (TCM) systems. A TCM system is able to monitor an internal stay of the system and also changing conditions coming from environment. Monitoring systems are using sensors which are located at some proper place of the system, usually such place is the tool stand, machine or some manipulating device. Sensors are identifying parameters, which are then used as input data of control system.

2.4.1 Predictive maintenance

Predictive maintenance (PdM) plays an important role in modern manufacturing industry as it highly influences production quality and quantity and directly affects production cost and customer satisfaction.

All manufacturing systems will eventually fail if maintenance is not performed on them. Maintenance in the manufacturing industry faces a number of challenges, but the goal of any maintenance organization is always the same: to maximize asset availability. Today, poor maintenance strategies can reduce a plant's overall productive capacity by 5 to 20 percent. Recent studies also show that unplanned downtime is costing industrial manufacturers an estimated $50 billion each year. This begs the question, "*How often should a machine be taken offline to be serviced?*" Traditionally, this dilemma forced most maintenance organizations into a trade-off situation where they had to choose between maximizing the useful life of a part at the risk of machine downtime (run-to-failure) or attempting to maximize uptime through early replacement of potentially good parts (time-based preventive maintenance), which has been demonstrated to be ineffective for most equipment components. Predictive maintenance (PdM) aims to break these trade-offs by empowering companies to maximize the useful life of their parts while avoiding unplanned downtime and minimizing planned downtime.

PdM is a method to monitor the status of the machinery in order to prevent expensive failures form occurring and to perform maintenance when it is really required. Predictive maintenance directly monitors the condition and performance of equipment during normal operation to reduce the likelihood of failures. It attempts to keep costs low by reducing the frequency of maintenance tasks, reducing unplanned breakdowns and eliminating unnecessary preventive maintenance. The ultimate goal of the approach is to perform maintenance at a scheduled point in time when the maintenance activity is most cost-effective and before the equipment loses performance within a threshold.

Predictive maintenance differs from preventive maintenance because it relies on the actual condition of equipment, rather than average or expected life statistics, to predict when maintenance will be required. The "predictive" component of predictive maintenance stems from the goal of predicting the future trend of the equipment's condition [19] and is enabled by the advances in sensor and communication technologies that are part of the ongoing trends of automation and the Industrial Internet of Things (IIoT). This approach usually includes the continual data collection from equipment through sensors and the use of cognitive machine learning techniques to determine at what point in the future maintenance activities will be appropriate.

***Figure 4**: Predictive Maintenance objective*

Adoption of PdM can result in many benefits, such as substantial time and cost savings and higher system reliability. As maintenance is performed when failure is likely to occur, there is high cost savings related to the production hours lost to maintenance, expenses related to parts and supplies and the time for the device to be fixed. Predictive maintenance can eliminate both failures by forecasting it ahead of time and unnecessary and wasteful maintenance: predictive maintenance systems are designed to ensure that maintenance activities do not happen too soon, wasting money on unnecessary work, or too late, after wear and time have caused undue deterioration (***Figure 4***). Organizations that commit to a predictive maintenance program can expect to see significant improvements in asset reliability and a boost in cost efficiency.

2.4.2 Tool condition monitoring

Condition Monitoring (CM) is a major component of Predictive Maintenance. It is defined as the process of monitoring a parameter of condition in machinery (vibration, acoustic emission, temperature, etc.), in order to identify a significant change which is indicative of a developing fault. In CM, Tool Condition Monitoring (TCM) investigates how these parameters affect tool wear.

During continuous machining process, deterioration in tool performance occurs due to the tool wear. Tool wear describes the gradual failure of tools due to regular operation is a major problem encountered in manufacturing industry during machining operations. Tool wear is an important characteristic of machining processes as it adversely affects the tool life, which is of foremost importance in metal cutting owing to its direct impact on the surface quality of the machined surface, and its dimensional accuracy, and consequently, the manufacturing costs [20]. The importance of tool wear monitoring is implied by the possible economic advantages. By replacing worn tools in time, it is possible to avoid the production of waste. Furthermore, tools costs and production time can be reduced noticeably with a precise exploitation of a tool's lifetime [21]. In fact, in production, it is often a disadvantage to use the tool until its breakage occurs, since this failure reduces the work surface quality and leads to a significant loss of dimensional accuracy, which result in defective parts and waste of workpiece material. On the other hand, an excessive preventive replacement of tools involves higher costs and production time: it will not only require additional tools to be purchased, which are generally expensive, but also considerable time to change the tool and reset.

Hence there is a need for a tool condition monitoring (TCM) system, which provides a better health condition of the process and particularly the cutting tool by using continuous monitoring of certain parameters. This TCM system promises higher productivity with reduced maintenance cost and by saving idle time [22]. A tool wear monitoring system is indispensable for better machining productivity, with the guarantee of machining safety by informing of the time due for changing a tool, in order to improve the quality of machining parts, to reduce the machine damage, and cost of machining.

TCM techniques include direct measurement and indirect measurement of tool wears. In direct measuring methods, the wear parameters of the tool are measured by microscope surface profiler, etc. These direct methods provide the advantage of acquiring high accurate dimension changes due to tool wear, but they are not attractive from the technical and economical point of view: they are vulnerable to field conditions, cutting fluid and various disturbances and are usually performed offline, and interrupts normal machining operations because of the contact

between the tool and the measuring device, which severely limits the application of direct measuring methods.

Indirect methods measure suitable parameters which are correlated with tool wear. They are based on significant parameters measured during the machining process that can be correlated to tool condition. Machining process data such as operation parameters and sensor signals (force signals, vibration signals, acoustic emission signals, current/power signals, etc.). are acquired and relevant features are extracted from the data; then these extracted features are used to predict the tool condition by applying artificial intelligent techniques. With effective monitoring system, worn tool can be change in time to avoid production time and scrapped components. The measuring accuracy is lower than that of the direct measuring methods. However, they have the advantages of easy installation and easy to implement online in real time. Generally, an indirect TCM system consists of hardware and software parts to perform signal acquisition, signal pre-processing, features extraction, features selection and decision making (i.e. tool condition estimation). A TCM system framework is presented in ***Figure 5***.

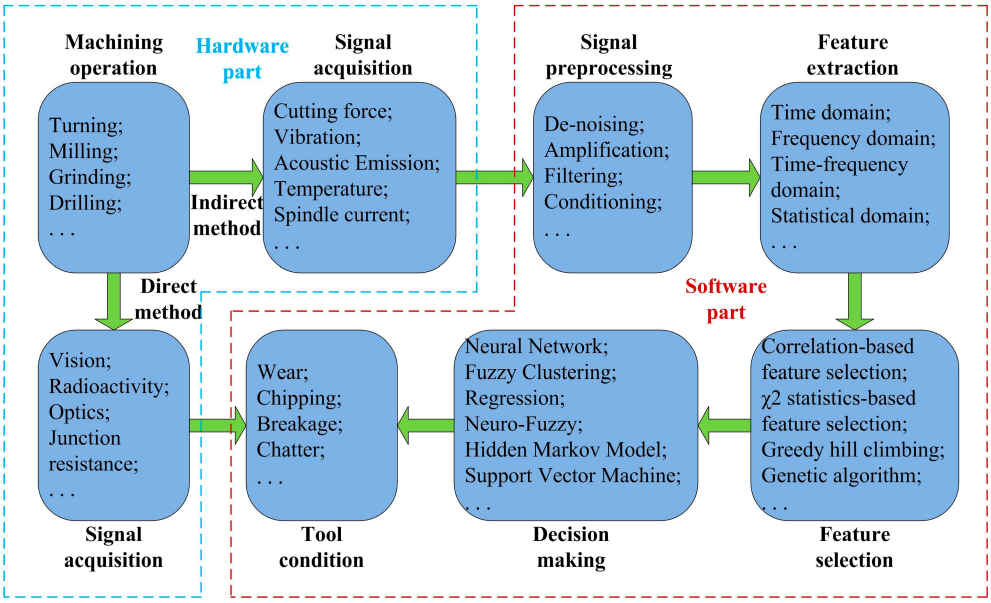


Figure 5*: The framework of a Tool Condition Monitoring (TCM) system*

The supervision of tool wear is the most difficult task in the context of tool condition monitoring for metal-cutting processes as many machining processes are non-linear time-variant systems, which makes them difficult to model. The Thesis will discuss an on-line indirect TCM model to predict the wear of a cutting tool used during milling operations by means of machine learning algorithms.

3. Use case: Milling machine

The use-case chosen for the purpose of the Thesis is the "*Milling Data Set*" collected by the Prognostic Center of Excellence (NASA – PCoE), previously explored by the student C. Xiaorui [23]. This chapter will first define the milling process and the cutting-tool wear measurement approaches and tool life criteria. Consequently, experiment and dataset description will be detailed. The last section will briefly descirbe the platform used to elaborate the solution, Microsoft Azure Machine Learning Studio.

3.1 Milling process

In manufacturing context, cutting operations are essential to obtain finished products with a wide variety of different shapes. Among all cutting processes, milling is the most used form of machining and it consists in using a rotating tool with multiple cutting edges, called cutter, to remove material from the surface of a workpiece. In detail, milling is used to machine the surface of a work part by performing several separate and small cuts with a certain depth. This function is accomplished by slowly advancing, or feeding, the workpiece against the milling cutter rotating at moderately high speed. The direction of the feed motion is perpendicular to the tool's axis of rotation and this feature distinguishes milling from drilling, where the tool advances parallel to its rotation axis. As the milling cutter enters the workpiece, the cutting teeth repeatedly shave off material from the item in the form of small chips, whose extent depends on the material of the workpiece [24].

3.1.1 Types of milling

There are two major classes of milling, as shown in ***Figure 6***: peripheral milling (a) and face milling (b).

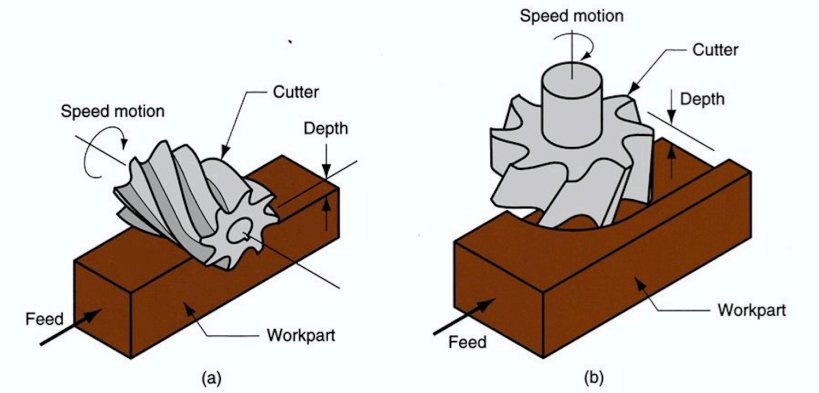


Figure 6*: Types of milling: (a) peripheral and (b) face milling*

In face milling, the axis of the cutter is perpendicular to the surface being machined. This operation is used to make flat surfaces of the workpiece in order to provide a smooth finish. ***Figure 7*** illustrates several forms of face milling: (a) conventional face milling, in which the cutter width extends beyond the work piece on both sides, so the cutter overhangs the work on both sides; (b) partial face milling, where the cutter overhangs the work on only one side; (c) end milling, in which the cutter diameter is less than the work width, so a slot is cut into the

part; (d) profile milling, a form of end milling used to cut a profile on the workpiece; (e) pocket milling, another form of end milling which machines a shallow into a work part; (f) surface contouring, which employs a ball-nose cutter along a curvilinear path to create a three dimensional surface form.

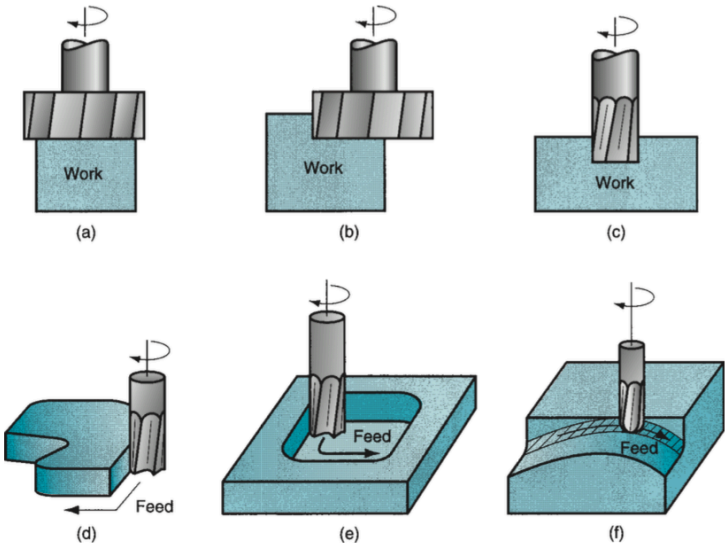


***Figure 7**: Forms of face milling*

In peripheral milling, also called plain milling, the axis of the tool is parallel to the surface being milled. The cutting action occurs primarily along the circumference of the cutter, so that the cross section of the milled surface ends up receiving the shape of the cutter. As for face milling, various forms of peripheral milling exist and they are represented in ***Figure 8***: (a) slab milling, the basic form of peripheral milling in which the diameter of the cutter is greater than the work part width; (b) slot milling, in which the diameter of the cutter is less than the workpiece width, creating a slot in the work when the cutter is very thin; (c) side milling, in which the cutter machines the side of the workpiece; (d) straddle milling, similar to side milling but the cutting action occurs on both sides of the item; (e) form milling, in which the teeth of the milling cutter have a shape which corresponds to the profile of the surface to be produced [25].

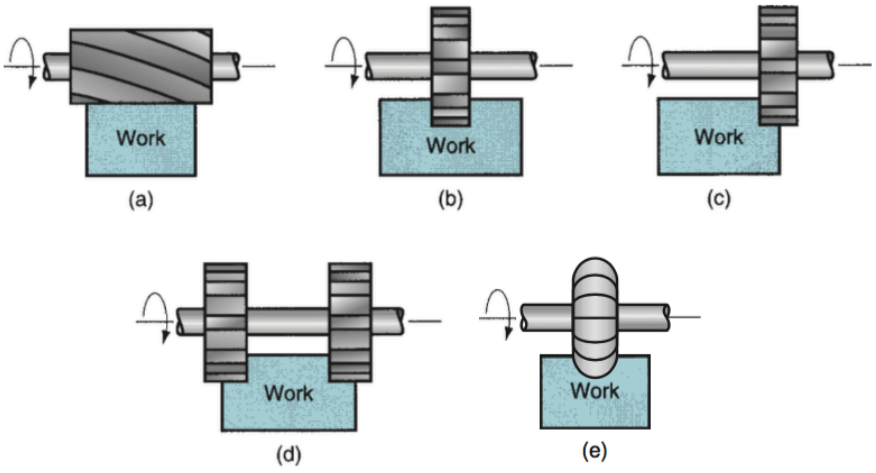


***Figure 8**: Forms of peripheral milling*

3.1.2 Milling machine

Milling process requires a milling machine whose main equipment includes a cutter, spindle and table. The cutter is a cutting tool with several sharp edges, called also teeth, rotating about it axis at high speed. Milling cutters exist in numerous shapes and sizes and they differ depending on type of coatings, rake angle and number of cutting surfaces [26]. The rotary effect is given by a spindle to which the cutter is secured. This fixture is a turning support powered by an electric motor which is used to hold and guide the cutting tool, allowing it to rapidly carve out and shape materials. Eventually, a table attached to the main structure of machine has the function of fastening, positioning, and advancing the work part.

According to the relative position of those three elements, milling machines can be distinguished in (a) horizontal or (b) vertical. ***Figure 9*** shows the major components constituting the two models.

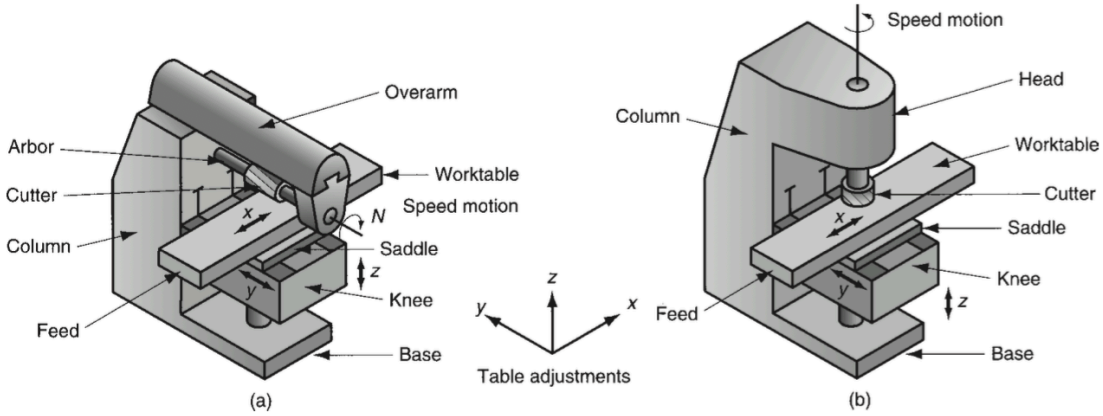


Figure 9*: Milling machines: (a) horizontal and (b) vertical*

As can be seen in the picture, the main difference concerns the orientation of the support holding the cutter. A vertical mill involves a vertically oriented spindle which is perpendicular to the table, and therefore to the workpiece. The cutter is directly secured to the support and then rotate within the axis. In a horizontal milling machine, the cutting tool is mounted on a horizontal arbor across the table. This shaft, maintained by the spindle and an additional support, is a shaped bar that varies in size, length and ending and rotate around its axis. A broader classification of milling machines is briefly discussed in ***Appendix 1***.

Choosing which kind of mill to adopt depends especially on the shape, size and material of the workpiece, as well as the number of planes on which a piece needs to be worked. While the vertical mill is most suited for smaller items and die sinking operations, the horizontal one is preferred for heavier pieces and those that require to be shaped on several sides, due to the arbor support and a larger cross-sectional area of the cutter, which enable rapid material removal rates [27].

3.1.3 Applications

Due to the variety of forms possible and its high production rates, milling is one of the most versatile and widely used machining operations. It is typically used to produce components of non-symmetrical shapes or which have high complexity such as holes, slots, pockets, grooves and even three-dimensional profiles. The characteristics of the process make it ideal for the production of batches of limited quantities, prototypes or custom designed elements. An additional application of milling is the manufacturing of tools for other machineries, for instance three-dimensional moulds. By virtue of the high accuracy and surface finish that can be achieved, milling can be used as secondary process on semi-finished products obtained by means of a different operation in order to add or refine precision features. Eventually, milling can also be used alternatively to drilling, boring, countersinking and threading to create blind holes, through holes, threads, cavities and a wide range of surfaces [28].

3.1.4 Cutting parameters

Milling requires the definition of several cutting parameters, the combination of which influences the process in terms of speed of doing the job, the quality and accuracy of the finish, the life of the tool and the cost of production. The main parameters involve the following: cutting speed (vc), spindle sped (N), feed rate (vf), depth of cut (ap), tool material and workpiece material. ***Figure 10*** represents some of them.

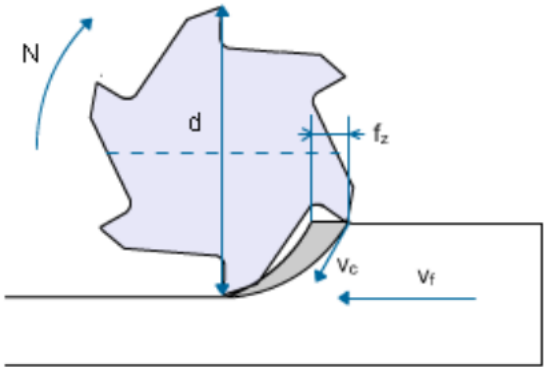


Figure 10*: Cutting parameters: (N) spindle speed, (d) cutter diameter, (vc) cutting speed, (fz) feed per tooth, (vf) feed rate*

3.1.4.1 Cutting speed

Relative motion is required between the cutter and work material to perform a milling operation. The primary motion is given by cutting speed, which has to do with the speed of rotation of the tool. In detail, cutting speed is the relative velocity at which the cutter passes through the workpiece and removes material. It is measured at the outside edge of the cutting tool as it is rotating and it is normally expressed as distance units along the work part surface per unit of time, like Surface Feet per Minute (SFM), meters per minute (m/min). An ideal cutting speed

is always suggested by the tool manufacturers depending on the combination of cutter material, workpiece material and type of milling.

3.1.4.2 Spindle speed

The spindle speed is defined as the rotational frequency of the spindle of the milling machine [29]. It is normally measured in revolutions per minute (RPM), describing the number of turns completed in one minute around the spindle fixed axis. It can be derived from the cutting speed, determined at the outside diameter of the milling cutter, by dividing it for the tool circumference. Therefore, the formula for calculating the spindle speed is the following:

$$RPM = \frac{CS}{\pi D}$$

Where
RPM = Spindle speed (in rev/min).
CS = Cutting speed
D = Tool diameter

3.1.4.3 Feed rate

The secondary motion involved in the milling operation is the feed rate, also called simply feed. It refers to the velocity at which the cutting tool is advanced through the workpiece to remove material. The desirable feed rate depends on the amount of material removed by each cutting edge, expressed as feed per tooth or chip load. By taking into consideration the number of teeth on the cutter, the feed can be measured in length of material per full revolution of the cutter (mm/rev), known as feed per revolution. The feed per revolution states how much the workpiece advances in one revolution of the cutting tool. To actually calculate the time necessary for the milling operation, it is feed per minute (in mm/min) that is useful. It is obtained by multiplying the feed per revolution, given by the product of chip load and number of teeth, for the spindle speed, as in the formula below.

$$FR = CL \times T \times RMP$$

Where:
FR = Feed rate (mm/min)
CL = Chip load (mm/tooth)
T = Number of teeth on the cutter
RMP = Spindle speed (rev/min)

3.1.4.4 Depth of cut

The depth of cut is the penetration of the cutting tool along its axis in the workpiece as it makes a cut. It indicates how much the tool digs into the component to remove material in one pass of the work under the cutter. It is the perpendicular distance measured between the original and

final surface of the workpiece, usually expressed in mm. A large depth of cut requires a low feed rate, otherwise it would result in a high load on the tool and a reduction of the tool life.

Cutting speed, feed rate, and depth of cut are called cutting conditions and they determine the material removal rate (MRR), which is the volume of workpiece material that can be removed per time unit (***Appendix 2***).

3.1.4.5 Tool material

All cutting tools that are used in milling can be found in a variety of materials, which determines the cutter's properties and the workpiece materials for which it is best suited. In order to produce good quality parts, a cutting tool must present three important characteristics:

- **Hardness** = The ability of the material to maintain its strength at elevated temperatures. Cutting tool materials must be harder than the material of the workpiece, even in high-temperature environment during the operation.
- **Toughness** = The capacity of the material to absorb energy without failing, usually deriving from a combination of strength and ductility. Toughness is necessary in order for the cutting tool to not chip or fracture, especially during interrupted cutting operations.
- **Wear resistance** = The attainment of acceptable tool life before cutting tool need to be replaced. Hardness is the single most important property needed to resist abrasive wear. Other characteristics affecting wear resistance include surface finish on the tool, chemistry of tool and work materials, and whether a cutting fluid is used. Tool wear affects the entire milling process and is further discuss in ***Section 3.2***.

Cutter materials achieve these properties in varying degrees. The most common cutting tool materials used nowadays include high-speed steels (HSS), cast-cobalt alloys, carbides, ceramics, cubic boron nitride (CBN), diamond.

Being aware of differences among tool materials is very important, since it influences the milling process. The choice of cutter's material is based on a number of factors, including the type of milling, the material of the workpiece and production cost [30].

3.1.4.6 Workpiece Material

In milling, the raw form of the work part is a piece of stock from which the required items are cut. This stock is available in a variety of shapes, such as flat sheets, solid bars and shaped beams, and materials, including most metals and plastics. Common materials that are used in milling involve steel, cast iron, aluminium, nickel, zinc, magnesium, titanium and thermoset plastics. Properties of the work material have a significant influence on the success of milling operation and the selection of the workpiece material must take into consideration several factors as cost, strength, resistance to wear and especially its machinability. The machinability of a material refers to the ease with which it can be machined, in this case milled. There is no a direct definition, but in a broader sense it has to do with the degree of surface finish, duration of tool life, cutting forces and temperature, power consumption and collection of chips. Machinability is difficult to quantify: it is often evaluated on a case-by-case basis and tests are

specific to the needs of a certain manufacturing process [31].

3.1.5 Cutting forces and emissions

The interaction of milling cutter and workpiece is a complex process and generates different physical effects. During the engagement of the tool in the work part, energy is released which results in cutting forces, vibration and acoustic emission.

3.1.5.1 Cutting forces

Cutting forces appear in shear zones where plastic deformation takes place (***Figure 11***). The predominant cutting action in milling involves deformation of the workpiece to form a chip in the primary shear zone: as the cutter advances, chip of material is removed and new surface is exposed. The force component applied by the workpiece on the chip is known as share force. Another shearing action occurs in the chip after it has been formed, in the secondary shear zone. It results from friction between the chip and the cutting tool as the chip slides along the rake face of the tool. This second force is normally referred as frictional force.

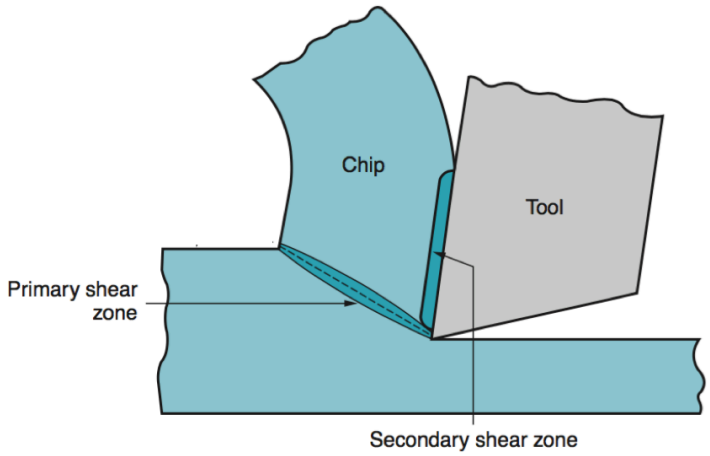


Figure 11: *Shear zones in milling*

3.1.5.2 Vibration

Vibration emission is a low frequency oscillation due to the acceleration of the object because of the dynamic changes of cutting forces resulting from periodical changes in tool geometry, chip formation, and built up edges.

3.1.5.3 Acoustic emission

Acoustic emission is a high frequency oscillation which occurs spontaneously within metals when they are deformed or fractured. It is caused by the release of strain energy as the micro

structure of the material is rearranged. Acoustic emission is generated in the shear zones, the primary as well as the secondary along the chip-tool interface through bulk deformation and sliding, respectively, and, lastly, at the tool flank-workpiece interface due to friction [32].

The analysis of cutting parameters, forces and emissions has a major impact in the evaluation of tool life during milling operations. The next paragraph focuses on the definition and methods for tool wear measurement, explaining its importance for the success of the entire milling process.

3.2 Tool wear monitoring

As milling proceeds, the various deterioration mechanisms, such as abrasion and plastic deformation, result in increasing levels of wear on the cutting tool. Generally, wear of cutters in milling processes depends on tool material and geometry, workpiece materials, cutting parameters (cutting speed, feed rate and depth of cut), cutting fluids and machine-tool characteristics. Speed of cutting, more than other parameters, influence the rate of wear; depth of cut and feed rate also affect the tool life. Monitoring the condition of the cutting tool in the machining process is very important since tool wear will affect the part size, quality and an unexpected tool failure may damage the tool, work-piece and sometimes the machine tool itself.

3.2.1 Tool wear measurement

Gradual wear occurs at two principal locations on a cutting tool, illustrated in ***Figure 12***. There are two basics zones of wear in cutting tools: flank wear and crater wear: apart from the intuitive rounding of the cutting edge, crater wear on the rake face due to the abrasion of the sliding of the chip on the rake face and flank wear due to friction of the tool on the workpiece occur.

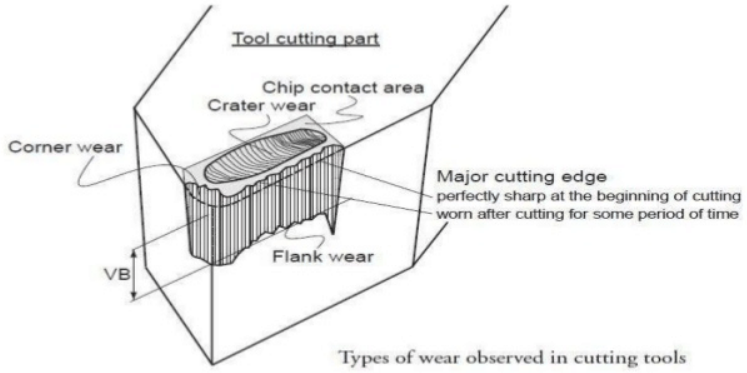


***Figure 12**: Types of wear observed in cutting tools*

Crater wear, ***Figure 13.a***, consists of a cavity in the rake face of the tool that forms and grows from the action of the chip sliding against the surface. High stresses and temperatures characterize the tool–chip contact interface, contributing to the wearing action. The crater can be measured either by its depth or its area. This is somewhat normal for tool wear, and does not seriously degrade the use of a tool until it becomes serious enough to cause a cutting edge failure.

Flank wear, ***Figure 13.b***, occurs on the flank, or relief face, of the tool. It results from rubbing between the newly generated work surface and the flank face adjacent to the cutting edge. Flank wear is measured by the width of the wear band, VB. This wear band is sometimes called the flank wear land.

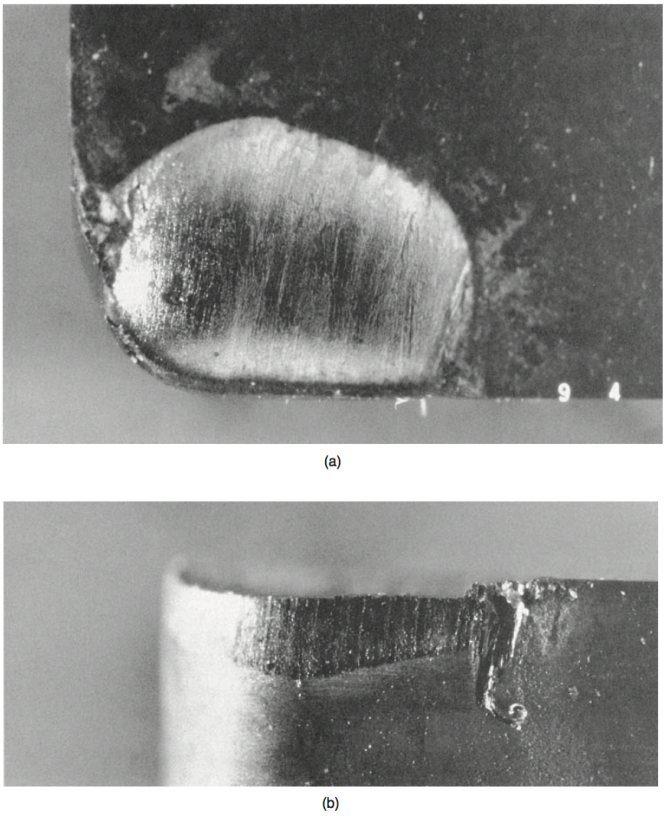


Figure 13*: (a) Crater wear and (b) flank wear on a cemented carbide tool, as seen through a toolmaker's microscope*

3.2.2 Tool life criteria

In milling, tool life is defined as the length of cutting time that the tool can be used. Operating the tool until final catastrophic failure is one way of defining tool life. However, in production, it is often a disadvantage to use the tool until this failure occurs because of difficulties in re-sharpening the tool and problems with work surface quality. As an alternative, a level of tool wear can be selected as a criterion of tool life, and the tool is replaced when wear reaches that level.

A conventional tool life criterion is a certain width of the flank wear (VB), because of its influence on workpiece surface roughness and accuracy. Procedures and conditions for tool-life testing in milling are recommended by ISO standards ISO 8688-1:1989 for face milling [33] and ISO 8688-2:1989 for end milling [34]. The level depends on several parameters, such as the type of milling and the material of the cutter. For instance, the standard ISO 8688-1:1989 defines a maximal flank wear (VB_{max}) equal to ***0.6 mm*** as effective tool life for cemented carbides, high-speed steels (HSS) and ceramics tools applied in face milling operations.

Although flank wear is of easy interpretation, this criterion is not very practical in a factory environment because of the difficulties and time required to measure flank wear. Hence, mathematical functions for gradual wear try to compute tool life, for example Taylor's tool life formula:

$$T = C_v {v_c}^k$$

Where:

- T = tool life
- v_c = cutting speed
- k = Taylor exponent
- C_V = constant related to 1 min tool life
- k and C_V are characteristics of the tool and the workpiece material combination.

The limitations of this approach are that different materials and wear due to the influence of feed cannot be accounted for. Apart from that the problem with tool wear in general is that it may not be very predictive; slight variations in a seemingly same setting vary the tool life considerably. These variations are manifold. One is that the same material can have different strengths which is often the case with cast iron. But this variation can also be local in form of inclusions. These can initiate increased wear when they remove large chunks of an insert. Even if the abrasion is of a smaller degree, the geometry of the insert changes, i.e., the insert becomes less sharp and as a result the tool has to plow through the material with greater force, more friction and, as a result, greater tool wear. Other variations are the amount of coolant used to control the temperature of the cutting process. Two thirds of the heat generated during the cutting process is removed via chip and tool. Changing this temperature may have an effect on chemical processes by allowing alloying elements to diffuse into the workpiece and thus weaken the structure of the insert. Furthermore, the depth of cut may not be steady, in particular when machining a new rough surface for the first time. All of these influences may be small, but they can add up over the period of machining to cause considerable uncertainty in predicting a priori the tool life.

The use of additional parameters s^i in Taylor's formula to result in

$$T = C_v {s^i}{v_c}^k$$

try to incorporate such influences, however only with modest success. These shortcomings again motivate the use of non-mathematical techniques.

3.3 Experiment description

The Prognostic Center of Excellence (PCoE) at NASA's Ames Research Center provides a collection of prognostic datasets that has donated by various universities, agencies and companies. The data repository focuses exclusively on datasets to be used for prognostic algorithms, i.e. to predict the time at which a system or a component will no longer perform its intended function. The database chosen for the purpose of the Thesis is called "Milling data set" and it was provided by A. Agogino and K. Goebel (2007) working at BEST lab in UC Berkeley[1]. The data in this set represents experiments from runs on a milling machine under various operating conditions, such as different feeds, depth of cut and workpiece material. In particular, tool wear was investigated in a regular cut as well as entry cut and exit cut and the flank wear of milling insert was recorded.

3.3.1 Milling machine and cutting parameters

The experiment has been carried out by employing the Matsuura machining center MC-510V (***Figure 14***), a CNC vertical milling machine equipped with a face milling cutter with six inserts.

Figure 14*: Matsuura machining center MC-510V*

[1] A. Agogino and K. Goebel (2007). BEST lab, UC Berkeley. "Milling Data Set ", NASA Ames Prognostics Data Repository (http://ti.arc.nasa.gov/project/prognostic-data-repository), NASA Ames Research Center, Moffett Field, CA

The cutting parameters for the different cases were guided by industrial applicability and recommended manufacturer's settings. Therefore, the cutting speed was set to 200 m/min which is equivalent to 826 rev/min. Two different depths of cut were chosen, 1.5 mm and 0.75 mm. Also, two feeds were taken, 0.5 mm/rev and 0.25 mm/rev which translate into 413 mm/min and 206.5 mm/min, respectively. Two types of material, cast iron and stainless steel J45 were used for the workpieces, whose size was 483mm x 178mm x 51mm. The milling inserts were of type KC710, a carbide with a PVD TiN coating over a general purpose alloyed substrate. These choices equal 8 different settings. All experiments were done a second time with the same parameters with a second set of inserts, for a total of 16 different cases.

3.3.2 Tool wear

In the experiment, the flank wear VB was selected as a generally accepted parameter for evaluating tool wear. It is measured as the distance from the cutting edge to the end of the abrasive wear on the flank face of the tool (***Figure 15***). For the reasons explained in ***Section 3.2.2***, the use of this technique to evaluate tool life is often preferred to mathematical functions such as the Taylor's formula. The VB was observed after almost each run: the milling insert was taken out of the tool and the wear was measured with the help of a microscope.

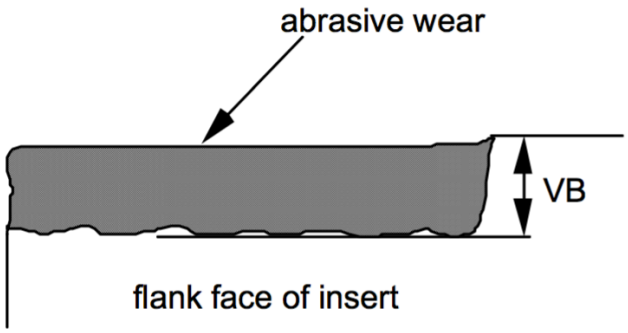


Figure 15*: Tool wear VB as it is seen on the insert*

3.3.3 Experiment setup

Six different sensors have been adopted during the experiment to capture signals related to acoustic emissions, vibrations and cutting forces, which all represent relevant phenomenon for monitoring machining processes and especially assessing milling tool condition.

The setup of the experiment is as depicted in ***Figure 16*** below. The basic setup encompasses the spindle and the table of the Matsuura machining center MC- 510V. An acoustic emission sensor and a vibration sensor are each mounted to the table and the spindle of the machining center. The signals from all sensors are amplified and filtered, then fed through two RMS devices[2] before they enter the computer for data acquisition.

[2] Removable Mass Storage device is a type of device and media that is high-capacity, random access, and interchangeable between systems and partitions. Similar to optical storage in regard to behaviour and access interfaces.

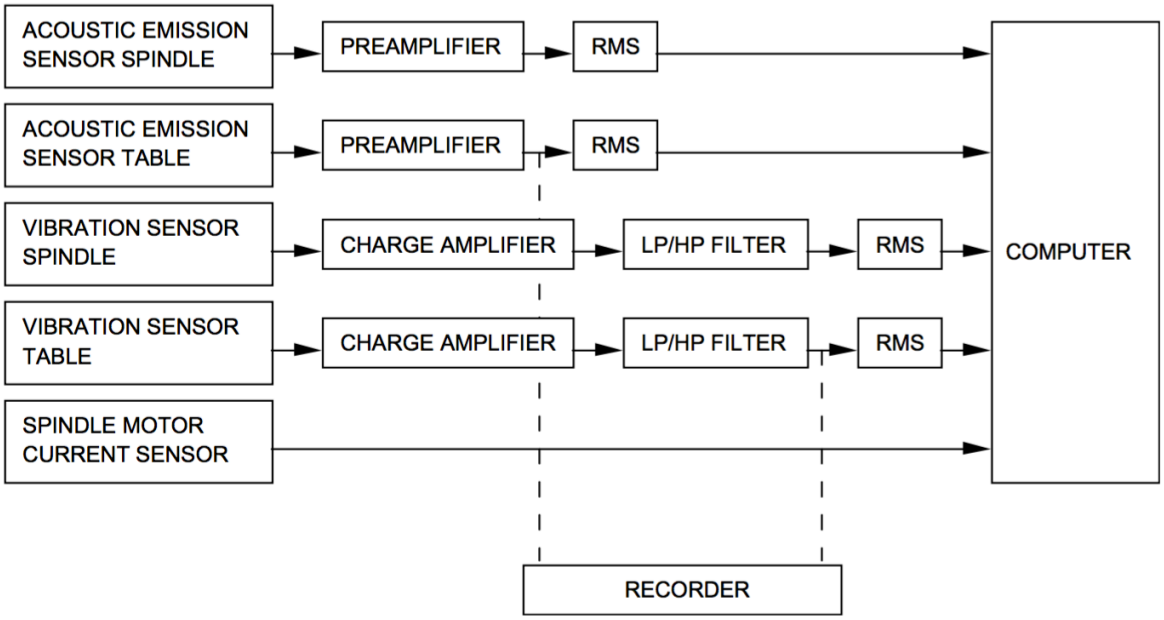


***Figure 16**: Experimental setup*

The signal from a spindle motor current sensor is fed into the computer without further processing. A more detailed experiment setup description can be found in ***Appendix 3***.

3.3.4 Data acquisition and processing

As described in the previous section, the data were sent through a high speed data acquisition board with maximal sampling rate of 100 KHz. The sampled output of the data was used for the signal processing software. LabVIEW® (National Instruments, USA) was used for this task. This software is a general purpose programming development system which uses a graphical language (G). With G, programs are created in block diagram form. The chosen layout allowed for data acquisition, storage, presentation, and processing. Data were stored to allow for real time simulation and also later analysis.

Several sensor signals underwent pre-processing. In most cases, the signal was amplified to be able to meet threshold requirements of equipment. In particular, the signals from the acoustic emission sensors and from the vibration sensors were amplified to be in the range of ±5V for maximum load, considering the maximum allowable range of the equipment. The signals were filtered by a high pass filter, the vibration sensor signals were additionally filtered with a low pass filter. Corner frequencies were chosen according to the noise that could be observed on an oscilloscope. Periodical noise of 180Hz was observed on the oscilloscope for the vibration signal corresponding to the third harmonic of the main power supply. Therefore, the chosen corner frequency for the low pass filter was 400Hz. For the high pass filter, 1kHz was chosen. Above 8KHz, the range of the acoustic emission sensor ends. That is, readings above that frequency cannot be attributed to any occurrence in the machining process. Since it clutters the signal unnecessarily, it was filtered out. Acoustic emission and vibration signals were fed through an RMS device. Its use smooths the signal and makes it more accessible to signal processing. The RMS is proportional to the energy contents of the signal, according to the formula:

$$RMS = \sqrt{\frac{1}{\Delta T}\int_0^{\Delta T} f^2(t)dt}$$

where:

- ΔT = time constant;
- f(t) = signal function.

The sampling rate has to be smaller than the time constant to ensure proper data sampling. The chosen parameters were:

- ΔT = 8.00 ms;
- sampling rate = 250 Hz.

Apart from the pre-processed data, raw data of the acoustic emission of the table and the vibration of the table were recorded on a tape recorder to allow for future comparison and evaluation. It might be of interest for feature extraction to have data that did not undergo previous selection; in the same spirit, it is worthwhile to be able to use data that are not "corrupted" yet for neural network techniques or data clustering algorithms. Typical sensor readings from spindle motor current AC and DC portion, acoustic emission at the table, vibration at the table, acoustic emission at the spindle, and vibration at the spindle are displayed in ***Appendix 4*.**

3.4 Dataset structure

The data set includes information about 16 cases, enumerated in ***Table 2***, with varying number of runs.

Case	Depth of Cut	Feed	Material
1	1.5	0.5	1 - cast iron
2	0.75	0.5	1 - cast iron
3	0.75	0.25	1 - cast iron
4	1.5	0.25	1 - cast iron
5	1.5	0.5	2 - steel
6	1.5	0.25	2 - steel
7	0.75	0.25	2 - steel
8	0.75	0.5	2 - steel
9	1.5	0.5	1 - cast iron
10	1.5	0.25	1 - cast iron
11	0.75	0.25	1 - cast iron
12	0.75	0.5	1 - cast iron
13	0.75	0.25	2 - steel
14	0.75	0.5	2 - steel
15	1.5	0.25	2 - steel
16	1.5	0.5	2 - steel

***Table 2**: Experimental conditions*

The cases are the result of different combination of experimental parameters for depth of cut (0.75 or 1.5), feed rate (0.25 or 0.5) and workpiece material (cast iron or steel). The number of runs for each case was dependent on the degree of flank wear (VB) that was measured between runs at irregular intervals up to a wear limit (and sometimes beyond). Flank wear was not always measured and at times when no measurements were taken, no entry was made.

The acquired data is organized in a 1x167 Matlab struct with fields as shown in ***Table 3*** below:

Field name	Description
case	Case number (1-16)
run	Counter for experimental runs in each case
VB	Flank wear, measured after runs (measurements for VB were not taken after each run)
time	Duration of experiment (restarts for each case)
DOC	Depth of cut (does not vary for each case)
feed	Feed (does not vary for each case)
material	Material (does not vary for each case)
smcAC	AC spindle motor current
smcDC	DC spindle motor current
vib_table	Table vibration
vib_spindle	Spindle vibration
AE_table	Acoustic emission at table
AE_spindle	Acoustic emission at spindle

Table 3*: Struct field names and description*

The main limitation of this dataset consists in the number of instances used: at best 167 instances, also including missing VB information that is not useful for developing the predictive model. As is well known, machine learning applications usually require thousands of instances in order to have consistent trained models and proper evaluations. However, due to the scarce availability of data and for the general purpose of the Thesis, this type of procedure should be appropriate.

Table 4 illustrates how the data looks in Matlab. The cells under the columns “smcAC”, “smcDC”, “vib_table”, “vib_spindle”, “AE_table”, “AE_spindle” contain the data captured by the relative sensor (each of them containing 9,000 data samples).

Fields	case	run	VB	time	DOC	feed	material	smcAC	smcDC	vib_table	vib_spindle	AE_table	AE_spindle
1	1	1	0	2	1.5000	0.5000	1	9000x1 d...	9000x1 d...	9000x1 do...	9000x1 double	9000x1 do...	9000x1 double
2	1	2	NaN	4	1.5000	0.5000	1	9000x1 d...	9000x1 d...	9000x1 do...	9000x1 double	9000x1 do...	9000x1 double
3	1	3	NaN	6	1.5000	0.5000	1	9000x1 d...	9000x1 d...	9000x1 do...	9000x1 double	9000x1 do...	9000x1 double
4	1	4	0.1100	7	1.5000	0.5000	1	9000x1 d...	9000x1 d...	9000x1 do...	9000x1 double	9000x1 do...	9000x1 double
5	1	5	NaN	11	1.5000	0.5000	1	9000x1 d...	9000x1 d...	9000x1 do...	9000x1 double	9000x1 do...	9000x1 double
6	1	6	0.2000	15	1.5000	0.5000	1	9000x1 d...	9000x1 d...	9000x1 do...	9000x1 double	9000x1 do...	9000x1 double
7	1	7	0.2400	19	1.5000	0.5000	1	9000x1 d...	9000x1 d...	9000x1 do...	9000x1 double	9000x1 do...	9000x1 double
8	1	8	0.2900	22	1.5000	0.5000	1	9000x1 d...	9000x1 d...	9000x1 do...	9000x1 double	9000x1 do...	9000x1 double
9	1	9	0.2800	26	1.5000	0.5000	1	9000x1 d...	9000x1 d...	9000x1 do...	9000x1 double	9000x1 do...	9000x1 double
10	1	10	0.2900	29	1.5000	0.5000	1	9000x1 d...	9000x1 d...	9000x1 do...	9000x1 double	9000x1 do...	9000x1 double
11	1	11	0.3800	32	1.5000	0.5000	1	9000x1 d...	9000x1 d...	9000x1 do...	9000x1 double	9000x1 do...	9000x1 double
12	1	12	0.4000	35	1.5000	0.5000	1	9000x1 d...	9000x1 d...	9000x1 do...	9000x1 double	9000x1 do...	9000x1 double
13	1	13	0.4300	38	1.5000	0.5000	1	9000x1 d...	9000x1 d...	9000x1 do...	9000x1 double	9000x1 do...	9000x1 double
14	1	14	0.4500	41	1.5000	0.5000	1	9000x1 d...	9000x1 d...	9000x1 do...	9000x1 double	9000x1 do...	9000x1 double
15	1	15	0.5000	44	1.5000	0.5000	1	9000x1 d...	9000x1 d...	9000x1 do...	9000x1 double	9000x1 do...	9000x1 double
16	1	16	NaN	46	1.5000	0.5000	1	9000x1 d...	9000x1 d...	9000x1 do...	9000x1 double	9000x1 do...	9000x1 double
17	1	17	0.4400	48	1.5000	0.5000	1	9000x1 d...	9000x1 d...	9000x1 do...	9000x1 double	9000x1 do...	9000x1 double
18	2	1	0.0800	3	0.7500	0.5000	1	9000x1 d...	9000x1 d...	9000x1 do...	9000x1 double	9000x1 do...	9000x1 double
19	2	2	0.1400	9	0.7500	0.5000	1	9000x1 d...	9000x1 d...	9000x1 do...	9000x1 double	9000x1 do...	9000x1 double
20	2	3	0.1400	12	0.7500	0.5000	1	9000x1 d...	9000x1 d...	9000x1 do...	9000x1 double	9000x1 do...	9000x1 double
21	2	4	0.1400	15	0.7500	0.5000	1	9000x1 d...	9000x1 d...	9000x1 do...	9000x1 double	9000x1 do...	9000x1 double
22	2	5	0.1500	22	0.7500	0.5000	1	9000x1 d...	9000x1 d...	9000x1 do...	9000x1 double	9000x1 do...	9000x1 double
23	2	6	NaN	24	0.7500	0.5000	1	9000x1 d...	9000x1 d...	9000x1 do...	9000x1 double	9000x1 do...	9000x1 double
24	2	7	0.1800	27	0.7500	0.5000	1	9000x1 d...	9000x1 d...	9000x1 do...	9000x1 double	9000x1 do...	9000x1 double
25	2	8	0.2200	33	0.7500	0.5000	1	9000x1 d...	9000x1 d...	9000x1 do...	9000x1 double	9000x1 do...	9000x1 double

***Table 4**: Milling Dataset structure in Matlab*

3.5 Microsoft Azure: Machine Learning Studio

Data collected during the experiment will be processed and analysed by using Microsoft Azure platform, in particular its service Machine Learning Studio. Machine learning platforms are among enterprise technology's most competitive realms, with most major vendors, including Amazon, Google, Microsoft, IBM and others, racing to sign customers up for platform services that cover the spectrum of machine learning activities, including data collection, data preparation, model building, training and application deployment. Among the vast choice of platforms available, Microsoft Azure offers promotions to student, i.e. offering their service for free to various universities (among which, Politecnico di Torino).

Microsoft Azure is a platform of interoperable cloud computing services, including open-source, standards-based technologies and proprietary solutions from Microsoft and other companies. Microsoft Azure is a cloud computing service created by Microsoft for building, testing, deploying, and managing applications and services through a global network of Microsoft-managed data centers. The rise of cloud computing provides businesses the ability to quickly provision computing resources without the costly and laborious task of building data centers, and without the costs of running servers with unutilized capacity due to variable workloads.

Among the many services offering by Azure Suite, the Thesis involves the use of Machine Learning Studio (MLS) to apply machine learning techniques to our use case. Microsoft Azure Machine Learning Studio [35] is a cloud predictive analytical service that makes it possible to quickly create and deploy predictive models as analytics solutions. It implements drag-and-drop tool that can be used to build, test, and deploy predictive analytics solutions on a dataset. Machine Learning Studio publishes models as web services that can easily be consumed by custom apps or BI tools such as Excel.

Developing a predictive analysis model, typically data from one or more sources are used, means transforming and analysing the data through various data manipulation and statistical functions, and generating a set of results. The development of a model like this is an iterative process. As the various functions and their parameters are being modified, the results converge

until satisfaction of the trained model. Azure Machine Learning Studio (Azure MLS) gives an interactive, visual workspace to easily build, test, and iterate on a predictive analysis model. The drag-and-drop method allows datasets and analysis modules to be set onto an interactive canvas, connecting them together to form an experiment, which is run in Machine Learning Studio. To iterate on the model design means editing the experiment, save a copy if desired, and run it again. When ready, the training experiment ca be converted into a predictive experiment, and then publish it as a web service so that the model can be accessed by others (***Figure 17***).

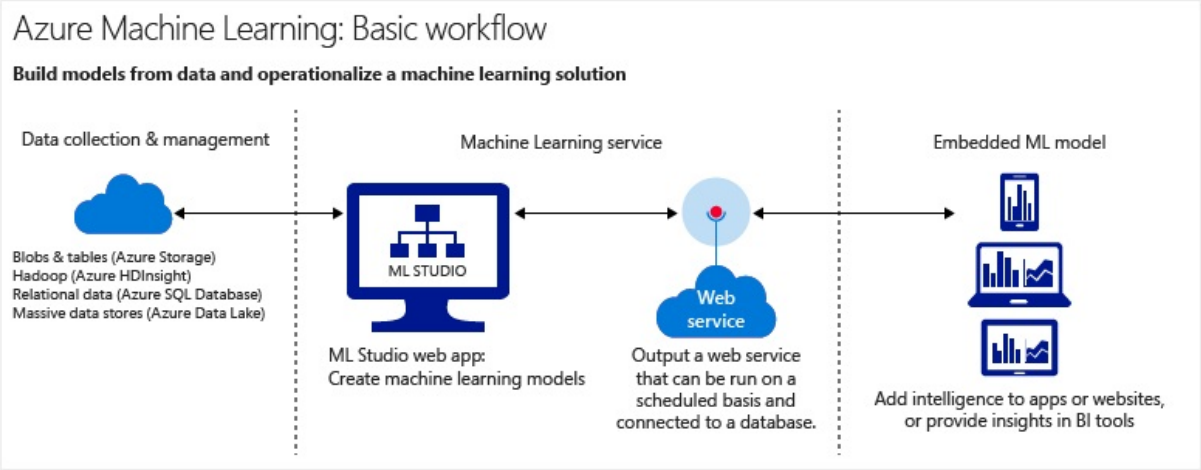


***Figure 17**: Azure Machine Learning: Basic workflow*

There is no programming required (apart from more customized modules that can be implemented on a R/Python script.), just visually connecting datasets and modules to construct the predictive analysis model.

3.5.1 Implementation of ML models

Azure MLS has available a large number of machine learning algorithms, along with modules that help with input data, output, preparation, and visualization. By using these components, one can develop a predictive analytics experiment, iterate on it, and use it to train your model.

The steps required to build a model are in common from model to another, but variations occur within step. The followings represent a general workflow in order to construct a ML model:

- Data import – datasets can be imported from local (saved in your Azure ML account's cloud) or through an URL (of suitable repository). The data format accepted are usually tables or raw/text data. As mentioned earlier, the data available is a Matlab structure, thus, unfortunately, not suitable for this service. However, with a little elaboration, it can be imported easily.
- Features engineering – the outcome of the model is heavily affected by the quality and quantity of the features; thus careful attention should be given to this step.
- Data preparation – it is a pre-training step, that is needed to make the input data suitable for training the machine learning algorithm. Preparation consists of cleaning missing values and/or treating non-adequate data, as well as selecting labels (in supervised tasks), normalizing data and applying transformations on the data.

- Split data – this phase allows the dataset to be split into training set and testing set according to specific splitting rules.
- Train model – specific modules are used to train ML algorithms ranging from simple training modules to cross-validation methods (supervised learning), as well as cluster training (unsupervised learning).
- Test model – phase dedicated to the evaluation of the model. If a supervised task is used, i.e. classification or regression, testing the model consists of using the testing set (from splitting the pre-training dataset) to evaluate the performance of the trained model. If an unsupervised model is chosen, i.e. clustering, then the evaluation of the model consists on visualizing the clusters obtained. It is best-practice to train and test different algorithms to evaluate which one performs the best in the specific problem.

Figure 18 is an example of ML model implemented in Azure MLS. It shows how the drag-and-drop mechanism and the connection between modules look like in the MLS canvas.

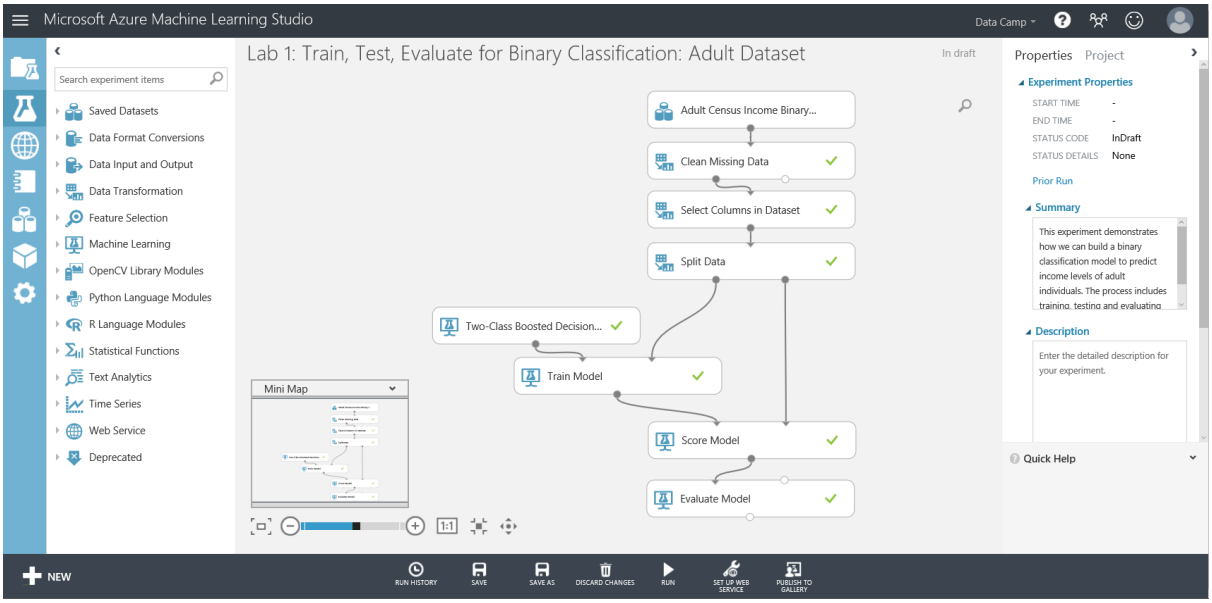


***Figure 18**: Sample model implementation in Azure MLS canvas*

Each module has its own input(s) and output(s). Inputs and outputs can have various nature. For instance, Data transformation modules, such as Clean missing data or Select Columns in Dataset data, take dataset (or columns of dataset) either as input and output. Training model modules take as input an un-trained model and a training dataset and return the trained model.

Connection between modules is represented by an arrow, starting from an output node of a module and ending to an input node of the next. Multiple arrows can take off from the same output node, i.e. the same output, for example a dataset, can be used by other modules. However, multiple arrow cannot end in the same input node of a module. Arrows can be interpreted as data flowing from one module to another.

4. Data Preparation

Figure 19 show the configuration of the data preparation step in the Classification task (a) and Regression task (b) of the model in the Azure MLS canvas.

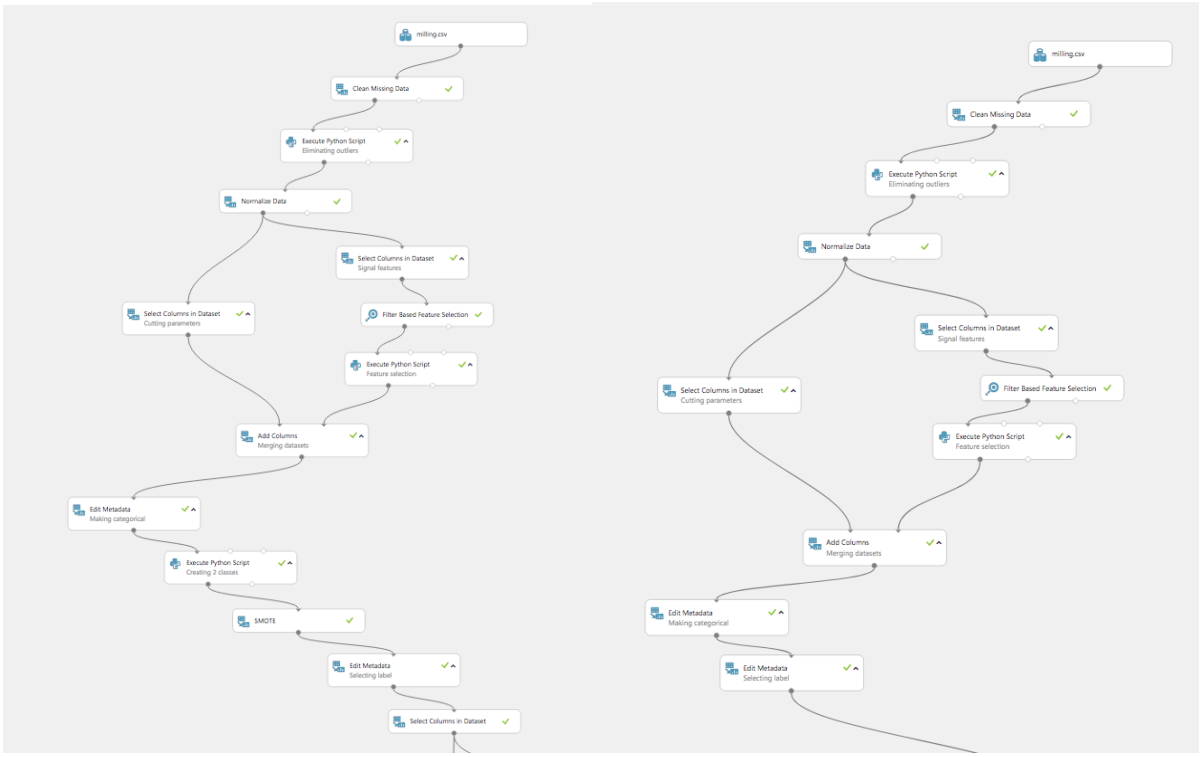


***Figure 19**: Data preparation for (a) classification task and (b) regression task in Azure MLS*

4.1 Feature extraction

As first action to develop the intended TCM model, available data must be prepared for the IoT platform. Microsoft Azure Machine Learning Studio (MLS) doesn't accept ".mat" structures format as dataset input, because it treats datasets as tables, which means that each entry in the table is considered like an "instance", or example that the machine can use to learn. In the Matlab structure, instead, the information related to signals is enclosed in a single cell (9000 x 1 vector), as seen in **Table 4**. Therefore, a Matlab signal processing task is necessary in order to make the dataset suitable for MLS. The way of proceeding chosen for the Thesis is to create relevant features so as to synthetize the information of the signals in single instances. This methodology is known as feature extraction and it has the advantage that no bias is introduced in the data, since only features extracted from original data are used.

Feature extraction is the process of transforming raw data into meaningful features more suitable for a machine data mining task, which act as inputs for machine learning algorithms and help in improving the overall predictive model performance. Generally, feature extraction starts from an initial set of measured data and builds derived values (features) intended to be explanatory and essential, simplifying the subsequent learning and modelling phases. A feature is simply an attribute on which the prediction is done. Considering a generic two-dimensional

dataset, a feature corresponds to an individual, measurable attribute, depicted by a column, which will have a specific value for any observation, represented by a row [36]. Thus, creating new features means substituting or adding new columns to the dataset containing relevant information for the purpose of the problem. The quality and quantity of features are key determinants which highly influence/affect/impact the result of the prediction [37].

As regards signal processing, data captured by signal sensors (9000 data samples for each original signal) are used to extract features in time and frequency domains. The different extraction approaches have different abilities in extracting the meaningful information about tool wear. The next sections show/discusses how feature extraction is applied to the milling dataset in Matlab.

4.1.1 Time domain

Feature extraction in time domain allows to evaluate the magnitude of the signal. Typical time domain features found in literature (Zhang et al. [38], Caesarendra et al [39]) which have been proved to efficiently identify differences between signals and related tool wear are summarized in ***Table 5***.

Index	Feature	Description
1	Maximum	$X_{MAX} = max(x_i)$
2	Mean	$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i$
3	Root mean square	$X_{RMS} = \sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}$
4	Variance	$X_V = \frac{\sum_{i=1}^{n}(x_i-\mu)^2}{n-1}$
5	Standard deviation	$\sigma = \sqrt{\frac{\sum_{i=1}^{n}(x_i-\mu)^2}{n-1}}$
6	Skewness	$X_S = \frac{1}{n}\frac{\sum_{i=1}^{n}(x_i-\mu)^3}{\sigma^3}$
7	Kurtosis	$X_K = \frac{1}{n}\frac{\sum_{i=1}^{n}(x_i-\mu)^4}{\sigma^4}$
8	Peak-to-peak	$X_{P2P} = max(x_i)\text{-}min(x_i)$
9	Crest factor	$X_{CF} = \frac{max(xi)}{\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}}$

***Table 5**: Features extracted in time domain*

where n is number of data samples.

Maximum and mean feature refers to maximum and mean amplitude of signal. RMS is the square root of the mean square and it represents the average power of a signal. Variance and standard deviations measures the dispersion of signal around its mean value. The standard deviation defines how far the signal fluctuates from the mean, while the variance represents the power of this fluctuation. Skewness and kurtosis are more advanced statistical-based features

that can be used for a signal which is not purely stationary to examine the probability density function. In particular, skewness quantifies the asymmetry behaviour/spread of signal around its mean value/through its PDF, while kurtosis measures the peak value of the PDF and indicates if the signal is impulse in nature. Peak-to-peak describes the difference between highest amplitude value of the signal and the lowest one. Lastly, crest factor represents the ratio of peak values to the effective value (RMS). In other words, it indicates how extreme the peaks are in a signal.
In order to avoid the addition of redundant features in the dataset, it has been decided not to take into consideration the variance, that is just the squared value of standard deviation, and the crest factor, which is simply computed from max value and RMS.

For each of the six signals in the milling dataset, the selected 7 features are computed in Matlab by using the Signal Processing Toolbox. This extension includes all functions for both descriptive statistics and subsequent spectral measurements. ***Figure 20*** shows the code which has been developed for one signal. This operation creates a total of 42 new features in the dataset, 7 for each of the 6 signals.

```
>> for i=1:167
X = mill(i).smcDC
mill(i).max_smcDC = max(X)
mill(i).mean_smcDC = mean(X)
mill(i).RMS_smcDC = rms(X)
mill(i).std_smcDC = std(X)
mill(i).skewness_smcDC = skewness(X)
mill(i).kurtosis_smcDC =  kurtosis(X)
mill(i).p2p_smcDC = peak2peak(X)
end
```

Figure 20*: Matlab code to extract signal features in time domain*

Time domain features are meaningful but only reflect the signal changes over time. Analysing the signal in the time-domain only cannot be sufficient in order to describe the correlation between wear and signals in an adequate manner, thus the frequency domain features need to be extracted too.

4.1.2 Frequency domain

A frequency-domain analysis is a representation of how much of the signal lies within each given frequency band over a range of frequencies. It is an important tool in signal processing and is also referred to as power spectrum analysis. The power spectrum describes the signal's power distribution over a range of frequencies.

Spectral analysis considers the problem of determining the spectral content (i.e., the distribution of power over frequency) of a time series from a finite set of measurements. By looking at the spectrum, one can find how much power is contained in the frequency components of the signal.

A signal can be converted between the time and frequency domains by using the Fourier transform, which converts a time function into a sum or integral of sine waves of different frequencies, each of which represents a frequency component / which decomposes a function into the sum of a (potentially infinite) number of sine wave frequency components (***Appendix 5***).

Sensor signals captured during the experiment are discrete-time signals, since they correspond to samples of original signals taken (from a finite set of measurements) at a certain sampling rate. Hence, the spectral analysis is performed by using the Discrete Fourier Transform (DFT), which is the discrete counterpart of the Fourier Transform.

DFT operates on samples of the signal and provides a mathematical approximation to the full integral solution. It converts a finite sequence of equally-spaced samples of a function into a same-length sequence of equally-spaced samples of the Discrete-Time Fourier Transform (DTFT), which is a complex-valued function of frequency. The interval at which the DTFT is sampled is the reciprocal of the duration of the input sequence. An inverse DFT is a Fourier series, using the DTFT samples as coefficients of complex sinusoids at the corresponding DTFT frequencies. It has the same sample-values as the original input sequence. The DFT is therefore said to be a frequency domain representation of the original input sequence [40].

$$X_k = \sum_{n=0}^{N-1} x_n e^{-i2\pi kn/N} = \sum_{n=o}^{N-1} x_n \left[\cos\left(\frac{2\pi kn}{N}\right) - i \sin\left(\frac{2\pi kn}{N}\right)\right]$$

$$x_n = \frac{1}{N}\sum_{k=0}^{N-1} X_k \, e^{i\,2\pi kn/N}$$

where:

- N is the number of sample in the signal;
- n is the current sample considered (0, …, N-1);
- k is the current frequency considered (from 0 Hz up to N-1 Hz);
- X_k is a complex number that represents the amount of frequency k in the signal (amplitude and phase);
- x_n value of the signal at sample.

Practically, the DFT is computed by an efficient algorithm called Fast Fourier Transform (FFT). The FFT rapidly computes such transformations by factorizing the DFT matrix into a product of sparse (mostly zero) factors. As a result, it manages to reduce the complexity of computing the discrete Fourier transform of series of data samples. The output of the FFT is a complex vector containing information about the frequency content of the signal. The magnitude tells you the strength of the frequency components relative to other components. The magnitude is conveniently plotted in a logarithmic scale (dB) [41].

In the frequency domain, DFT is used to compute a type of power spectrum called periodogram (S). This power spectrum is measured in the 0 to half of sampling rate frequency band as the square of the DDT's magnitude.

$$S = \frac{|F(X)|^2}{N}$$

where F(signal) is the Fourier transform of the signal X, and N is the normalization factor, which corresponds to the number of samples in the signal.

Typical signal features which can be extracted in frequency domain are summarize in ***Table 6*** as found in the literature [38]:

Index	Feature	Description
1	Maximum of band power spectrum	$S_{MAX} = max(S(f)_i)$
2	Sum of band power spectrum	$S_{SBP} = \sum_{i=1}^{n} S(f)_i$
3	Mean of band power spectrum	$S_{\mu} = \frac{1}{n} \sum_{i=1}^{n} S(f)_i$
4	Variance of band power spectrum	$S_V = \frac{\sum_{i=1}^{n} (S(f)_i - S_{\mu})^2}{n-1}$
5	Skewness of band power spectrum	$S_S = \frac{1}{n} \frac{\sum_{i=1}^{n} (S(f)_i - S_{\mu})^3}{S_V{}^{3/2}}$
6	Kurtosis of band power spectrum	$S_K = \frac{1}{n} \frac{\sum_{i=1}^{n} (S(f)_i - S_{\mu})^4}{S_V{}^{4/2}}$
7	Relative spectral peak per band	$S_{RSPPB} = \frac{max(S(f)_i)}{\frac{1}{n} \sum_{i=1}^{n} S(f)_i}$

Table 6*: Features extracted in frequency domain*

It has been decided to compute standard deviation instead of variance and to exclude the relative spectral peak per band measure, since it can be derived directly from the ratio between maximum and mean value of power spectrum, hence it is redundant. The resulting features are 36, 6 for each of 6 signals.

Once again, computations are performed in Matlab by applying spectral transformations and statistic functions (Figure () with code). The Fast Fourier Transform (FFT) is implemented by the function *fft()*.

```
>> for i=1:167
X = mill(i).vib_table
Y = fft(X)
S = (abs(Y).^2)/N
mill(i).maxpower_vibtable = max(S)
mill(i).totalpower_vibtable = sum(S)
mill(i).avgpower_vibtable = mean(S)
mill(i).stdpower_vibtable = std(S)
mill(i).skewnesspower_vibtable = skewness(S)
mill(i).kurtosispower_vibtable =  kurtosis(S)
end
```

Figure 21*: Matlbal code to extract signal features in frequency domain*

The total features being extract are 78: 42 features in time domain and 36 in frequency domain.

Nevertheless, using a huge amount of features is not advisable as it makes the problem more complex (from a computational point of view, given its increased dimensionality/ as it increases the dimension of the problem) and tends to over-fit the model and compromise its generality. Consequently, it is essential to select a subset of generated features. This operation takes the name of feature selection and will be discussed in ***Section 4.3***, after cleaning the data.

The obtained dataset presents 167 rows and 85 columns. The next step is to transform the Matlab structure in a ".csv" format so that it could be fed into Microsoft Azure Machine Learning Studio, and from here on, all the subsequent operations will be implemented in the platform.

4.2 Data cleaning

Data cleaning is one of the major steps in the overall data preparation phase. It is the process of identifying and correcting (or removing) incomplete, improper and inaccurate data. The aim is to address what are referred to as data quality issues, which negatively affect the quality of the model and compromise the analysis process and results. There are several types of data quality issues, including missing values, duplicate data, outliers, inconsistent or invalid data. Some techniques to clean data involve removing data records with missing values, merging duplicate records, generating a best, or at most reasonable, estimate for invalid values.

The dataset obtained from the feature extraction phase (***Section 4.1***) is saved in the platform's cloud ("*milling.csv*"). This dataset is a 167 x 85 matrix, plus headers. It will be "dragged and dropped" in the canvas as the starting point of the model.

4.2.1 Missing values

Missing values occur when no data value is stored for the variable in an observation [42]. There are two ways to handle missing data: deletion and imputation. The first method has to do with the removal of data for an observation that has one or more missing values, either the deletion of entire row or the deletion of the columns containing missing value, and it is used especially when no relationship exists between missing variable and other variables. The second technique refers to the estimation of missing value and it is preferred when missing value is dependent on some other variable's value.

Since during the experiment the flank wear was not measured for each run, the dataset presents some empty values in the VB measurement. To deal with this problem, the dataset is fed to the *Clean Missing Data* module, which can treat missing values in different ways: custom substitution value, replace value with mean-median-mode, or remove. Given the fact that empty value in the Flank wear measurement (VB) cannot be used to train and test the supervised model and that missing data is limited to a small number of observations (21), removing the entire row was chosen for the Thesis to eliminate those cases from the analysis. The resulting dataset has 146 rows and 85 columns.

4.2.2 Outliers

The second module is an *Execute Python script* which performs the task of detecting and removing outliers. An outlier is an observation point that lies an aberrant distance from the majority of the other observations. It may arise due to natural variability in measurement or it may indicate experimental errors, such as human error and instrument inaccuracy. Outliers can produce serious problems in data mining tasks and must be identified and analysed in order to understand their nature and evaluate their exclusion from the dataset. Most used techniques to detect outliers include statistical methods like standard deviation, z-score or interquartile range method. All these approaches foreseen the computation of statistics referring to dispersion of data around mean value for some, or all, variable (columns) and the identification of the observations (rows) whose value for at least one variable does not fall within a certain range of variability.

After removing missing values, in the milling dataset there is only one clear outlier that can be detected by simply looking at the data. *(Case 2, Run 1)* has signal features whose values greatly differ from the ones computed for all other observations. For instance, the value of "*max_smcDC*" for this observation has an order of magnitude 19 times greater than other observations (***Figure 22***).

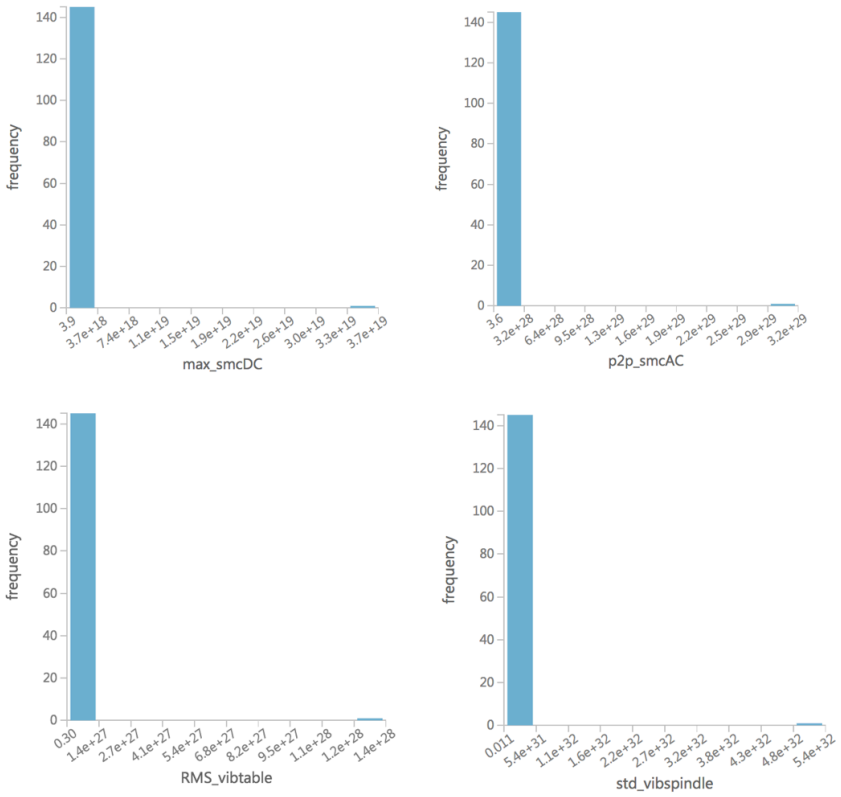


***Figure 22**: Frequency distribution of some sample signal features*

Before considering the possible elimination of this point from the dataset, it's fundamental to understand why it appeared. The reason can be found by plotting the signals as registered by the sensors (***Figure 23***).

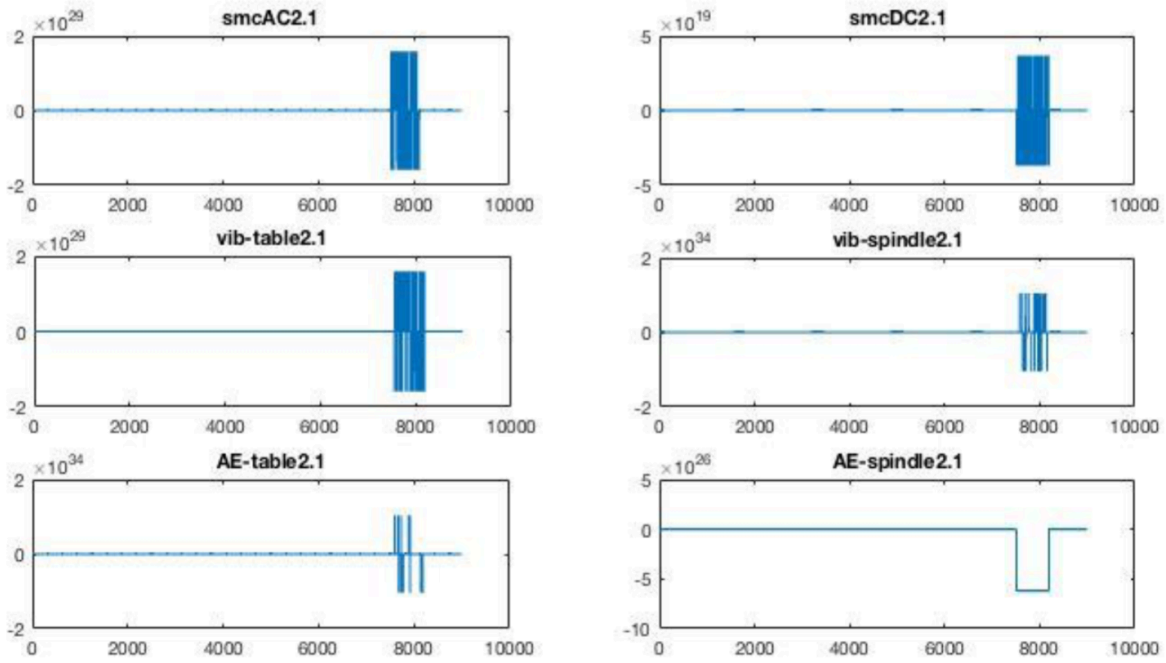


***Figure 23**: Signal values registered by sensors for Case 2, run 1*

As can be seen, the recorded signals show a particular trend with 0 mean value and peaks of 10^{19} - 10^{34} order of magnitude around 8000 points in the sample. This tendency is abnormal compared to the one of all other observations (see ***Appendix 4*** for typical sensor signals in the experiment), hence we can deduct that the outlier derives from an error in sensor measurement/acquisition during the experiment. Therefore, this observation point can be safely removed from the dataset. The *Execute Python script* (***Figure 24***) takes the dataset and drop the row 13 corresponding to *(Case2, Run1)*. The resulting dataset has 145 rows and 85 columns.

```python
# imports up here can be used to
import pandas as pd
import numpy as np

# The script MUST contain a function named azureml_main,
# which is the entry point for this module.
# The entry point function can contain up to two input arguments:
#   Param<dataframe1>: a pandas.DataFrame
#   Param<dataframe2>: a pandas.DataFrame
def azureml_main(dataframe1 = None, dataframe2 = None):

    # Execution logic goes here
    dataframe1.drop(dataframe1.index[13], inplace=True)

    # Return value must be of a sequence of pandas.DataFrame
    return dataframe1
```

***Figure 24**: Python code to eliminate the outlier representing Case 2, run 1*

While in this circumstance, given the limited number of observations, outliers were easily spotted by purely looking at the data and "manually" removed by indicating the exact row,

generally more automatic functions could be implemented by adopting the above mentioned statistical methods. For instance, if one would like to remove all rows which have outliers in at least one column of the dataset by applying the z-score method, the following expression in Python would do that in one-line code: *df[(np.abs(stats.zscore(df)) < 3).all(axis=1)]*.

4.3 Data normalization

The cleaned dataset is connected to the *Normalize Data* module. Data normalization, also known as feature scaling or data standardization, is an important step in the data pre-processing phase. It is a technique used to standardize the range of features in the dataset, which means to adjust values of numeric columns measured on different scales to a notionally common scale, without altering differences in the values' ranges or losing information. The goal of normalization is to improve the overall quality of a dataset by rescaling the dimension of the data and avoiding situations in which some values overweighting others, since great differences in scale of numbers could cause problems when attempting to combine values as features during modelling: any distance metric would automatically say a change in a variable having a small variance is less significant than a change in a variable with a large variance. *Normalization* eludes these problems by creating new features which maintain the general distribution and ratios in the source data, while keeping values within the same scale applied across all columns used in the model [43]. There are several normalization methods, among which the most famous include z-score, min-max and lognormal transformation

In the dataset, while signal features are important, the values taken vary in scale and some of them are very high (for instance the features related to maximum and total power of signals) and will tend to overweight other features such as "feed", "DOC" or "material" that are few-digit values. Hence, the *Normalize Data module* is added to take all features representing the signals and performs a Min-Max normalization on the data. The min-max normalization consists in linearly rescaling the range of features to the [0,1] interval. Rescaling is performed by shifting the values of each feature in such a way that the minimum value is zero, and then dividing by the new maximum value. The values in the column are transformed applying the following formula:

$$z = \frac{\mathrm{x} - \min(x)}{\max(x) - \min\,(x)}$$

4.4 Feature selection

As previously mentioned in ***Section 4.1***, feature extraction allows to obtain relevant information from raw dataset, but at the same time it increases its dimensionality, (i.e. the volume of the space increases so fast that the available data become sparse.) The number of features in the cleaned and normalized dataset is indeed very large, containing a total of 78 features extracted from sensor signals in time and frequency domains. Some of these features could be either irrelevant or redundant and could negatively influence the performance of the monitoring and prediction model by overwhelming the algorithms. In order to improve the accuracy of the model and increase the efficiency of calculation performance, it is desirable to reduce the number of utilized features.

For this purpose, the feature selection technique is adopted in this paper. Feature selection is a process of selecting a subset of relevant features to be used in the model construction. Choosing informative, discriminating and independent features is a crucial step for effective machine learning algorithms. It leads to multiple advantages such as (1) reducing the complexity of the model and making it easier to interpret, (2) speeding up training time, (3) improving the predictive accuracy and (4) enhancing generalization by reducing overfitting [44].

There are several methods to implement feature selection. Feature selection algorithms may use a scoring method to rank and choose features, such as correlation or other feature importance methods. More advanced methods may search subsets of features by trial and error, creating and evaluating models automatically in pursuit of the objectively most predictive sub-group of features.

Figure 25 shows the flow chart implemented in Azure to perform the feature selection phase.

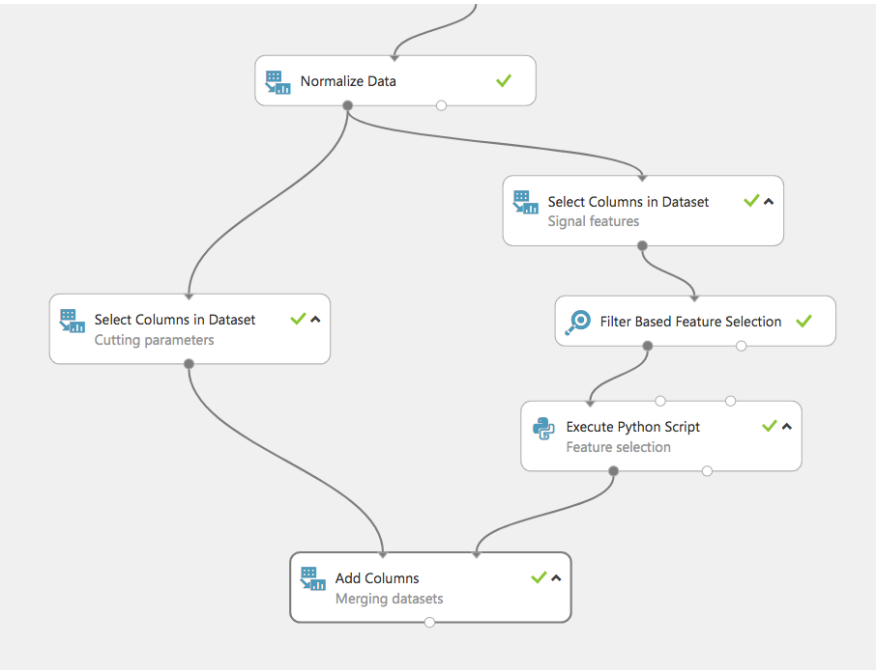


Figure 25*: Feature selection as implemented in Azure MLS*

It has been decided to limit feature selection to signal features, in order to avoid loss of information about cutting parameters. Thus, the dataset is first split into two subsets: one containing data on case, run, feed, DOC and material, the other containing remaining columns related to VB and sensor signals. The *Execute Python Script* includes the effective elaboration of feature selection. The feature selection method chosen for the Thesis involves a statistical approach based on two steps: analysis of correlation matrix and multicollinearity. The aim of this phase is to perform statistical tests on features to determine and remove the ones strictly correlated to one another or clearly explained/derivable by some others.

4.4.1 Correlation matrix

Statistical analysis of correlation matrix is adopted to remove highly correlated features based on the Pearson's correlation coefficient. A correlation matrix is a table, square and symmetrical,

showing correlation coefficients between variables. Correlation coefficient describes the degree of any statistical relationship/association between two variables. There are several correlation coefficients, the most common is the *Pearson's correlation coefficient (PCC)*, which is a measure of the linear relationship between two variables. PCC is calculated as the covariance of the two variables divided by the product of their standard deviations when applied to a population. It assumes a value between +1 and −1, where 1 indicates a total positive linear correlation, −1 a total negative one and 0 says there is not a linear correlation. Correlation matrix can be used as a tool to select relevant features: if a features shows a high correlation with another one, it means changes in one variable are associated with shifts in another variable, thus one of the two is redundant in the dataset and can be removed from the model without incurring loss of information.

To automatically detect and eliminate the variables showing high correlation in the dataset, in the *Execute Python Script* module (***Figure 26***), the Pearson correlation coefficient is computed for each pair of features; when a feature has correlation greater than 0.9 with at least another variable, it is simply removed from the set of significant features.

```
15 def azureml_main(df):
16
17     features = []
18     for x in list(df.columns):
19         features = features + [x]
20
21     sel = []
22
23     for x in features:
24        sel = sel + [x]
25        for y in features[features.index(x)+1:]:
26             [cor,p] = stats.pearsonr(df[x], df[y])
27             if abs(cor) > 0.9 and p<0.05:
28                     sel.remove(x)
29                     break
30     features=sel
31
32     df = df[features]
```

***Figure 26**: Python script to eliminate features showing a coefficient of correlation (PCC) greater than 0.9*

4.4.2 Multicollinearity

Another phenomenon which can occur in a dataset is multicollinearity. Multicollinearity arises when an independent/explanatory variable in a multiple regression model can be linearly predicted from the other variables with a significant degree of accuracy. One can assess the existence of this relationship by obtaining estimates for the parameters of the multiple regression equation:

$$Y_i = \beta + \beta X_{1i} + \cdots + \beta X_{ki} + \varepsilon_i$$

where Y_i is any explanatory variable treated as a predictive one to evaluate the level with which the others independent variables forecast it. The strength of the correlation is evaluated by

means of the coefficient of determination R^2 (R-squared), which indicates the percentage of variance in the dependent variable that can be explained by independent variables. A detailed explanation of this statistic is provided in ***Section 5.2.1*** when discussing the evaluation of regression algorithms. The coefficient of determination ranges from 0 to 1 and the higher is its value, the higher is the linear dependency between variables.

Multicollinearity is a type of disturbance in the data which does not reduce the predictive power or reliability of the model as a whole but it makes it hard to evaluate the relative importance of the independent variables in explaining the variation caused by predictor variables [45]. In the use case dataset, therefore, multicollinearity between signal features can reduce the ability of the model to identify those features that are statistically significant to predict the flank wear VB.

To avoid this problem, the *Execute Python Script* takes the data frame of features resulting from analysis of simple correlation, isolates one variable at a time from the set and performs multiple linear regression considering the isolated feature as dependent variable and computes the R^2. If the obtained R^2 is higher than 0.9, the feature is removed from the dataset. The code excerpt below (***Figure 27***) shows the definition of the function for computing the coefficient of determination and the iterative operation of evaluating the features.

```
34      ciclo = features[:]
35
36      def rsquared(df,i):
37          target = df[i]
38          df = df.drop(i, axis=1)
39          X = add_constant(df)
40          y = target
41          lm = linear_model.LinearRegression()
42          model = lm.fit(X,y)
43          r = lm.score(X,y)
44          return r
45
46      for x in ciclo:
47          r = rsquared(df[features], x)
48          if r > 0.9:
49              features.remove(x)
50              df = df.drop(x, axis=1)
```

Figure 27*: Python script to eliminate features that can be explained by other features with a coefficient of determination (R-squared) greater than 0.9*

4.4.3 Improved feature selection

The operations described above work well but they both tend to favour the last features listed in the dataset, just due to the implementation of the code which always removes the first variable in cases of high correlation and multicollinearity (and not as a preference for the construction of the model). To improve the entire process, it is possible to pre-organize the features before performing the selection in a way that is significant for the final purpose of

predicting the tool wear. The significance is evaluated in terms of correlation between each signal feature and the value of VB. Therefore, a *Filter Based Feature Selection* module is added in the flow chart to compute the PCC of any pair of signal feature and flank wear VB and to rank features in decreasing order of correlation. The module allows also to define a maximum number of desired features to keep in the dataset, but this option is not utilized since an optimal number is not known a priori. To do so, the module is configured by specifying a number greater than the number of columns in the dataset, so that all features are returned (***Figure 28***).

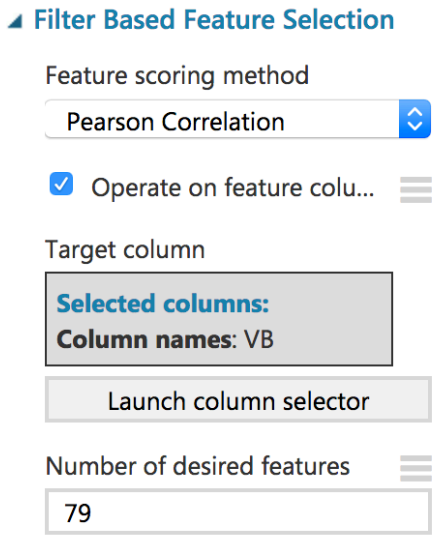


Figure 28: *Configuration panel of Filter Based Feature Selection module in Azure MLS*

The ordered dataset with the 78 features is fed into the *Execute Python Script module* (***Figure 28***), which, before applying the analysis of correlation matrix and multicollinearity as mentioned above, reverses the list of features so to have them in increasing order of correlation with VB, thus promoting the algorithms to keep the most meaningful ones for the end of the work.

```
17      features = df.columns.tolist()
18      features = features[::-1]
```

Figure 29: *Python script to reverse list of features*

After applying the analysis of correlation matrix and multicollinearity, the selected features are 23 out of 78: “kurtosispower_smcDC”, “skewness_AEtable”, “kurtosispower_AEtable”, “kurtosis_AEspindle”, “skewness_AEspindle”, “kurtosis_AEtable”, “kurtosispower_smcAC”, “skewness_smcDC”, “skewness_vibtable”, “skewnesspower_vibspindle”, “kurtosis_vibtable”, “skewness_smcAC”, “skewnesspower_vibtable”, “mean_vibspindle”, “std_vibtable”, “p2p_vibspindle”, “skewnesspower_AEspindle”, “skewness_vibspindle”, “mean_smcAC”, “max_AEspindle”, “max_AEtable”, “stdpower_smcAC”, “maxpower_smcDC”.

Lastly, the module *Add Columns* simply merges the dataset containing this selected features with the one containing the cutting parameters. The combined dataset has 145 rows and 30 columns.

4.5 Data transformation

The resulting dataset contains only numeric values, though some of them do not represent numbers but categories. As outlined in the dataset description (***Section 3.4***), the "material" field is a categorical feature, which means a variable that can take on one of a limited, and often fixed, number of possible values. Indeed, it can assume only values "1" and "2" which indicate respectively cast iron and steel. Thus, numeric values just represent the two types of material adopted in the experiment. In Azure MLS, it is important to specify which values should be treated as numeric or categories, in order to ensure that categorical parameters are not uses in mathematical calculations, but only to group data [46].

The *Edit Metadata module* is used for this purpose: it simply takes the "material" feature and flag it as a categorical one. **Figure 30** shows the configuration panel for the module, where the "material" column is selected and it is indicated to make the variable categorical.

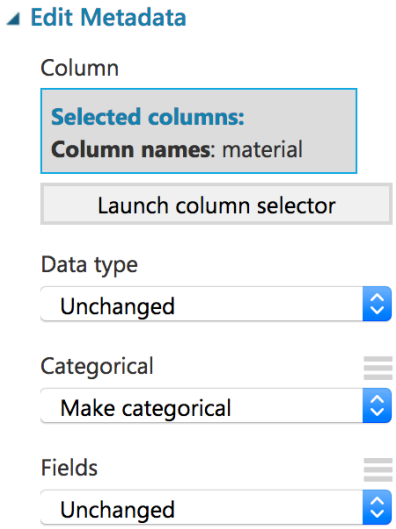


***Figure 30**: Parameter setting for Edit Metadata module in Azure MLS*

The data preparation phase developed until this point is typical for any kind of data mining task and it is just related to the input dataset. According to the type of machine learning algorithms subsequently applied, classification and regression algorithms as chosen for the purpose of the Thesis, an additional specific preparation of data is required before the training and testing stage. For the sake of clarity, the following sections will be divided in 2 parts: one for the classification task and one for regression task.

4.6 Data preparation – Classification

The objective of this Thesis is to predict when the cutting-tool must be replaced. A way to do so is to implement a binary classification model where instances are classified according to the value of flank wear. As discussed in ***Section 3.2.2***, the ISO standard *ISO 8688-1:1989* indicates as critical flank wear the value VB=0.6 for this particular type of operation. Hence, two classes are created using this measure as partition and the model is successively trained to assign new instances to the correct class, in order to identify whether the tool can be safely used or it has to be substituted prior to that run.

4.6.1 Classes definition

A classification model in Azure MLS requires the definition of a label feature designating the class. The *Execute Python script* (***Figure 31***) creates the 2 classes used to classify the instances by categorizing wear in [0, 0.6) as “safe”, indicating the normal cutting condition, and wear in [0.6, +∞) as “worn”, indicating that the substitution has to be done. This operation adds a new feature “label” to the dataset, reaching a total of 31 columns.

```
1  import pandas as pd
2
3
4  def azureml_main(df):
5
6      classes = []
7      for x in df['VB']:
8          if x >= 0.6:
9              y = 'worn'
10         if x < 0.6:
11             y = 'safe'
12
13         classes = classes + [y]
14
15     df['label'] = classes
16
17
18     return df
```

Figure 31*: Python script for creating classes*

4.6.2 Imbalanced data: SMOTE algorithm

Figure 32 shows the classes distribution across the dataset. The classes are not balanced since the number of observations belonging to the class “worn” (20) is significantly lower than that of the “safe” class (125). This problem is known as imbalanced dataset.

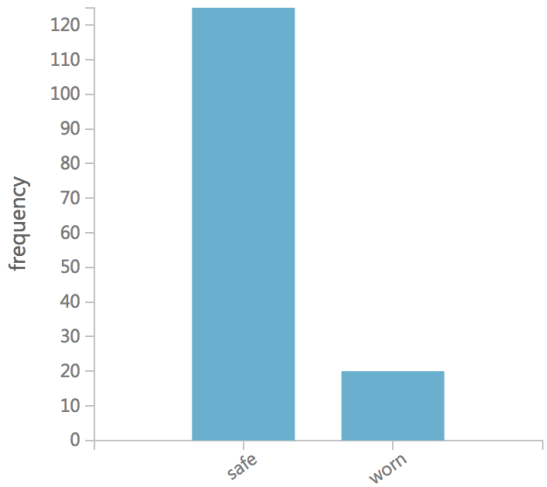


Figure 32*: Class distribution across the dataset: 125 instances labelled as "safe", 20 instances labelled by "worn"*

Imbalanced data typically refers to a situation in classification tasks where classes are not represented equally. This can cause problems with how the model will classify instances since machine learning algorithms tend to produce unsatisfactory results when faced when the class of primary interest is under-represented. Indeed, standard classifier algorithms like Decision Tree or Logistic Regression have a bias towards classes which have a higher number of instances: they tend to predict the majority class data. The minority classes are treated as noise and are often ignored, leading to a high probability of misclassification of the minority class.
Imbalanced data is a frequent problem in a predictive maintenance model as failures rarely can be recorded since the objective is to prevent machineries from failing. In our use case, imbalanced data is not as emphasized as other predictive maintenance models since the classes numerosity is of the same order of magnitude. Still, misclassification of a "worn" insert to a "safe" insert can lead to costly waste of workpieces.

Azure MLS treats imbalanced data with the *SMOTE* module [47]. SMOTE stands for Synthetic Minority Oversampling Technique. It is a statistical technique to adjust the number of the class distribution of a dataset: it increases the number of cases in the dataset in a balanced way, by generating new instances of the minority class. The new instances are not just copies of existing cases; instead, the algorithm takes samples of the feature space for each target class and its nearest neighbours, and generates new examples that combine features of the target case with features of its neighbours. This approach allows to balance the dataset without as much overfitting, as it creates new synthetic examples rather than using duplicates The SMOTE module in Azure takes the entire dataset as an input and generates new instances from existing cases. It returns a dataset containing original samples, plus an additional number of synthetic minority cases, depending on the percentage one specifies. This implementation of SMOTE does not change the number of majority cases.

Figure 33 illustrates the configuration panel of SMOTE module. Smote percentage option indicates the target percentage of minority cases in the output dataset. For instance, typing 100% results in adding the same number of minority cases that were in the original dataset, while 200% doubles the percentage of minority cases compared to the original dataset. This second option does not result in having twice as many minority cases as before. Rather, the size of the dataset is increased in such a way that the number of majority cases stays the same, and the number of minority cases is increased till it matches the desired percentage value.

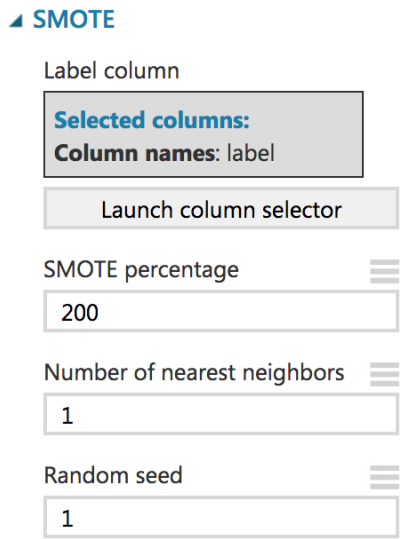


***Figure 33**: SMOTE module configuration panel in Azure MLS*

Number of nearest neighbours box determines the size of the feature space that the SMOTE algorithm uses when in building new cases. A nearest neighbour is a row of data (a case) that is very similar to some target case. The distance between any two cases is measured by combining the weighted vectors of all features. By increasing the number of nearest neighbours, you get features from more cases, while by keeping the number of nearest neighbours low, you use features that are more like those in the original sample.
The SMOTE module generates a random seed based on processor clock values when the experiment is deployed, which can cause slightly different results over runs. To ensure the same results over runs of the same experiment, with the same data, it is necessary to type a value in the Random seed textbox.

Given the low number of "worn" instances, it has been decided to increase the percentage of minority cases to twice the previous percentage, thus entering 200 for SMOTE percentage in the module's properties. To use features more similar to those in the original dataset, the number of nearest neighbours is set to 1. Eventually, random seed box is filled with the value 1 to fix the results over runs of the same experiment in order to simplify the subsequent training and testing phase. The SMOTE module adds 200% more "worn" class cases with only 1 nearest neighbour, resulting in 40 new instances in the dataset. The updated number of the label classes is shown in ***Figure 34***, with a total number of instances in the dataset of 185, obtaining a more balanced class distribution.

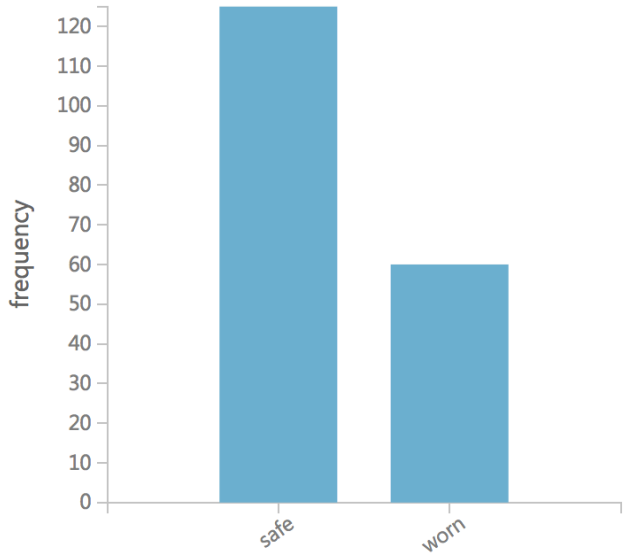


***Figure 34**: Updated frequency distribution of classes in dataset after the application of the SMOTE algorithm*

4.6.2 Selecting training and testing variables

The last two modules of the data preparation phase for the classification task specify the features to consider for the training and testing step. The *Edit Metadata module* selects the label feature, also known as the predictable attribute or target variable. In a classification task, the label corresponds to the column indicating the class. Usually, Azure Machine Learning can

automatically deduce which column contains a class label, but by setting this metadata one can ensure that the correct column is identified.

Label assignment depends on the objective of the model, which in our case is to classify the state of the in inserts' wear in a specific run. In the dataset, the measurement of the wear is given (VB), and thus the label. However, in order to perform a classification, the continuous variable VB is not suitable, that is why the classes were previously computed. ***Figure 35*** shows the configuration panel for this module, where the Fields option is used to mark the label.

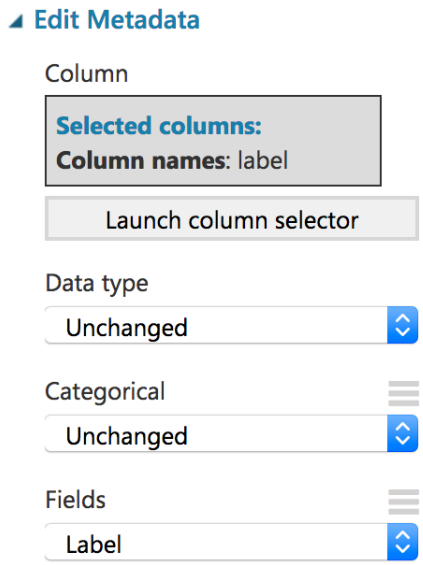


***Figure 35**: Configuration panel for Edit Metadata module in Azure MLS to mark the label*

Eventually, the *Select Columns in Dataset* excludes flank wear measurement (VB) column from the dataset. This step is necessary, otherwise the most important feature would be VB as the classes (and thus the label) are built based on it.

4.7 Data preparation – Regression

In the classification task, the objective of the model is to classify the state of the in inserts' wear as "safe" or "worn" in each run, to determine when the cutting tool must be replaced. As an alternative way to evaluate the cutter's substitution, the flank wear can be predicted by implementing a regression model which directly estimates the continuous value VB. The milling machine operator will then be aware of the state of the tool and he will change it when the predicted VB is higher than 0.6. The data preparation step for the regression task is more streamlined than the classification one as it only involves the identification of the label feature.

4.7.1 Selecting label feature

Unlike Classification, a Regression model returns a real number rather than a discrete class. The choice of the label depends on the results we want answered by the model. In this case, the label feature case corresponds to the column VB itself, which is the value we want to predict. As for the classification task, the *Edit Metadata* is added to the canvas to mark the label (VB).

5. Model Training and Testing

Training and testing is a crucial phase in order to evaluate the performance of the model. First of all, the term machine learning model refers to the model artefact that is created by the training process. The process of training a machine learning model involves providing a ML algorithm with the training data to learn from [48]. The training data must include the correct answer, which is known as a label or target attribute, and related variables. The learning algorithm looks for patterns in the training data that explain how the input data attributes affect the target, and it provides an ML model that is able to captures these patterns. Machine learning models are used to make predictions on new data on which the label is not given.

Creating a training and testing split of a dataset is one method to rapidly assess the performance of an algorithm on a given problem. The training set is used to fit the model, while the testing set is used to make predictions based on the trained model, by pretending that the label in the test set is not specified. Comparing the predicted and actual outputs of the label feature in the test dataset allows to evaluate a performance measure for the machine learning model. This is an estimate of the skill of the algorithm trained on the problem when making predictions on unseen data. Unfortunately, there is no perfect algorithm suitable for all kind of problems yet, but the most performant ML algorithm exist for a specific problem. In general, we don't know which ML algorithm gives the highest result, thus it is best practice to train different ones and sort out the most appropriate.

5.1 Classification Algorithms

For the Classification task, five machine learning algorithms are trained, tested, and compared against each other. Since the classes of the label are two, binary classification algorithms are chosen to train the model. The choices fall for Two-class Logistic Regression, Two-class Decision Forest, Two-class Decision Jungle, Two-class Boosted Decision Tree and Two-class Neural Network.

Figure 36 illustrates the model training and testing phase for the classification task in Azure MLS canvas. In this phase, it has been decided to proceed by using the default settings for each algorithm, just to test which, by default, perform better. Later, the model will be improved using improving model techniques.

As first step, the *Split Data* module randomly splits the instances into training and testing set. The proportion chosen are 70% (130 instances) for training and 30% (55 instances) for testing. Machine Learning Studio provides a flexible and customizable framework for model training and testing phase. Each task in this process is performed by a specific type of module, which can be modified, added, or removed, without breaking the rest of the experiment [49]. All machine learning algorithms are trained and tested with the same modules:

- the *Train Model* module configures the model to learn from training data.
- the *Score Model* module creates predictions of label class for the testing data using the trained models.
- The *Evaluate Model* module assesses the performance of a trained model by standard metrics.

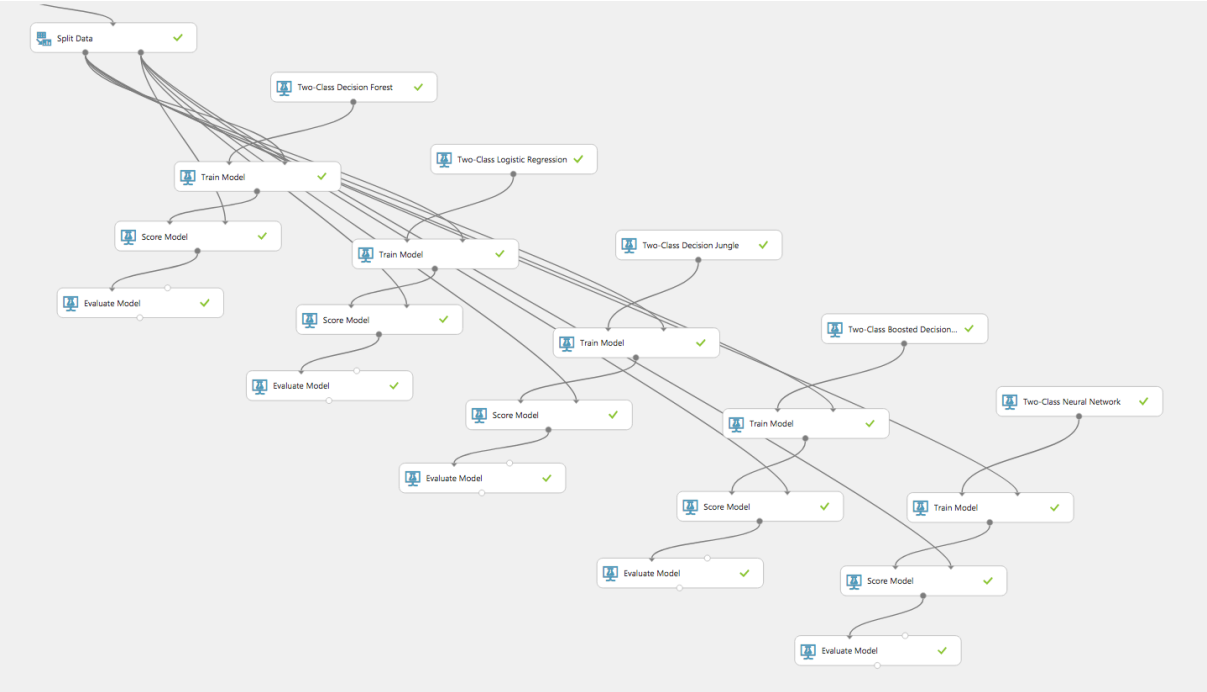


***Figure 36**: Model training and testing phase for the classification task in Azure MLS*

5.1.1 Confusion matrix

The *Evaluate Model* module for classification returns the confusion matrix, which contains information about the actual and the predicted class, along with some metrics to evaluate the performance of the model. ***Figure 37*** below shows the confusion matrix for a binary classification task.

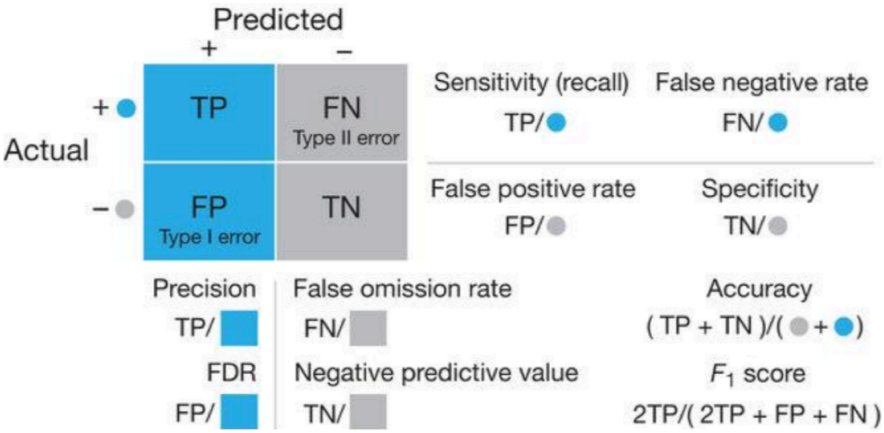


***Figure 37**: Confusion matrix and evaluation metrics for a classification task*

A confusion matrix is a table which allows to visualize the prediction results on a classification problem and to evaluate the performance of an algorithm. Each row of the matrix represents the instances in a predicted class while each column represents the instances in an actual class. The number of correct and incorrect predictions are summarized with count or percentage

values and broken down by each class. It gives the vision not only into the errors being made by the classifier but more importantly the types of errors that are being made.

The class labels in the training set can assume only two possible values, usually referred to as positive or negative. Azure MLS automatically establishes which of the two classes in the dataset is the positive one. The positive and negative instances that a classifier predicts correctly are called true positives (TP) and true negatives (TN), respectively. Similarly, the incorrectly classified instances are called false positives (FP) and false negatives (FN). FP rate is also known as type I error, while FN rate is recognized as type II error. The confusion matrix simply shows the number of instances (in percentage terms) that fall under each of these four categories [50].

The *Evaluation Model* module also reports some standard metrics available for binary classification algorithms which include *Accuracy, Precision, Recall, F1 Score,* and *AUC*.

The **accuracy** measures the goodness of a classification model as the proportion of correctly classified instances to total cases [(TP+TN)/(FP+TN+TP+FN)]. It is usually the first metric one looks at when evaluating a classifier. Nevertheless, when the dataset is unbalanced or one is more interested in the classification performance on one of the label classes, accuracy does not truly capture the effectiveness of a classifier. For this reason, it is helpful to compute additional metrics that describe more detailed aspects of the evaluation.

Precision is the proportion of true positives, i.e. the number of items correctly labelled as belonging to the positive class, over all positive results [TP/(TP+FP)].

Recall is the fraction of correct results over all results that should have been returned. It is computed as the number of true positives divided by the total number of elements that actually belong to the positive class [TP/(TP+FN)].

Another metric that is used is the **F1-score**, which is a measure that balances precision and recall. It is computed as the weighted average mean of these two metrics [2TP/(2TP+FP+FN)], where the ideal value is 1. The F1-score is a good way to summarize the evaluation in a single number, but it's always a good practice to look at both precision and recall together to better understand how a classifier behaves.

In addition, one can inspect true positive rate vs. false positive rate with the Receiver Operating Characteristic (ROC) curve and the corresponding Area Under the Curve (**AUC**) value. The ROC curve is created by plotting the true positive rate against the false positive rate at various threshold settings. The closer this curve is to the upper left corner, the better the classifier's performance is. The AUC is a useful measure because it provides a single number that lets you compare models of different types.

5.1.2 Two-class Decision Forest

The first algorithm in the canvas is the *Two-class Decision Forest*. The decision forest algorithm is an ensemble learning method for classification. Ensemble models provide better coverage and accuracy than single decision trees. The decision forest classifier in Azure MLS operates by constructing multiple decision trees (***Appendix 6***) and then voting on the most popular output class. In detail, many individual classification trees are created, using the entire

dataset, but different and usually randomized starting points. Then, each tree in the decision forest tree outputs a non-normalized frequency histogram of labels and the aggregation process sums these histograms and normalizes the result to get the "probabilities" for each label. The trees that have high prediction confidence will have a greater weight in the final decision of the ensemble [51]. The algorithm has multiple parameters to configure the model, but, as for now only the default parameters are used.

Bagging technique, also called bootstrap aggregating, is chosen as Resampling method, i.e. method used to create the individual trees. In this method, each tree is built on a new sample set, which is created by randomly sampling the original dataset with replacement until a dataset with the same size of the original one is obtained. This bootstrapping procedure leads to better model performance because it reduces the variance of the model and helps to avoid overfitting. Single parameters are provided to construct the algorithm. The maximum number of decision trees which can be created in the ensemble is set to 8. By creating more decision trees, one can potentially get better coverage, but training time increases. The maximum depth of the decision trees is set to 32. Increasing the depth of the tree might increase precision, at the risk of some overfitting and increased training time. The number of random splits used when building each node of the tree is 128. A split means that features in each level of the tree (node) are randomly divided. Eventually, the minimum number of samples requires to build any leaf, i.e. terminal node, in a tree is 1, which means even a single case can cause the creation of a new rule.

Figure 38 shows the results of this first model. Although the metrics indicate good results, the prediction seems a little bit off, since it misclassifies up to around 20% of the "worn" class, a situation not desirable since the objective is to keep near-worn inserts under monitor.

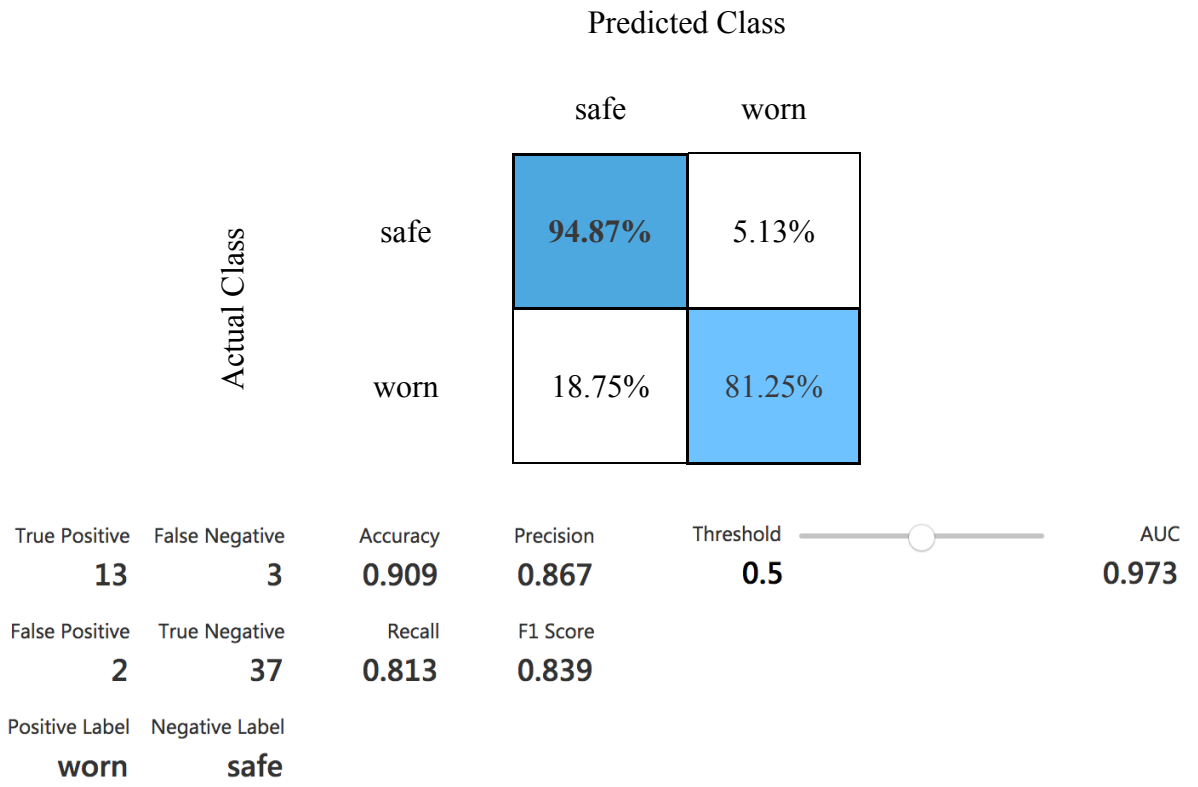


***Figure 38**: Confusion matrix and evaluation metrics for the Two-Class Decision Forest algorithm*

5.1.3 Two-class Logistic Regression

The second ML algorithm is the *Two-class Logistic Regression*. Logistic regression is a well-known method in statistics that is used to predict the probability of an outcome, and is particularly popular for classification tasks. The algorithm predicts the probability of occurrence of an event by fitting data to a logistic function [52].

Logistic regression assumes a logistic distribution of the data, where the probability that an instance belongs to the default class (i.e. class 1) is given by the following formula:

$$p(x; \beta_0, \ldots, \beta_{D-1})$$

Where:

- x is a D-dimensional vector containing the values of all the features of the instance;
- p is the logistic distribution function;
- β{0},..., β {D-1} are the unknown parameters of the logistic distribution.

The algorithm tries to find the optimal values for the coefficients β{0},..., β{D-1} by maximizing the log probability of the parameters given the inputs. Maximization is performed by using a popular method for parameter estimation, called Limited Memory BFGS.

The algorithm requires the specification of multiple parameters to configure the model. Optimization tolerance denotes as threshold to be used when optimizing the model, the value 1e-07. If the improvement between iterations falls below this specified threshold, the algorithm is considered to have converged on a solution, and training stops.

In order to address overfitting, this algorithm makes use of two regularization techniques: L1 and L2 regularization. Since not all features are important in the model bust just some of them have actual predictive power, one way to avoid overfitting is to reduce predictors which are not so relevant in the model. Regularization is a method for preventing overfitting by penalizing high-valued regression coefficients. Penalization works by biasing those coefficient towards smaller values. L1 regularization adds a penalty equal to the absolute value of the magnitude of coefficients, while L2 regularization adds a penalty equal to the square of the magnitude of coefficients. L1 can be applied to sparse models, which is useful when working with high-dimensional data, since it forces some of the predictors to zero. In contrast, L2 regularization is preferable for data that is not sparse as it penalizes big coefficients and tries to minimize them, although not making them exactly to zero.

Two-class Logistic Regression supports a linear combination of L1 and L2 regularization values: that is, if x = L1 and y = L2, then ax + by = c defines the linear span of the regularization terms. The weights of the two regularization parameters indicate their impact on the model.

Another critical parameter to configure this model is the amount of memory to use for L-BFGS optimization. L-BFGS stands for "limited memory Broyden-Fletcher-Goldfarb-Shanno" and it is a popular optimization algorithm for coefficient estimation. This parameter defines the number of past positions and gradients to store for the computation of the next step. By specifying less memory, training is faster but less accurate. The default value 20 is applied in this phase.

A seed non-zero value is defined in order to reproduce the same results over multiple runs of the same experiment, just to allow a better evaluation.

Results are summarized in ***Figure 39***. Logistic Regression performs slightly better than Decision Forest with respect to "safe" class, improving the classification from 94.85% to 97.44%. However, the misclassification of the "worn" class is still high.

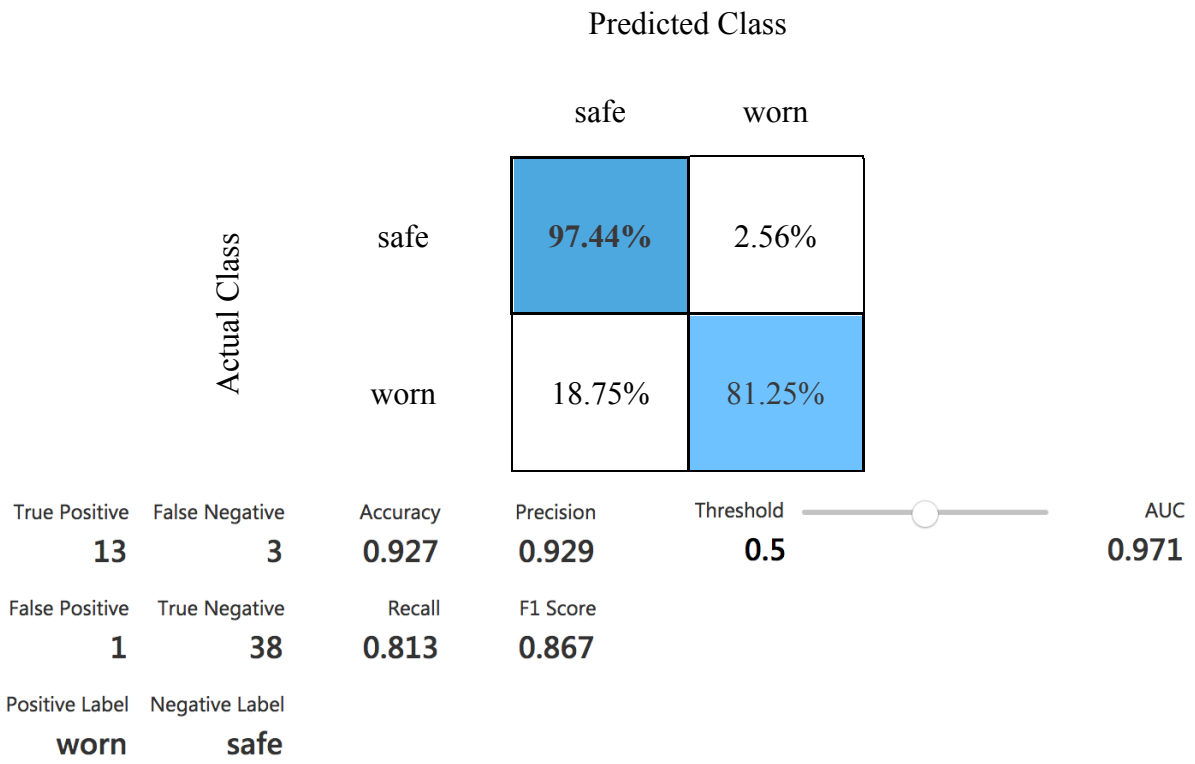


Figure 39*: Confusion matrix and evaluation metrics for the Two-Class Logistic Regression algorithm*

5.1.4 Two-class Decision Jungle

The third ML algorithm is the *Two-Class Decision Jungle*. Decision jungles are a recent extension to decision forests. A decision jungle consists of an ensemble of decision directed acyclic graphs (DAGs) Unlike conventional decision trees that only allow one path to every node, a DAG in a decision jungle allows multiple paths from the root to each leaf [53].

Decision jungles leads to several advantages [54]:

- By allowing tree branches to merge, a decision DAG typically require less memory while considerably improving generalization, at the cost of a higher training time;
- Decision jungles are non-parametric models, which can represent non-linear decision boundaries;
- They perform integrated feature selection and classification and are resilient in the presence of noisy features.

The configuration panel in Azure MLS requires the setting of several parameters. As for Decision Forest, bagging method is selected to create individual trees. Each tree in a decision forest outputs a Gaussian distribution as prediction. The aggregation is to find a Gaussian whose first two moments match the moments of the mixture of Gaussians given by combining all Gaussians returned by individual trees.
The maximum number of DAGs which can be generated in the ensemble is defined as 8. The maximum depth of each graph is set to 32, while the maximum width is set to 128. The number of optimization steps per decision DAG layer indicates the number of iterations over the data that are performed when building each graph (2048).

The results (***Figure 40***) are better than the Logistic Regression, outperforming it. The misclassification of "worn" class is highly improved (6%), but it could still be refined.

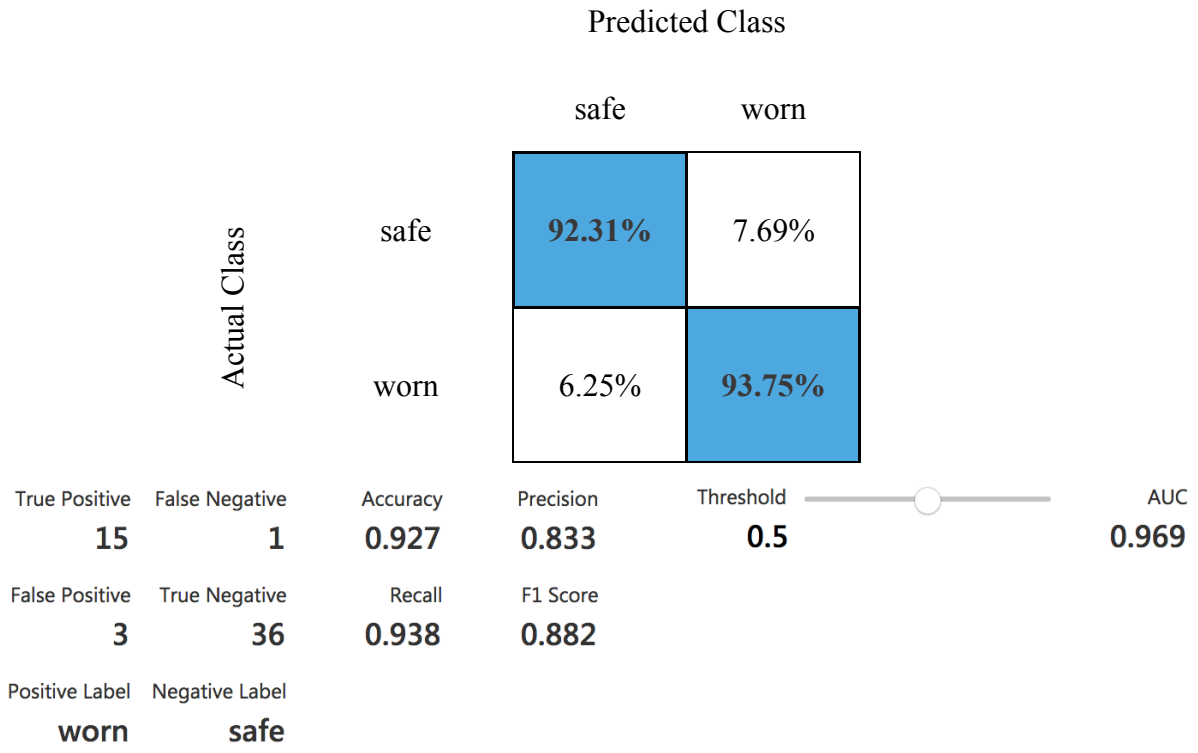


Figure 40*: Confusion matrix and evaluation metrics for the Two-Class Decision Jungle algorithm*

5.1.5 Two-class Boosted Decision Tree

The fourth ML algorithm is the *Two-class Boosted Decision Tree*. A boosted decision tree is an ensemble learning method which combines weak decision trees into a single strong learner in an iterative way. The general idea is to compute a sequence of very simple trees, where each successive tree is built for the prediction residuals of the preceding tree: the second tree corrects for the errors of the first tree, the third tree corrects for the errors of the first and second trees, and so forth. Predictions are based on the entire ensemble of trees together that makes the prediction [55].

The boosted decision tree algorithm in Azure MLS uses the following boosting method:

1. Start with an empty ensemble of weak learners.
2. For each training example, get the current output of the ensemble. This is the sum of the outputs of all weak learners in the ensemble.
3. Calculate the gradient of the loss function for each example. This depends on whether the task is a binary classification problem or a regression problem.
    - In a binary classification model, the log-loss is used, much like in logistic regression.
    - In a regression model, the squared loss is used, and the gradient is the current output, minus the target).
4. Use the examples to fit a weak learner, using the gradient just defined as the target function.
5. Add that weak learner to the ensemble with a strength indicated by the learning rate, and if desired, go to Step 2.

    In this implementation, the weak learners are the least-squares regression trees, based on the gradients calculated in Step 3. The trees are subject to the following restrictions:

    - They are trained up to a maximum number of leaves.
    - Each leaf has a minimum number of examples to guard against overfitting.
    - Each decision node is a single feature that is compared against some threshold. If that feature is less than or equal to the threshold, it goes down one path, and if it is greater than the threshold, it goes down the other path.
    - Each leaf node is a constant value.
6. The tree-building algorithm greedily selects the feature and threshold for which a split minimizes the squared loss with regard to the gradient calculated in Step 3. The selection of the split is subject to a minimum number of training examples per leaf.

    The algorithm repeatedly splits until it reaches the maximum number of leaves, or until no valid split is available.

The model requires the configuration of multiple parameters. The maximum number of leaves per tree, indicating the maximum number of terminal nodes which can be created, is set to 20. The greater this value, the greater the possibility to increase the size of the tree and get better precision by increasing this value, the greater is the risk of overfitting and longer training time. As minimum number of samples required to build any leaf in a tree, the value 10 is chosen.
The learning rate defines the step size used while training and it determines the velocity with which the learner converges on the optimal solution. If the value is too large, there is the risk to overshoot the optimal solution; if the step size is too small, training requires longer time to converge on the best solution.
The total number of decision trees constructed in the ensemble is 100. By creating more decision trees, one can possibly obtain better coverage, at the expenses of training time.
For Random number seed, a non-negative integer (1234) is typed in the textbox to ensure reproducibility across runs with same data and parameters.

As can be seen in ***Figure 41***, the results show improved overall performance metrics (precision and F1-score) and a better classification of "safe" class, while the "worn" class presents same misclassification rate/percentage as for the Decision Jungle algorithm.

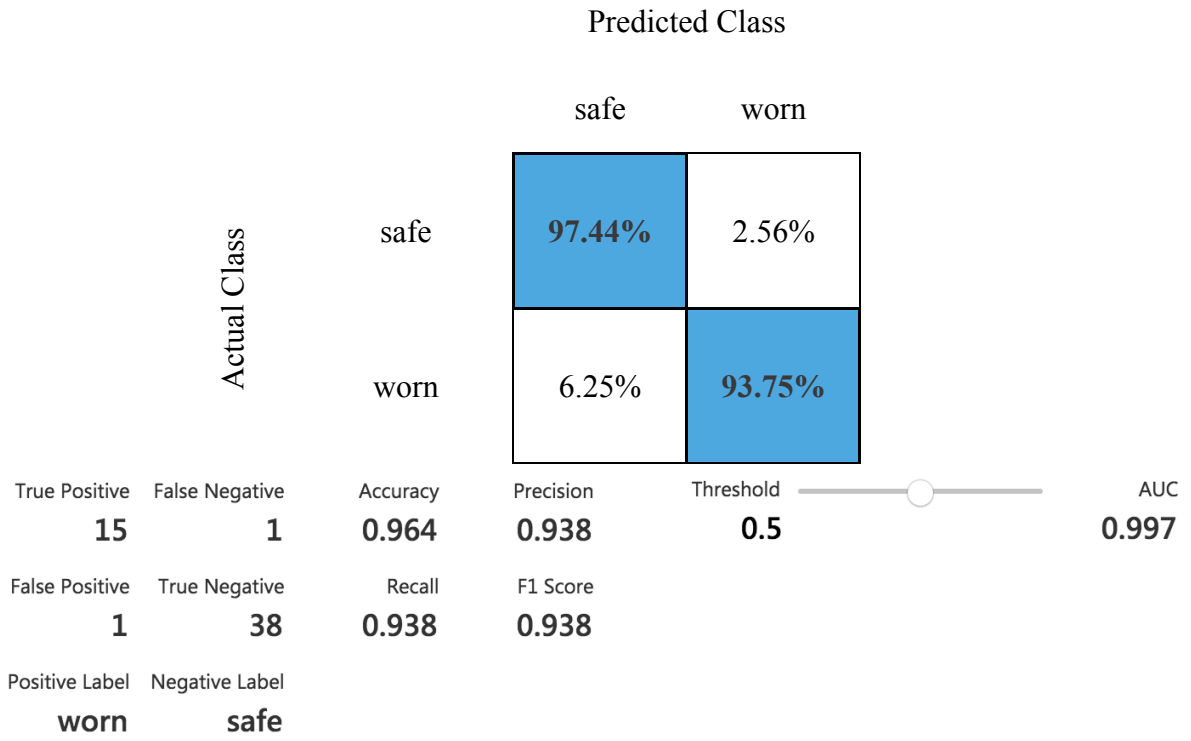


***Figure 41**: Confusion matrix and evaluation metrics for the Two-Class Boosted Decision Tree algorithm*

5.1.6 Two-class Neural Network

The last ML algorithm is the Two-Class Neural Network. A neural network is a set of interconnected layers, in which the inputs lead to outputs by a series of weighted edges and nodes. The weights on the edges are learned when training the neural network on the input data. The direction of the graph proceeds from the inputs through the multiple hidden layers, with all nodes of the graph connected by the weighted edges to nodes in the next layer.

Most predictive tasks can be accomplished easily with only one or a few hidden layers, although recent research has shown that deep neural networks (DNN) can be very effective for complex tasks (such as image or speech recognition), in which successive layers model the increasing levels of semantic depth. A neural network can be thought of as a weighted directed acyclic graph. The nodes of the graph are organized in layers, and they are connected by weighted edges to nodes in the next layer. To compute the output of the network for any given input, a value is calculated for each node in the hidden layers and in the output layer. This value is determined by computing the weighted sum of the values of the nodes in the previous layer and applying an activation function to this weighted sum [56]. Neural networks can be computationally expensive, due to a number of hyperparameters and the introduction of custom network topologies. Although in many cases neural networks produce better results than other algorithms, obtaining such results may involve fair amount of sweeping (iterations) over hyperparameters. But as for now, default single-parameter is evaluated.

The type of network architecture selected is the default fully-connected case, which has the following features:

- It has one hidden layer;
- The output layer is fully connected to the hidden layer, and the hidden layer is fully connected to the input layer.
- The number of nodes in the input layer equals the number of features in the training data.
- The number of nodes in the hidden layer is set by the user. The default value is 100.
- The number of nodes equals the number of classes. For a two-class neural network, this means that all inputs must map to one of two nodes in the output layer.

The learning rate defines the size of the step performed at each iteration. It is set to 0.1. A bigger value for this parameter could make the model to converge quicker, but at the same time it could overreach local minima. The Number of learning iterations specifies the maximum number of times the algorithm should process the training cases, while the initial weights diameter defines the node weights at the start of the learning process. The value utilized are respectively 100 and 0.1. For The momentum, i.e. the weight applied during learning to nodes from previous iterations, a zero value is chosen. The min-max normalization is applied to all features. Eventually, the shuffle examples option is unflagged so to process cases in exactly the same order each time the experiment is run.

The obtained results (***Figure 42***) are excellent: 100% of "worn" instances are correctly classified and only around 5% of "safe" instances are misclassified. These predictions would allow to forecast the exact time in which the cutting tool must be replaces and they would limit the loss of material arising from changing inserts which could be further exploited.

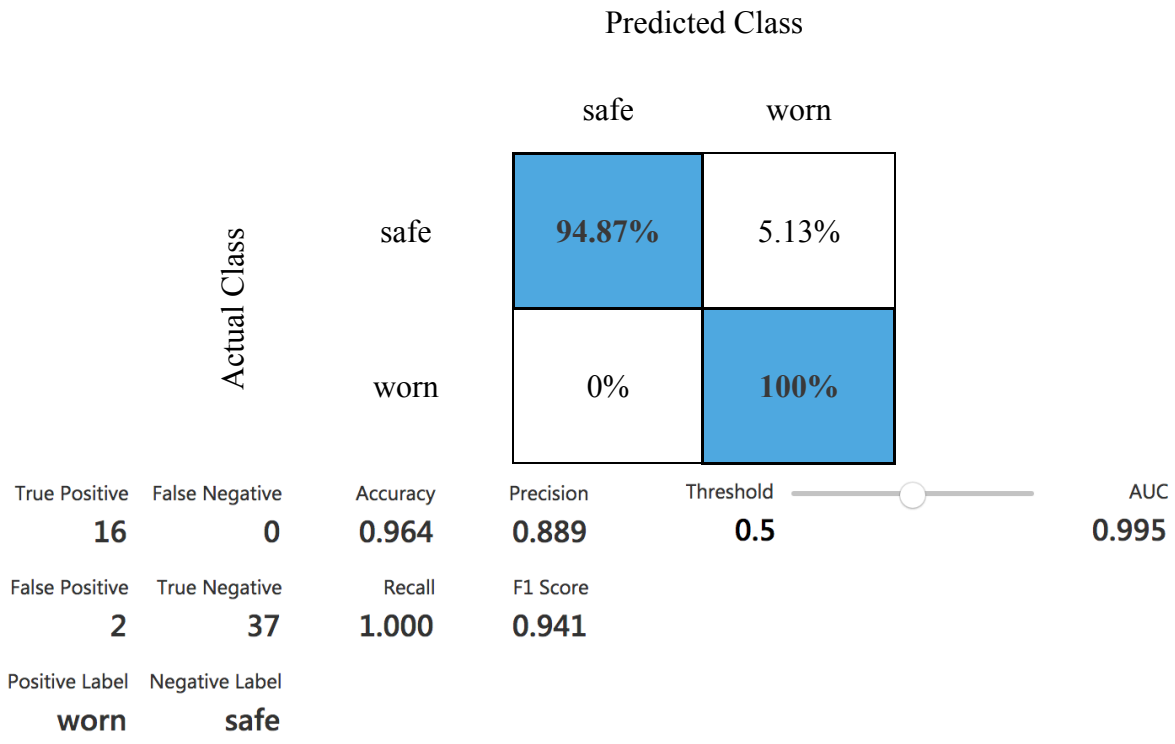


***Figure 42**: Confusion matrix and evaluation metrics for the Two-Class Neural Network algorithm*

This model seems to perform really well and to operate better than the other algorithms. However, the 100% value could be a sign of overfitting. The subsequent chapter will discuss how to improve the machine learning process by addressing this risk of overfitting through cross-validation techniques.

5.2 Regression Algorithms

For the Regression task, five machine learning algorithms are trained, tested and evaluated: Linear Regression, Boosted Decision Tree Regression, Decision Forest Regression, Neural Network Regression and Bayesian Linear Regression. ***Figure 43*** illustrates the model training and testing phase for the regression task in Azure MLS canvas.

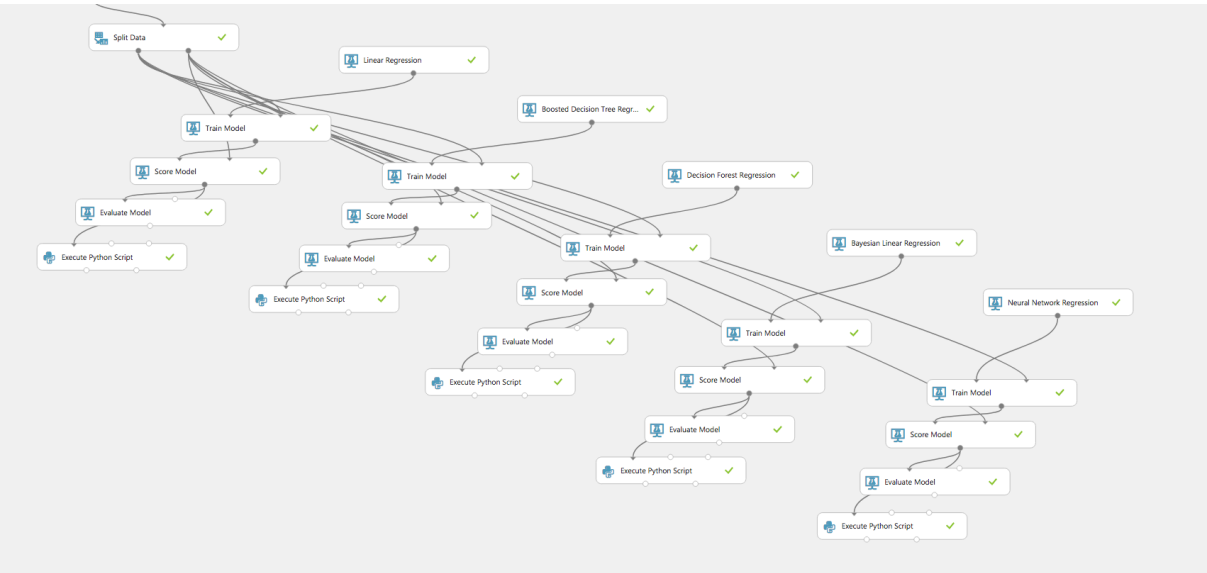


Figure 43*: Model training and testing phase for the Regression task in Azure MLS canvas*

As for the classification, the *Split Data* module splits randomly the dataset in 70% for training and 30% for testing. The resulting number of instances is respectively 102 and 43, lower than classification where new instances were created by the SMOTE algorithm.

Every ML algorithm uses the same modules:

- the *Train Model* module uses the input training set and trains a ML algorithm;
- the *Score Model* module creates predictions of flank wear (VB) for the testing data using the trained models;
- The *Evaluate Model* module assesses the performance of the model by returning evaluation metrics;
- The *Execute Python Script* plots the predicted value vs. actual value.

As for now, default parameters are chosen to configure the regression algorithms.

5.2.1 Evaluation metrics

The evaluation metrics used for regression models are: *Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), Relative Absolute Error, Relative Squared Error*, and the *Coefficient of Determination* (CoD, also known as R^2). These metrics are commonly designated to estimate the amount of error (or residual), which represents the difference between the predicted value and the observed/actual one. The absolute value or the square of this difference is generally computed to explain the total magnitude of error across all instances [50]. The error metrics measure the quality, i.e. predictive performance, of a regression model in terms of the mean deviation of its estimates from the real values. The lower the error values, the more the model is considered accurate in making predictions.

The **Mean absolute error (MAE)** measures the average sum of absolute residual (difference between the actual value and the predicted value) for all data points, without considering error directions. MAE defines how close the predictions are to the actual outcomes; thus, a lower score is better. It is defined by formula below.

$$MAE = \frac{1}{n}\sum_{i=1}^{n} |\widehat{y_i} - y_i|$$

Where *n* is the sample size, $\widehat{y_i}$ is the predicted value, and y_i is the actual value.

The **Root mean squared error (RMSE)** creates a single value which summarizes the overall error in the model. It is a measure of the standard deviation of the residuals and it is calculated as the square root of the average of squared differences between predicted and actual observation. By squaring this difference, the metric ignores the disparity between over-prediction and under-prediction. Since the errors are squared before they are averaged, the RMSE emphasizes the impact of large errors, thus it is preferred to MAE when huge errors are particularly undesirable.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n} (\widehat{y_i} - y_i)^2}$$

Where *n* is the sample size, $\widehat{y_i}$ is the predicted value, and y_i is the actual value.

Less used metrics are the **Relative absolute error (RAE)** and the **Relative squared error (RSE).** The RAE is the relative absolute difference between expected and actual values, where the term relative refers to the fact that the mean difference is divided by the arithmetic mean.

$$RAE = \frac{\sum_{i=1}^{n} |\widehat{y_i} - y_i|}{\sum_{i=1}^{n} |\bar{y} - y_i|}$$

Similarly, RSE normalizes the total squared error of the predicted values by dividing it by the total squared error of the actual values.

$$RSE = \sqrt{\frac{\sum_{i=1}^{n}(\hat{y}_i - y_i)^2}{\sum_{i=1}^{n}(\bar{y} - y_i)^2}}$$

The **Coefficient of determination (CoD)**, often referred to as $\boldsymbol{R^2}$, is a standard measure for representing the predictive power of the model. It can be interpreted as the proportion of variance in the dependent variables which is explained by the independent variable. This value describes how well the model fits/approximates the data points:

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}}$$

With:
$SS_{tot} = \sum_i (y_i - \bar{y})^2$ is the total sum of squares (proportional to variance);
$SS_{res} = \sum_i (\hat{y}_i - \bar{y})^2$ is the sum of squares of residuals.

Where y_i is the actual value, $\bar{y}$ is the mean of the observed data, and$\hat{y}_i$ is the predicted value. R-squared can assume a value between 0 and 1; the higher this number, the better is the model in forecasting the target parameter.

Although these metrics summarize well the overall performance of the models, a graphic tool is often required to visually represent the predictions vs actual values. To do so, the *Execute Python Script* module takes the scored dataset and selects and rename the two columns containing the actual VB and its estimate made by the model (‘*Scored Label’*). By visualizing the resulting dataset in Excel, it is possible to create/plot the trend of predicted VB vs. actual VB for the tested observations. The plot of actual vs predicted value shows the performance of the model in terms of how much the prediction deviates from observed value. To have a good fit, the two lines should be close to one another. As mentioned above, the testing dataset includes 43 instances, of which 8 have flank wear (VB) greater than the critical value 0.6. The plot is of particular interest for evaluating the performance of the model with respect to those cases.

5.2.2 Linear Regression

The first ML Regression Algorithm is the *Linear Regression*. Regression is a machine learning task used to predict a numeric outcome. Linear regression is a common statistical method for predictive analysis. It attempts to define a linear relationship between one or more independent variables and a scalar response (or dependent variable). The relationship is modelled by estimating model parameters/coefficients from the data and then, if additional values of the explanatory variables are gathered without a response value, it can be used to get response estimates. Although for years statisticians have been developing increasingly advanced methods for regression, linear regression is still a good way to implement a simple model for a basic predictive task [57].

The Linear Regression module in Azure MLS supports two methods to fit the regression line:

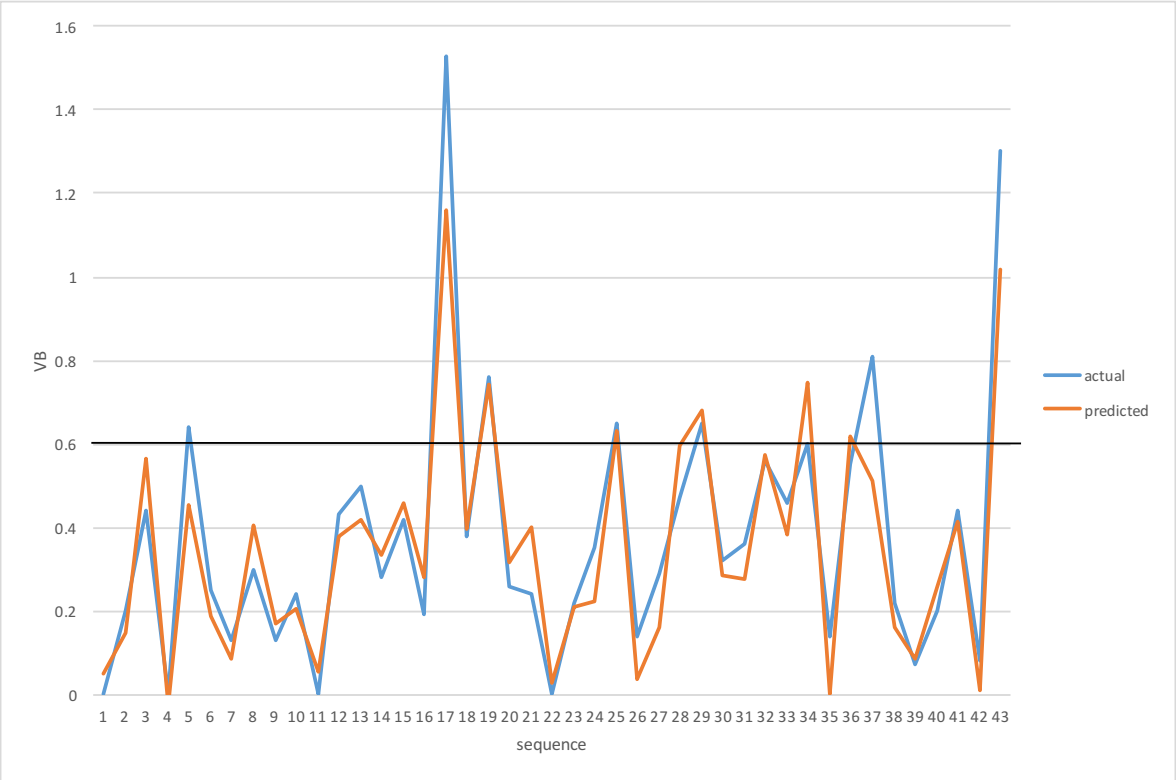
- Online gradient descent – Gradient descent is a method which minimises the amount of error at each step of the model training process. There are several types of gradient

descent algorithms. This option allows the setting of multiple parameters to control/configure the learning rate, L2 regularization and others.

- Ordinary Least Squares (OLS) – Least squares linear regression is one of the most commonly used method for linear models. This method determines the parameters of the regression line by minimizing the total squared error, defined as the sum of the squares of the difference between the observed/actual predicted value of the dependent variable.

In this first model deployment, the OLS linear model is applied, with default regularization L2 equal to 0.001 and visualization of the term for the intercept.

Results are represented in ***Figure 44***. The linear regression fits satisfactorily the data, with a 0.86 R^2 and a MAE of 0.085. The graph also shows the 6 out of 8 instances where VB>=0.6 are correctly identified as over this limit, while 2 results would make the milling operator to use inserts more than their wear limits. In only one circumstances, a still safe cutting tool is confused with a dangerous one.



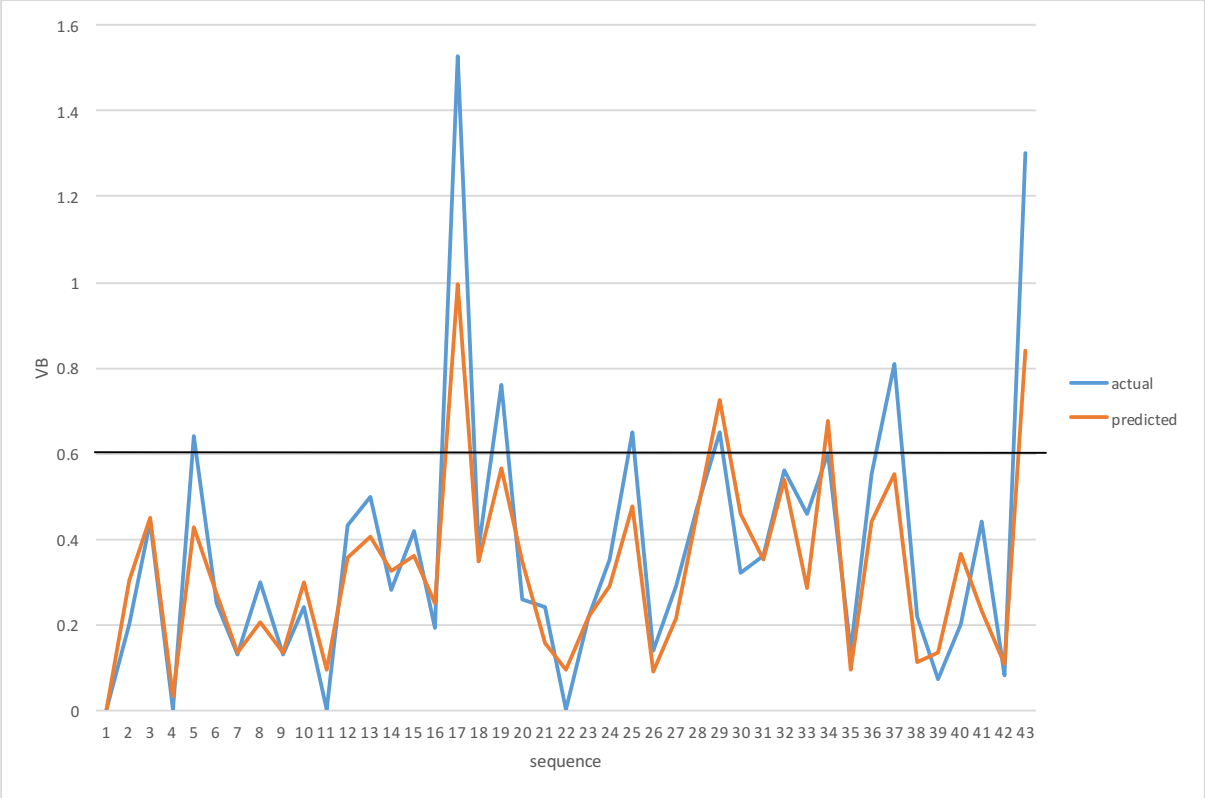
Mean Absolute Error	**0.085131**
Root Mean Squared Error	**0.115057**
Relative Absolute Error	**0.380632**
Relative Squared Error	**0.138268**
Coefficient of Determination	**0.861732**

Figure 44*: Linear Regression model output. On the top, Predicted VB (red) vs actual VB (blue). On the bottom, overall performance of the regression by standard metrics.*

5.2.3 Boosted Decision Tree Regression

The second ML regression algorithm is the *Boosted Decision Tree Regression*. The Boosted Decision Tree Regression module creates an ensemble of regression trees using boosting. In Azure MLS, boosted decision trees use an implementation of the MART (Multiple Additive Regression Trees) gradient boosting algorithm. This algorithm is a machine learning technique for regression problems which creates a series of trees in an iterative way and then identifies the optimal tree by means of an arbitrary differentiable loss function [58].

In ***Figure 45***, the metrics show that the model behaves quite well but not as good as linear regression (0.77 CoD and 0.10 MAE). The graph confirms the metrics, but the predictions are slightly under-estimating VB, which could be an expensive estimation for the reasons previously illustrated.



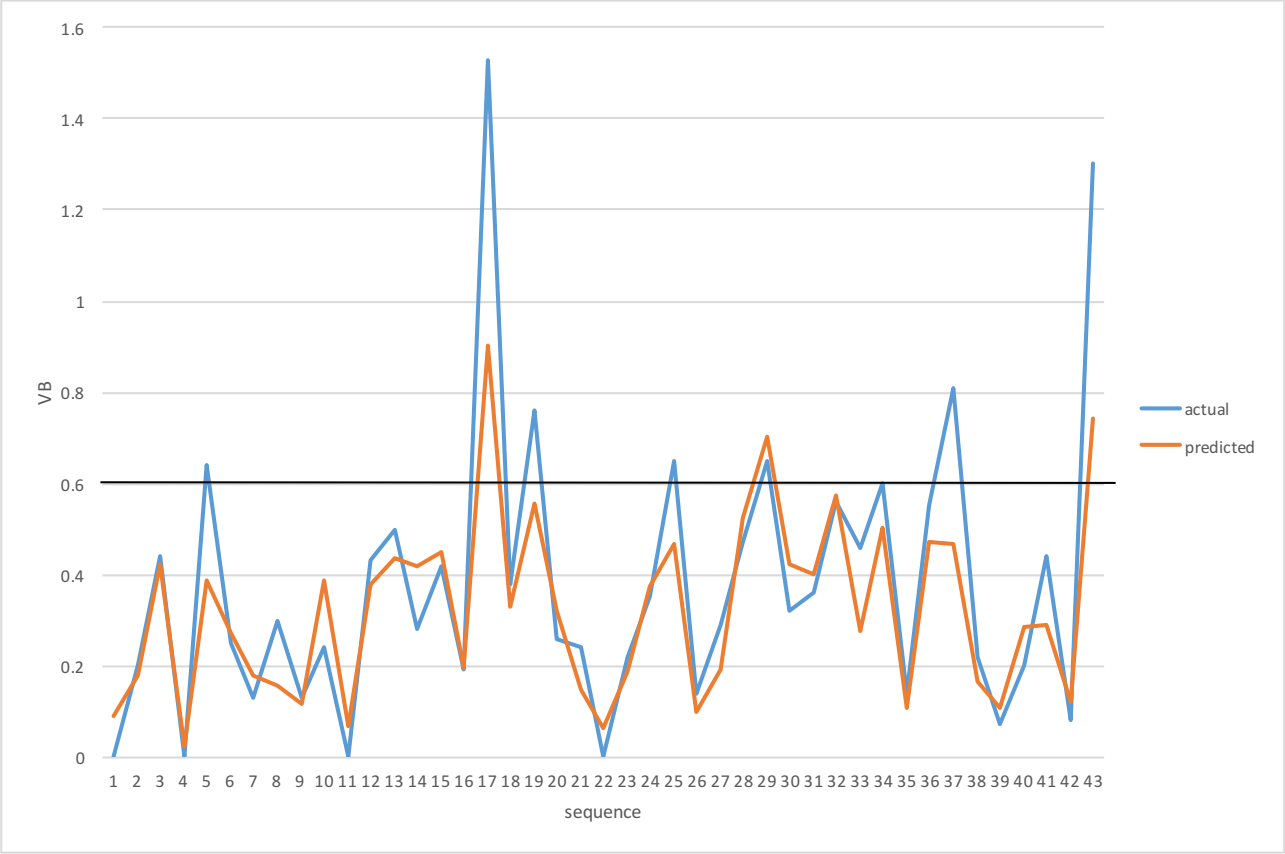
Mean Absolute Error	**0.100308**
Root Mean Squared Error	**0.14753**
Relative Absolute Error	**0.448494**
Relative Squared Error	**0.227328**
Coefficient of Determination	**0.772672**

Figure 45*: Boosted Decision Tree Regression model output. On the top, Predicted VB (red) vs actual VB (blue). On the bottom, overall performance of the regression by standard metrics*

5.2.4 Decision Forest Regression

The third ML regression algorithm is the *Decision Forest Regression*, which operates in a similar manner as for the classification task.

Evaluation metrics indicates a worse performance compared to the Boosted Decision Tree Regression model. The MAE is still low, but the R^2 decreases to 0.71. From **Figure 46**, it can be seen that the number of data points under-estimated increases: for 5 instances with observed VB>0.6, the tool is predicted as safe when it actually should be substituted.



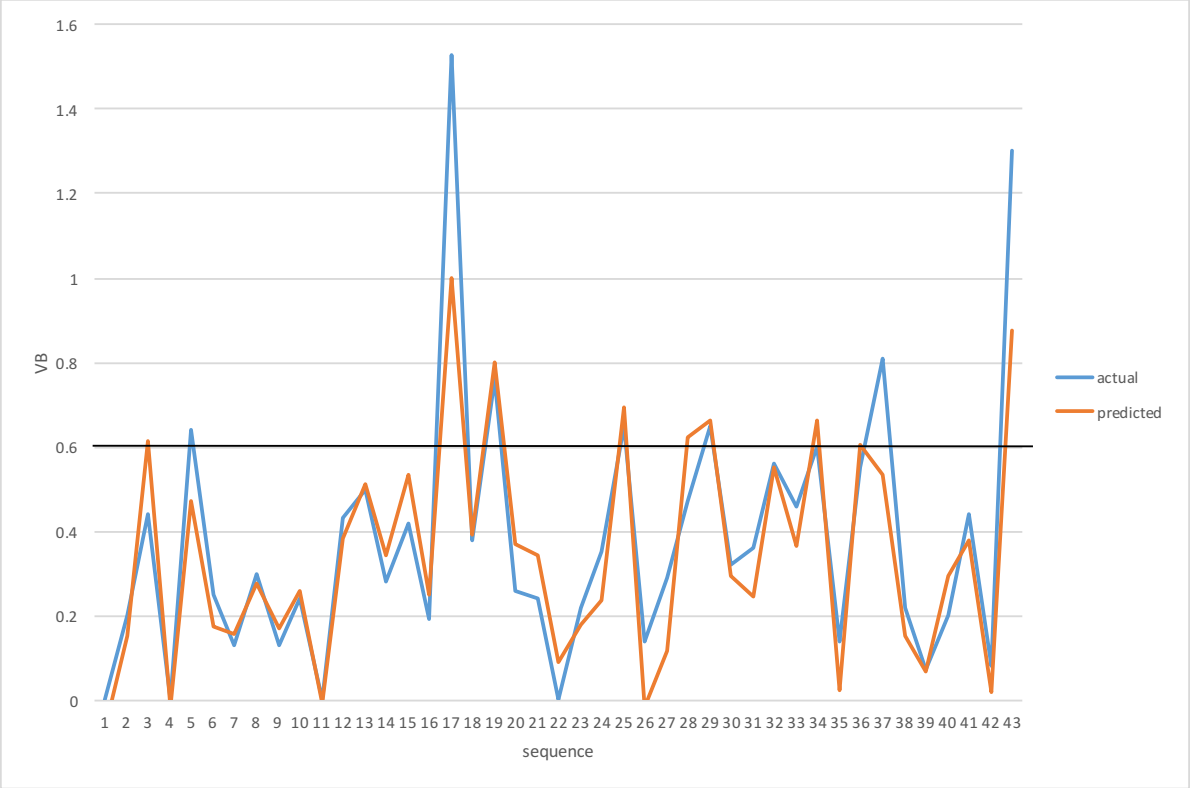
Mean Absolute Error	**0.105895**
Root Mean Squared Error	**0.165831**
Relative Absolute Error	**0.473472**
Relative Squared Error	**0.287227**
Coefficient of Determination	**0.712773**

***Figure 46**: Decision Forest Regression model output. On the top, Predicted VB (red) vs actual VB (blue). On the bottom, overall performance of the regression by standard metrics*

5.2.5 Bayesian Linear Regression

The fourth ML regression algorithm is the *Bayesian Linear Regression*. The Bayesian method formulates a linear regression line by combining a prior probability distribution about parameters with a likelihood function to generate estimates for the model [59]. The dependent variable is not estimated as a single value but it is assumed to be predicted form a probability distribution. Bayesian linear regression is especially suitable for insufficient or poor distributed data.

Results (***Figure 47***) show that Bayesian Linear Regression works pretty fine, fitting more than 78% of data (CoD) and with a low MAE of 0.093. The performance is decent and it is confirmed by the plot, showing that the predictions are close to the real value.



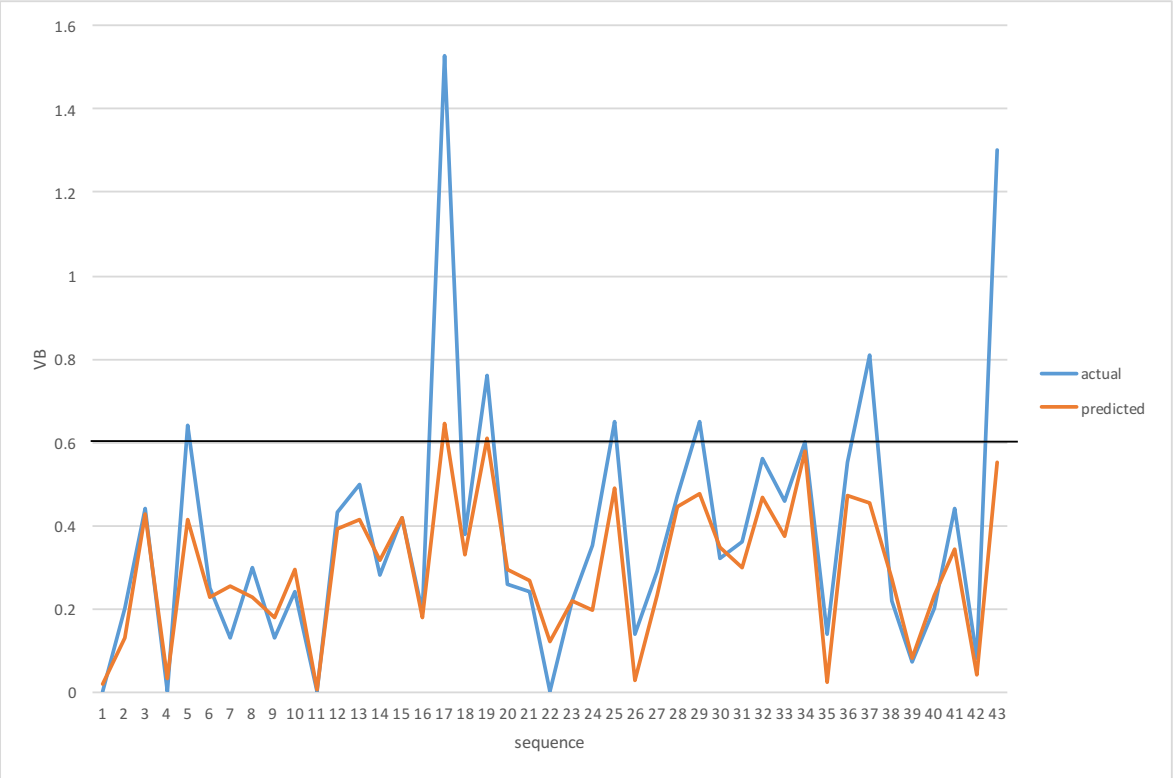
Mean Absolute Error	**0.093346**
Root Mean Squared Error	**0.138829**
Relative Absolute Error	**0.417365**
Relative Squared Error	**0.201305**
Coefficient of Determination	**0.798695**

Figure 47*: Bayesian Linear Regression model output. On the top, Predicted VB (red) vs actual VB (blue). On the bottom, overall performance of the regression by standard metrics*

5.2.6 Neural Network Regression

The last ML Regression algorithm is the *Neural Network Regression*. Although neural networks are principally used for modelling complex problems, they are also adapted to regression problems. Any statistical model can be represented as a neural network by using adaptive weights and can approximate non-linear functions of the input variables. The output layer consists of one node only and it identifies the dependent variable. Neural network regression is suited to problems where a more traditional regression model cannot fit a solution [60].

The results are summarized in ***Figure 48***. The evaluation metrics are not exceptional: while the MAE is still small, the R^2 is just 0.57, meaning that the proportion of variance explained by the model is lower than 60%, the worst result obtained so far. The graph also indicates that most of the data points with an actual VB<0.6 are not detected as critical, since the model tend to underestimate the flank wear.



Mean Absolute Error	**0.107618**
Root Mean Squared Error	**0.202155**
Relative Absolute Error	**0.481175**
Relative Squared Error	**0.426836**
Coefficient of Determination	**0.573164**

Figure 48*: Neural Network Regression model output. On the top, Predicted VB (red) vs actual VB (blue). On the bottom, overall performance of the regression by standard metrics*

6. Model Improvement

Model Improvement has the objective of boosting the performance of machine learning algorithms. In the first implementation of training and testing phase:

- for classification task, the best performance is given by *Two-class Neural Network* algorithm;
- for regression task, the most performant ML algorithm is the *Linear Regression*.

Although the achieved results are respectable, two main problems are associated with the discussed scenarios: manual configuration of model's parameters and risk of overfitting.

The ML models implemented until now use only single parameter's default settings. When training a machine learning algorithm, it is very important that the right set of parameters is chosen. How ca we be sure that the set of chosen parameters is the optimal one for our algorithm? This issue is addressed in this Section by means of a Hyperparameter Tuning method.

The second limitation derives from the way in which the evaluation of model's performance is carried out. The dataset was split in two subsets: 70% of instances were involved in training the algorithms, while 30% in testing and evaluating the predictions. Given the limited size of our dataset, so far nothing can be said about generalization: how well the model will behave in front of new data? Is the model overfitting? To address these questions, k-Fold Cross Validation technique is discussed in the current Chapter.

6.1 Hyperparameter tuning

Machine learning algorithms require to configure parameters before you train the model. These parameters whose value must be specified before the learning process begins are referred as hyperparameters. By thinking at the previous implementation, they are simply the measures defined in the configuration panel for each ML algorithm. Hyperparameters significantly affect the performance of the model, thus choosing the correct values is essential. Usually, one doesn't know in advance which is the optimal configuration for a given model and therefore should evaluate all different combinations. This process of finding the optimal set of hyperparameters for a learning algorithm is known as hyperparameter optimization or tuning. When a ML algorithm is tuned for a given problem, the goal is to figure out those parameters that result in the most accurate predictions. Most of the machine learning software are able to perform hyperparameter tuning by automatically exploring and selecting the best model configuration for a specific ML algorithm.

Microsoft Azure Machine Learning Studio provides the so called *Tune Model Hyperparameters* module to accomplish hyperparameters tuning [61]. The module takes as input the training dataset and an untrained model with a range of possible values for all hyperparameters and it executes an iterative process in which it constructs and tests multiple parameter combinations and compares evaluation metrics over all results in order to determine the optimal configuration of settings for a given ML algorithm. For doing so, the *Create trainer mode* option of the ML properties panel for the learner must be set to *Parameter Range* and several values must be specified in the textbox for each hyperparameter. Almost all the classification and regression modules support an integrated parameter sweep.

The Tune Model Hyperparamets supports different methods for parameter sweeping:

- *Entire grid* allows to test all possible combination of values; this option is the most exhaustive but time-consuming;
- *Random grid* calculates a matrix of all parameter combinations and tests some random set of these, over a limited number of iterations;
- *Random sweep* uses a subset of parameter values randomly selected from the specified range; one must define a maximum number of runs executed by the module.

Eventually, the module requires the specification of the evaluation criteria to be used for measuring the performance and choose the best model /judge/rank the model. The adopted metric varies depending on the purpose of the problem and the cost of false positives and false negatives. A sample panel for the configuration of the tuning process is illustrated in **Figure 49** below.

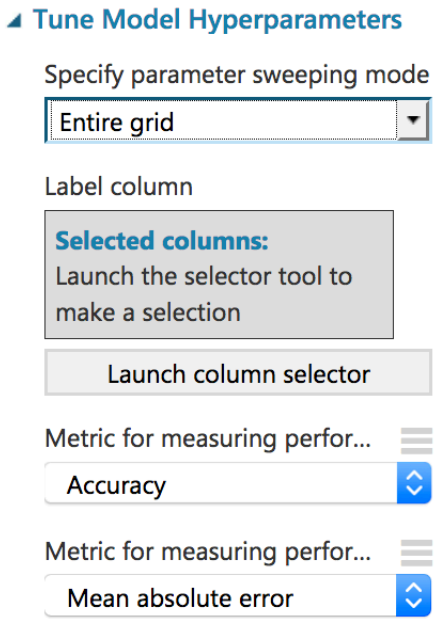


***Figure 49**: Configuration panel for the Tune Model Hyperparameters in Azure MLS*

6.2 k-Fold Cross-validation

The main goal of any machine learning model is to learn from samples in such a way that the model is able to generalize the learning to unseen (real) data that it has never seen before, that is to accurately perform in practice when making predictions on new data from a real problem. When evaluating the performance of a model, it is always necessary to assesses how well the learner will generalize to an unseen dataset. This process of deciding whether the model generalizes to unseen data is known as validation.

A basic manner to do so is the holdout method previously implemented, which involves using a random subset of total dataset to train the model (the so called training set), and the remaining data to make predictions based on the trained model and evaluate the performance (known as test or validation set). This method is easy to implement but it has some significant limitations, especially for small dataset as ours. Holdout method suffers from problems of high variance

while testing the effectiveness of the model, since it is not certain which data points will end up in the validation set and the result might be entirely different for different sets [62]. Additionally, by reducing the training data, there is a risk of overfitting or under fitting, which in turn increases error induced by bias: we need to verify if the model recognises most of the patterns from the data correctly, and it's not picking up too much on the noise, or in other words its low on bias and variance [63].

Therefore, there is the need for a method providing ample data for training but also leaving large amount data for validation. K-fold cross-validation is applied exactly for this purpose. It is an important technique often used in machine learning to assess both the variability of a dataset and the reliability of any model trained using that data. In k-fold cross-validation, the entire dataset is randomly partitioned into k groups of approximately equal size. One at a time, a single partition is kept/treated as validation set for testing the model and the remaining k-1 subsamples are put together and used as training data/ the model is fit on the remaining k-1 folds. For instance, considering 5 folds, the module would generate five models during cross-validation, each model trained using 4/5 of the data, and tested on the remaining 1/5 (**Figure 50**). Hence, the process is repeated k times and the evaluation results are computed for each validation. The k results can be averaged to obtain a single performance metric of the total effectiveness of the model.

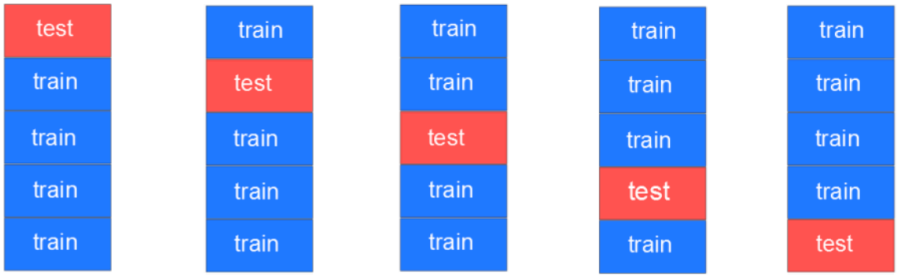


***Figure 50**: How cross-validation works with 5 folds*

The procedure requires the definition of the parameter k, which that refers to the number of folds which the given dataset is split into. This value must be chosen carefully for each data sample. A poorly chosen value for k may result in a misrepresentative idea of the skill of the model, such as a score with a high variance (that may change a lot based on the data used to fit the model), or a high bias, (such as an overestimate of the skill of the model) [64]. As a general rule and empirical evidence, K = 5 or 10 is generally preferred, as these values have been shown empirically to yield test error rate estimates that suffer neither from excessively high bias nor from very high variance, but nothing's fixed and it can take any value.

Cross-validation leads to multiple advantages:

- It uses all observations for both training and evaluation, instead of some portion. In contrast, if you validate a model by using data generated from a random split, typically you evaluate the model only on 30% or less of the available data;
- It significantly reduces both bias and variance, increasing the performance of the model;
- It does not simply measure the accuracy of a model, but also gives some idea of how representative the dataset is and how sensitive the model might be to variations in the data.

In summary, cross-validation combines measures of fitness in prediction to derive a more accurate estimate of model prediction performance. However, this method trains and validates the model several times over a larger dataset, it is much more computationally intensive and takes much longer than validating on a random split.

Cross-validation is implemented in Azure MLS by the *Cross-Validate Model* module together with the *Partition and Sample* module. First, the Partition and Sample module randomly divides the dataset into k folds of the same size [65]. In the configuration settings (***Figure 51***), one must indicate the partition mode, the custom number of folds and the proportion of rows in each subset. By deselecting the Use replacement in the partitioning option, each observation in the dataset is assigned to only one group.

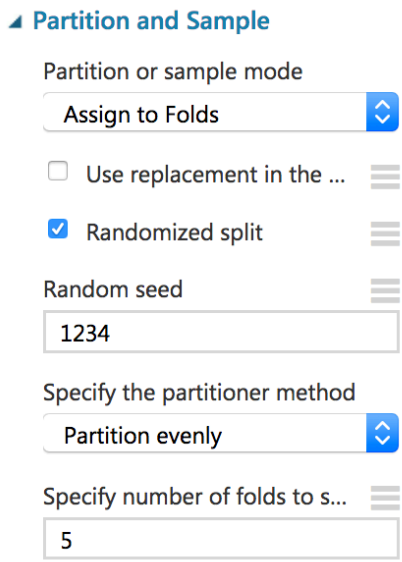


Figure 51*: Configuration panel for the Partition and Sample module in Azure MLS*

The Cross-Validate Model module takes as input the partitioned dataset and an untrained classification or regression model and it builds a model on each fold [66]. During testing of the model, the module returns multiple accuracy statistics for each subset. By comparing those values, we can interpret the quality of the dataset and understand whether the model is susceptible to variations in the data: if the metrics are consistent across all set of folds, which means results have low variation, then the model generalize well. When the building and evaluation process is completed for all folds, Cross-Validate Model generates a set of overall performance/evaluation metrics and scores results for all data, so that the reliability of the predictions can be assessed. In conclusion, the best result corresponds to a good and consistent set of metrics, which represents a performant and general model.

6.3 Model Improvement – Classification

In this section, model improvement techniques are applied to all the classification tasks presented in ***Section 5.1***. A combination of Tune Model Hyperparameters, Partition and Sample and Cross-Validate Model modules is employed to find out the best results.

The procedure will be the same for all model, and described as follows:

1. From the data prepared in ***Section 4***, the data flow will follow two paths: the first one will lead to the input node of a Tune Model Hyperparameters module, the second will be directed to a Partition and Sample module.
2. The Tune Model Hyperparameters will take an untrained ML algorithm and will perform a parameter sweep, set on Entire Grid mode, and optimizing **Accuracy metric**. The untrained model uses the Parameter Range as trainer mode (previously set as Single Parameter). Parameter Range specifies a range of model parameters to test the model, thus the parameter sweeping module will choose the most accurate model within the specified range.
3. The Partition and Sample module will divide the dataset into **5 Folds** for the Cross-validate Model Module.
4. The Cross-validate Model module will take the best model from the Tune Model Hyperparameters and the dataset divided in folds by the Partition and Sample module and will perform a cross validation training.
5. The results of the cross validation will be shown either on Evaluation results by fold output of the Cross-validate Model module (metrics for each test fold) and on the Evaluate Model module (confusion matrix).

6.3.1 Two-class Decision Forest

The first ML algorithm is the *Two-class Decision Forest*. The configuration flow in Azure MLS and the setting of the parameters range are shown in **Figure 52**.

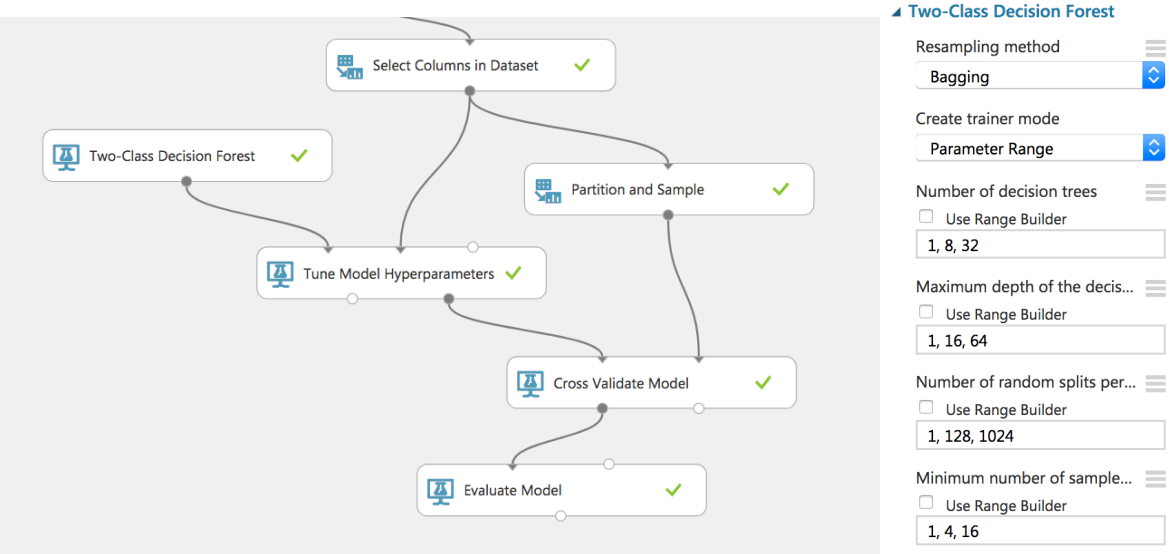


Figure 52*: Improved Two-Class Decision Forest (left) and the algorithm parameters settings (right)*

The confusion matrix of the model (***Figure 53***) is improved compared to the model in ***Section 5.1.1***, in particular in relation to the classification of the "worn" class. The evaluation of the Cross-validate Model module by folds is shown in **Table 7**. The data are not so consistent, especially as regards the recall metric, which indicates the percentage of worn inserts correctly classified. Thus, the model does not generalize well and it susceptible to variations in new data.

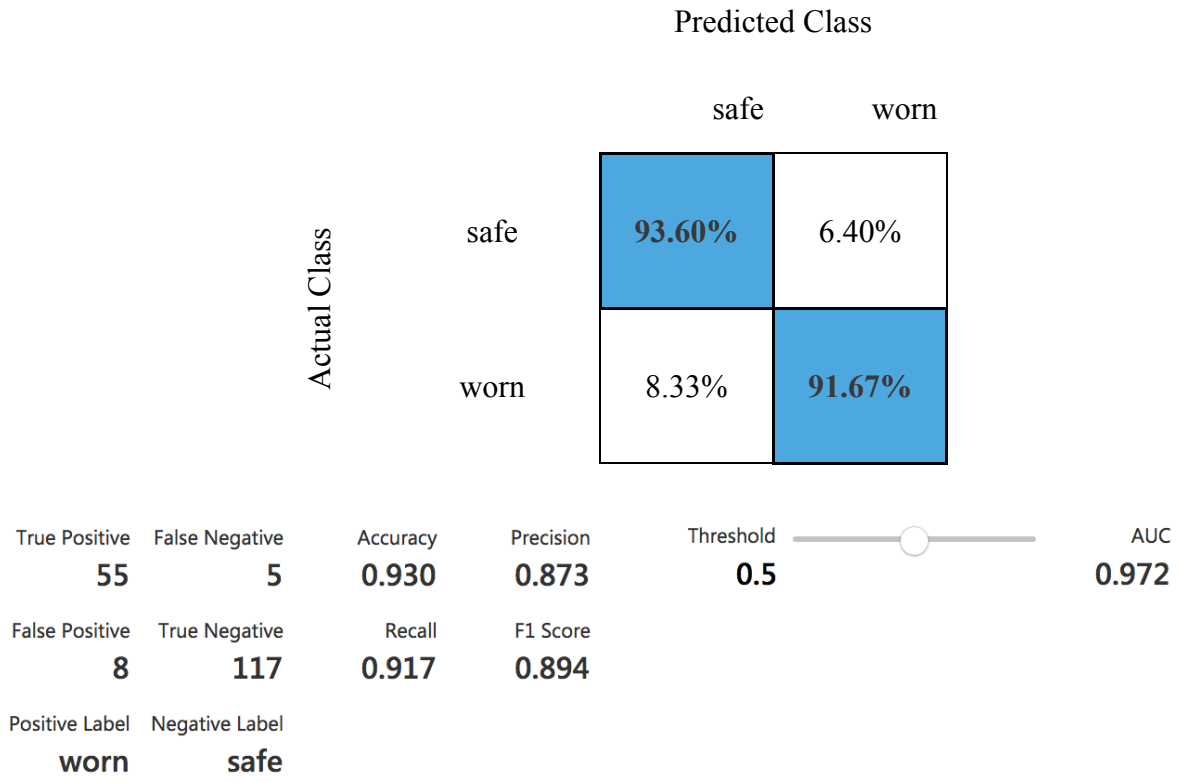


***Figure 53**: Improved Two-Class Decision Forest's confusion matrix and overall evaluation by standard metrics*

Fold Number	Number of examples in fold	Model	Accuracy	Precision	Recall	F-Score	AUC	Average Log Loss	Training Log Loss
0	37	Two-class Decision Forest	1	1	1	1	1	0.156614	74.264677
1	37	Two-class Decision Forest	0.891892	1	0.714286	0.833333	0.992236	0.195605	70.508752
2	37	Two-class Decision Forest	0.945946	0.833333	1	0.909091	0.968519	0.260335	55.38598
3	37	Two-class Decision Forest	0.864865	0.615385	1	0.761905	1	0.256788	50.81408
4	37	Two-class Decision Forest	0.945946	0.941176	0.941176	0.941176	0.977941	0.24735	64.144749
Mean	185	Two-class Decision Forest	0.92973	0.877979	0.931092	0.889101	0.987739	0.223338	63.023648
Standard Deviation	185	Two-class Decision Forest	0.052685	0.161803	0.123846	0.093138	0.014021	0.045512	9.887516

***Table** 7: Improved Two-Class Decision Forest evaluation by folds of the Cross-validate Model*

6.3.2 Two-class Logistic Regression

The second ML algorithm is the *Two-class Logistic Regression* (***Figure 54***).

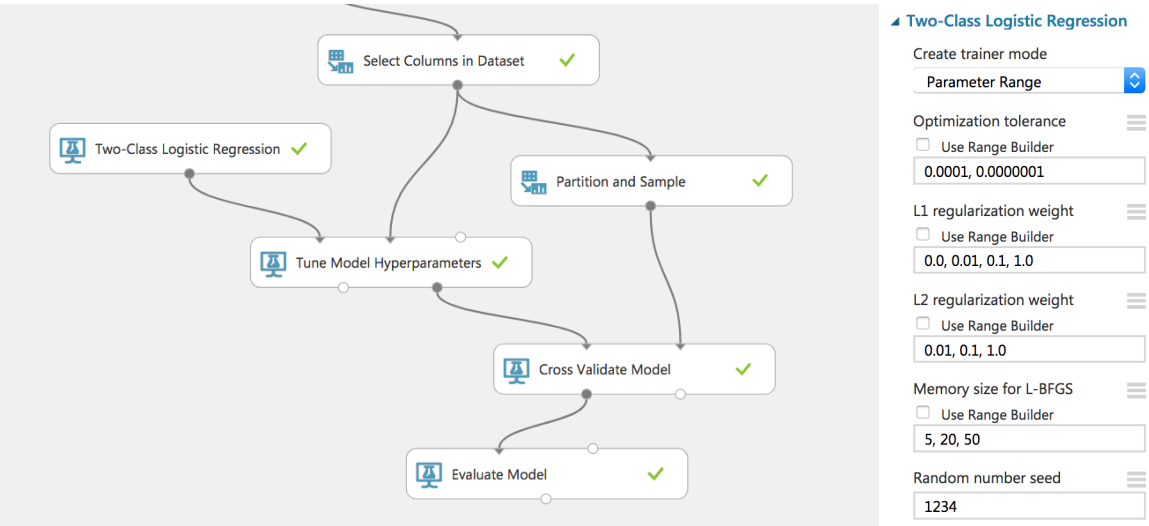


Figure 54*: Improved Two-Class Logistic Regression (left) and the algorithm parameters settings (right)*

The model is improved in accuracy with respect to the one in ***Section 5.1.3*** as shown in ***Figure 55***. Although the misclassification of the "worn" class is slightly reduced (from 18% to 10%), the standard deviation of the metrics of each fold is not sufficiently low with respect to the mean (***Table 8***).

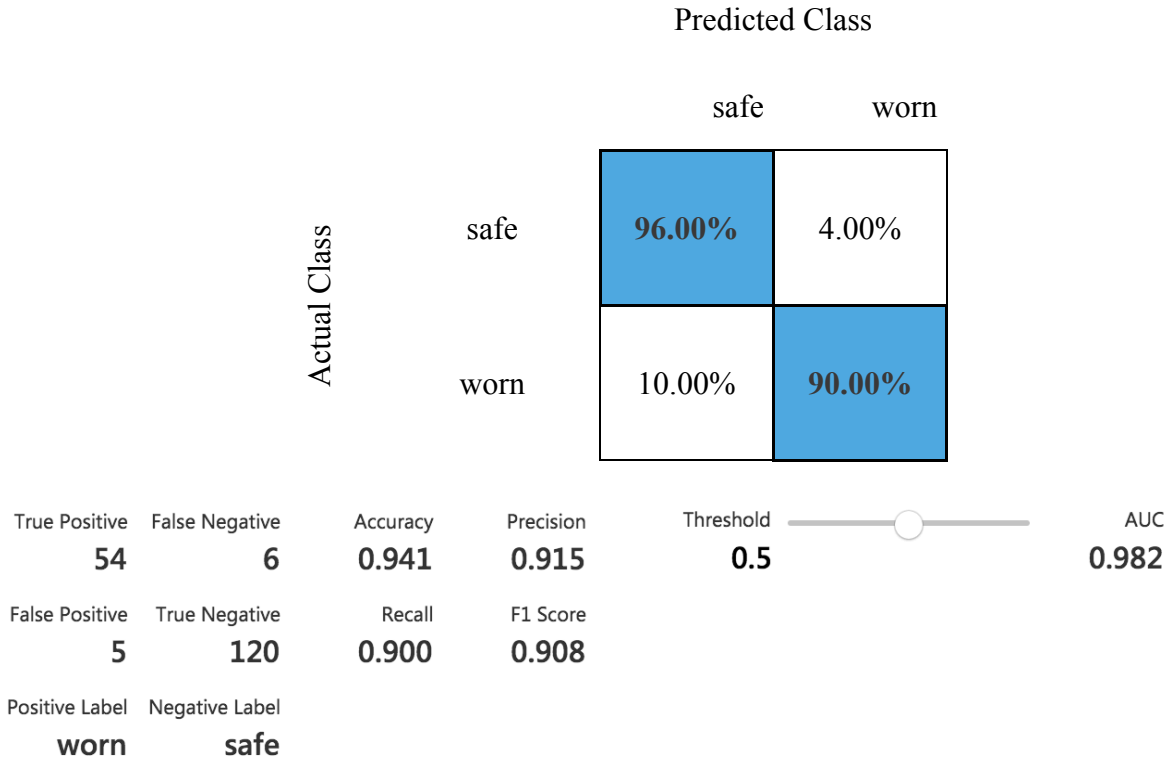


Figure 55*: Improved Two-Class Decision Forest's confusion matrix and overall evaluation by standard metrics*

Fold Number	Number of examples in fold	Model	Accuracy	Precision	Recall	F-Score	AUC	Average Log Loss	Training Log Loss
0	37	Two-class Logistic Regression	1	1	1	1	1	0.023201	96.187474
1	37	Two-class Logistic Regression	1	1	1	1	1	0.092174	86.103028
2	37	Two-class Logistic Regression	0.918919	0.818182	0.9	0.857143	0.966667	0.293283	49.739703
3	37	Two-class Logistic Regression	0.918919	0.727273	1	0.842105	0.982759	0.19116	63.384735
4	37	Two-class Logistic Regression	0.864865	1	0.705882	0.827586	1	0.224881	67.401716
Mean	185	Two-class Logistic Regression	0.940541	0.909091	0.921176	0.905367	0.989885	0.16494	72.563331
Standard Deviation	185	Two-class Logistic Regression	0.058593	0.128565	0.127906	0.087018	0.014973	0.107398	18.528562

***Table 8**: Improved Two-Class Logistic Regression evaluation by folds of the Cross-validate Model*

6.3.3 Two-class Decision Jungle

The third ML algorithm subject to improvement is the *Two-class Decision Jungle* (***Figure 56***).

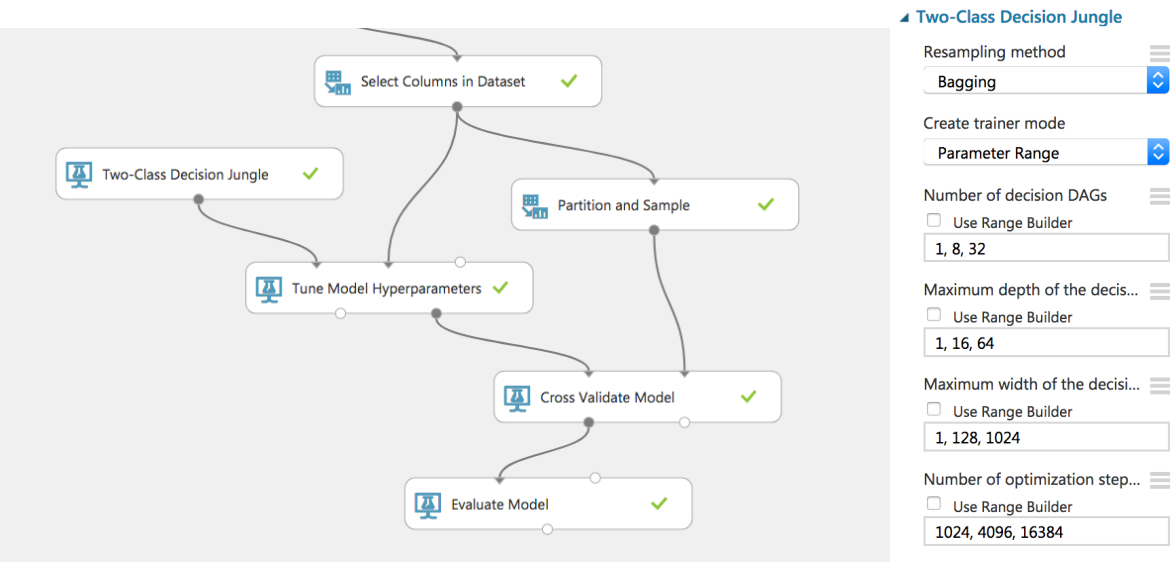


***Figure 56**: Improved Two-Class Decision Jungle (left) and the algorithm parameters settings (right)*

As illustrated in **Figure 57**, the confusion matrix approximately confirms the accuracy and classification rate of the model in ***Section 5.1.4***. In ***Table 9***, the output of the Cross-validate Model's evaluation by folds is reported. As can be seen, the model generalizes better than the previous algorithms.

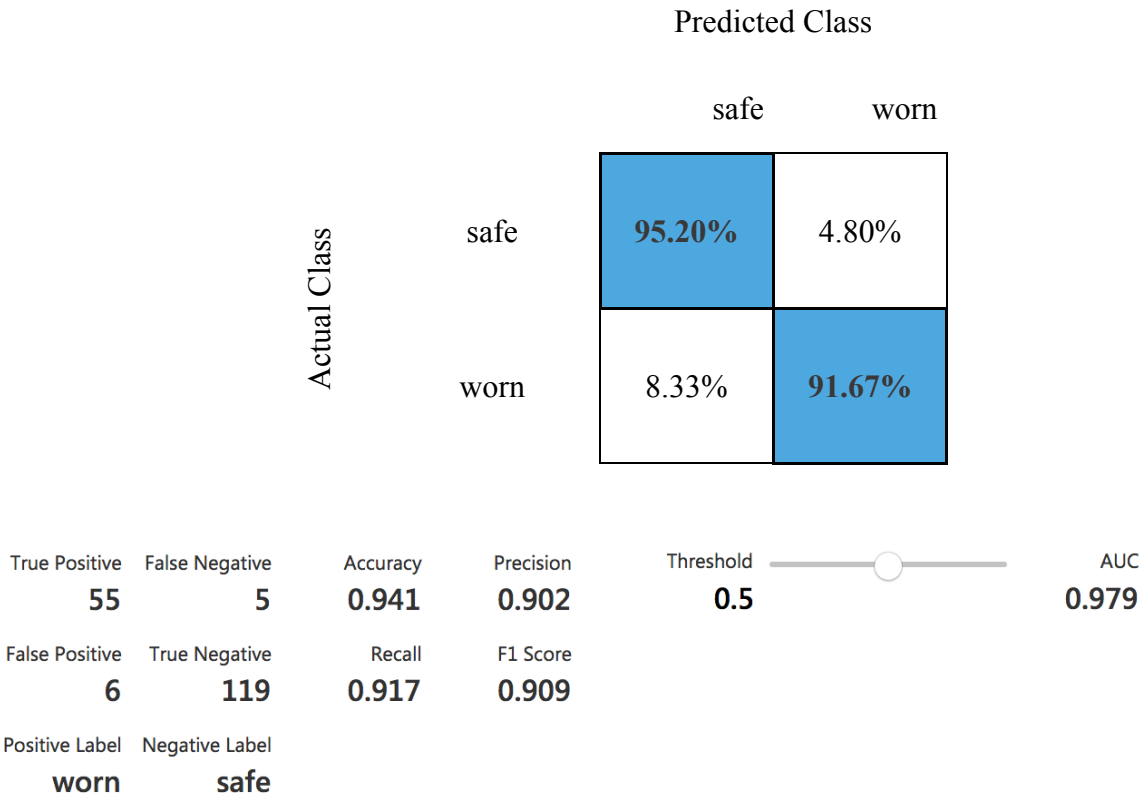


***Figure 57**: Improved Two-Class Decision Jungle's confusion matrix and overall evaluation by standard metrics*

Fold Number	Number of examples in fold	Model	Accuracy	Precision	Recall	F-Score	AUC	Average Log Loss	Training Log Loss
0	37	Two-class Decision Jungle	0.945946	0.857143	1	0.923077	0.99	0.202173	67.913463
1	37	Two-class Decision Jungle	0.945946	0.857143	1	0.923077	0.983333	0.197525	68.651156
2	37	Two-class Decision Jungle	0.918919	0.909091	0.833333	0.869565	0.985	0.201367	68.041388
3	37	Two-class Decision Jungle	0.918919	0.909091	0.833333	0.869565	0.965	0.228805	63.686767
4	37	Two-class Decision Jungle	0.972973	1	0.916667	0.956522	0.995	0.164436	73.902661
Mean	185	Two-class Decision Jungle	0.940541	0.906494	0.916667	0.908361	0.983667	0.198861	68.439087
Standard Deviation	185	Two-class Decision Jungle	0.022612	0.058369	0.083333	0.037957	0.01139	0.022917	3.637121

***Table 9**: Improved Two-Class Decision Jungle evaluation by folds of the Cross- validate Model*

6.3.4 Two-class Boosted Decision Tree

The fourth ML algorithm is the *Two-class Boosted Decision Tree* (***Figure 58***).

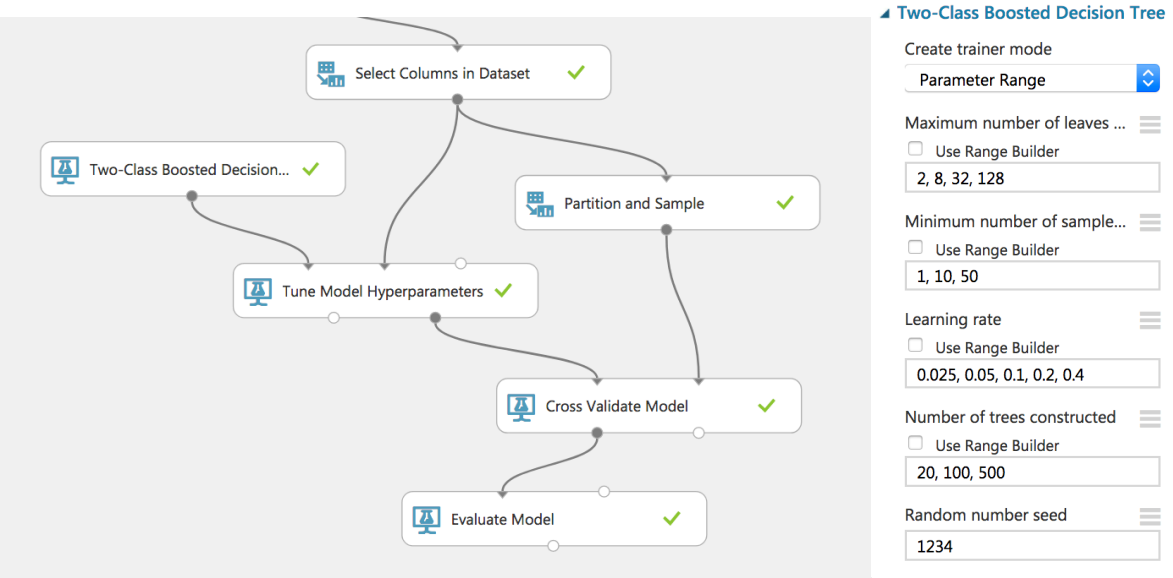


***Figure 58**: Improved Two-Class Boosted Decision Tree (left) and the algorithm parameters settings (right)*

The confusion matrix shows excellent results, confirming the already good performance achieved without improvement (***Figure 59***). At the same time, the evaluation metrics computer for each fold (***Table 10***) look consistent with a low variance. Thus, we can conclude that the model is also able to generalize on unseen data.

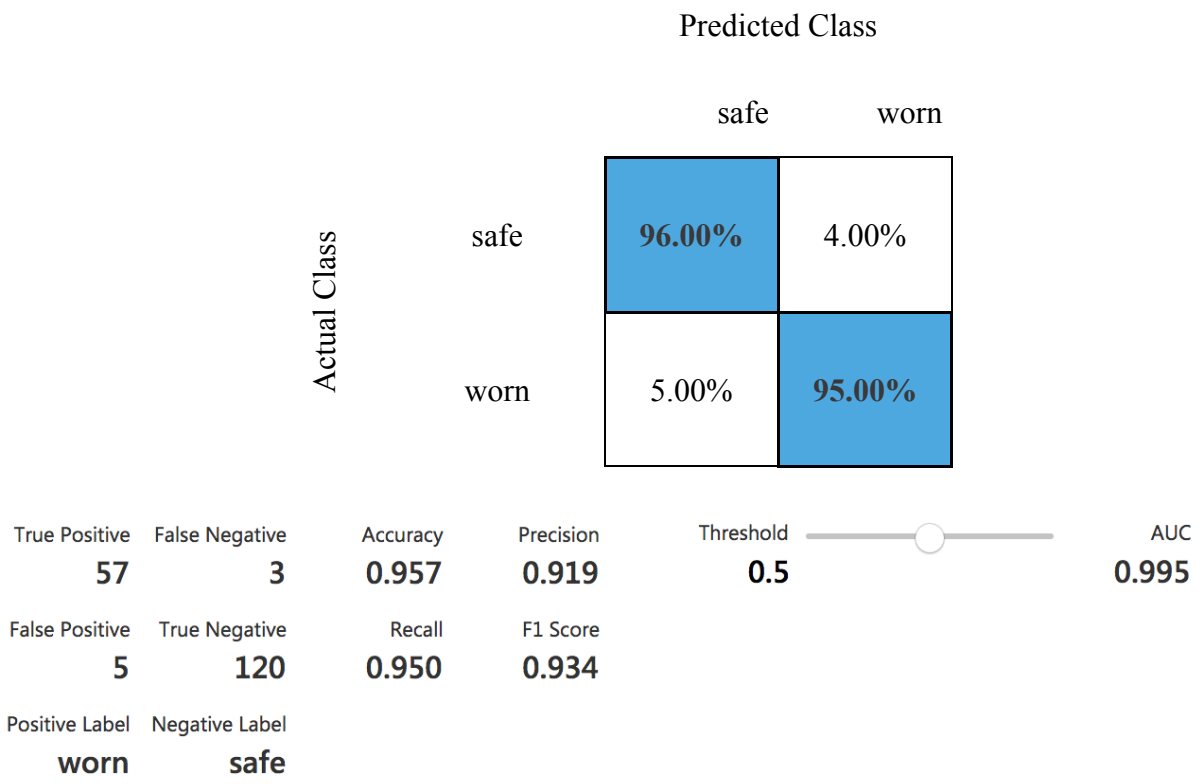


***Figure 59**: Improved Two-class Boosted Decision Tree's confusion matrix and overall evaluation by standard metrics*

Fold Number	Number of examples in fold	Model	Accuracy	Precision	Recall	F-Score	AUC	Average Log Loss	Training Log Loss
0	37	Two-class Boosted Decision Tree	0.972973	0.916667	1	0.956522	1	0.049631	91.844472
1	37	Two-class Boosted Decision Tree	0.972973	1	0.928571	0.962963	1	0.085212	87.152621
2	37	Two-class Boosted Decision Tree	0.945946	0.833333	1	0.909091	1	0.128366	78.001723
3	37	Two-class Boosted Decision Tree	0.945946	0.8	1	0.888889	1	0.145495	72.131569
4	37	Two-class Boosted Decision Tree	0.945946	1	0.882353	0.9375	0.997059	0.152075	77.955604
Mean	185	Two-class Boosted Decision Tree	0.956757	0.91	0.962185	0.930993	0.999412	0.112156	81.417198
Standard Deviation	185	Two-class Boosted Decision Tree	0.014803	0.092496	0.054298	0.031498	0.001315	0.043598	7.929283

***Table 10**: Improved Two-class Boosted Decision Tree evaluation by folds of the Cross-validate Model*

6.3.5 Two-class Neural Network

The last ML algorithm is the *Two-Class Neural Network* (***Figure 60***). As shown in the configuration panel, some of the hyperparameters, such as number of hidden nodes and initial learning weights diameter, must be manually specified even if applying the parameter sweeping optimization, since the Tune Model Hyperparameters module doesn't allow to indicate a range. Then, the optimal value for those settings are figured out through some iterations and then tuning is performed over the remaining parameters.

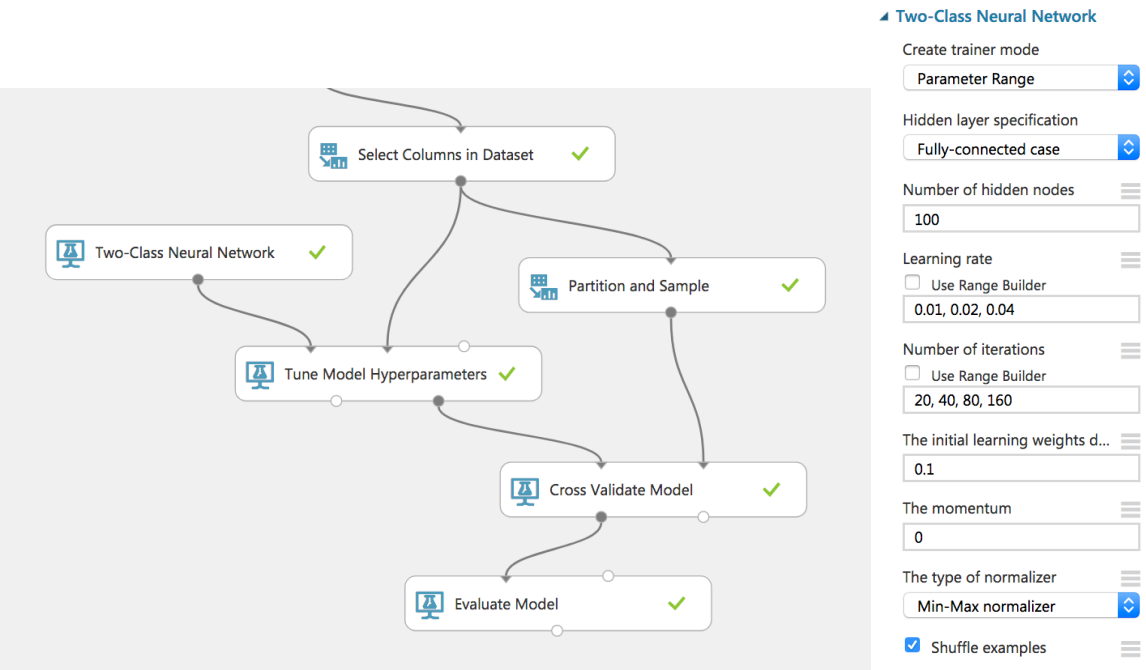


***Figure 60**: Improved Two-Class Neural Network (left) and the algorithm parameters settings (right)*

Differently from the other algorithm, the improved model for Neural Network binary classification task shows a worse performance compared to the one obtained in ***Section 5.1.6*** (***Figure 61***). As mentioned above, the 100% true positive rate previously achieved could be

due to overfitting: the model is too sensitive and captures random patterns (noise) which are present only in the current dataset. Indeed, the performance metrics by fold in ***Table 11*** suggest that the model is not good in generalizing, given the variability registered for precision, recall and f-score statistics.

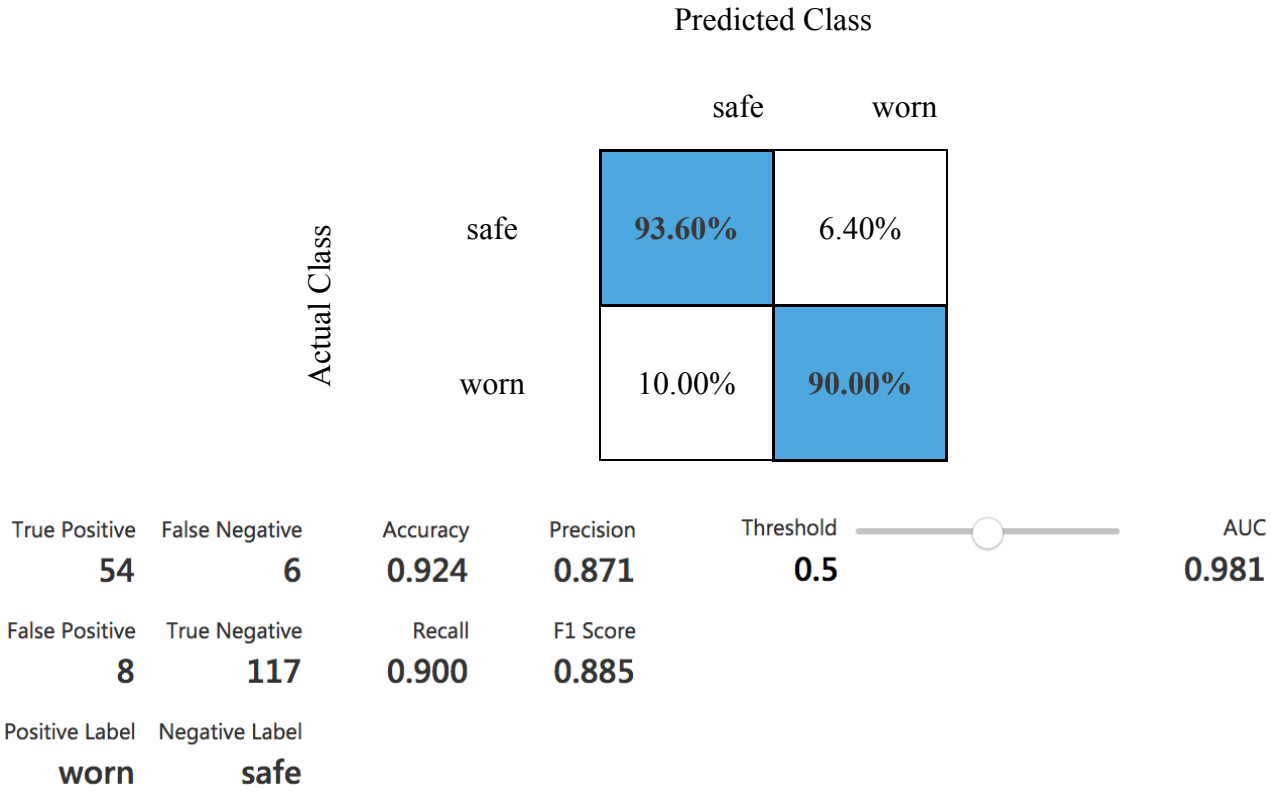


Figure 61*: Improved Two-Class Neural Network's confusion matrix and overall evaluation by standard metrics*

Fold Number	Number of examples in fold	Model	Accuracy	Precision	Recall	F-Score	AUC	Average Log Loss	Training Log Loss
0	37	Two-class Neural Network	0.918919	0.818182	0.9	0.857143	0.97037	0.229642	60.645832
1	37	Two-class Neural Network	0.945946	0.9375	0.9375	0.9375	0.982143	0.214812	68.59425
2	37	Two-class Neural Network	1	1	1	1	1	0.125737	81.617073
3	37	Two-class Neural Network	0.837838	0.692308	0.818182	0.75	0.965035	0.25697	57.773913
4	37	Two-class Neural Network	0.918919	0.833333	0.714286	0.769231	0.985714	0.155279	67.986691
Mean	185	Two-class Neural Network	0.924324	0.856265	0.873994	0.862775	0.980652	0.196488	67.323552
Standard Deviation	185	Two-class Neural Network	0.058593	0.118445	0.110915	0.107139	0.013709	0.054307	9.248604

Table 11*: Improved Two-Class Neural Network evaluation by folds of the Cross-validate Model*

6.4 Model Improvement – Regression

Model improvement techniques are applied to all the regression tasks according to the following procedure:

1. From the data prepared in ***Section 4***, the data flow will follow 2 path: the first one will lead to the input node of a Tune Model Hyperparameters module, the second will directed to a Partition and Sample module;
2. The Tune Model Hyperparameters will take an untrained ML algorithm and will perform a parameter sweep, set on **Entire Grid** mode, and optimizing **Mean Absolute Error metric**. The untrained model uses the Parameter Range as trainer mode (previously set as Single Parameter);
3. The Partition and Sample module will divide the dataset into **5 Folds** for the Cross-validate Model Module;
4. The Cross-validate Model module will take the best model from the Tune Model Hyperparameters together with dataset divided in folds by the Partition and Sample module and will perform a cross validation training;
5. The results of the cross validation will be shown either on Evaluation results by fold output of the Cross-validate Model module (metrics for each test fold) and on the Evaluate Model module;
6. The custom Execute Python script allows to implement the graphical tool used to visualize how the model fit the data (predicted VB vs. actual VB).

6.4.1 Linear Regression

The first ML algorithm is the Linear Regression (Figure ()). The Ordinary Least Squares method does not accept a range of parameter to sweep, therefore the Tune Model Hyperparameters module is not applied. We are only required to specify the L2 regularization weight, which is manually set to 0.001 after some iterations.

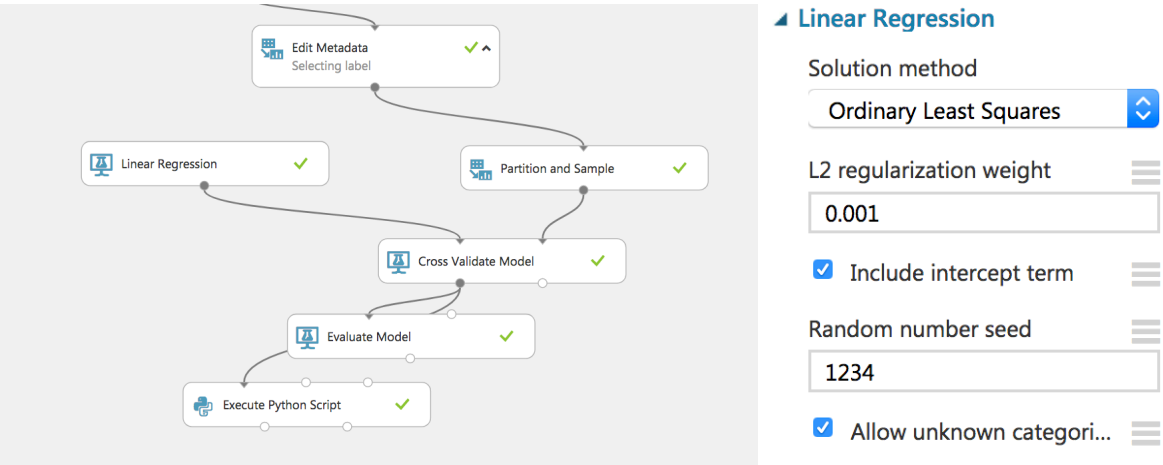
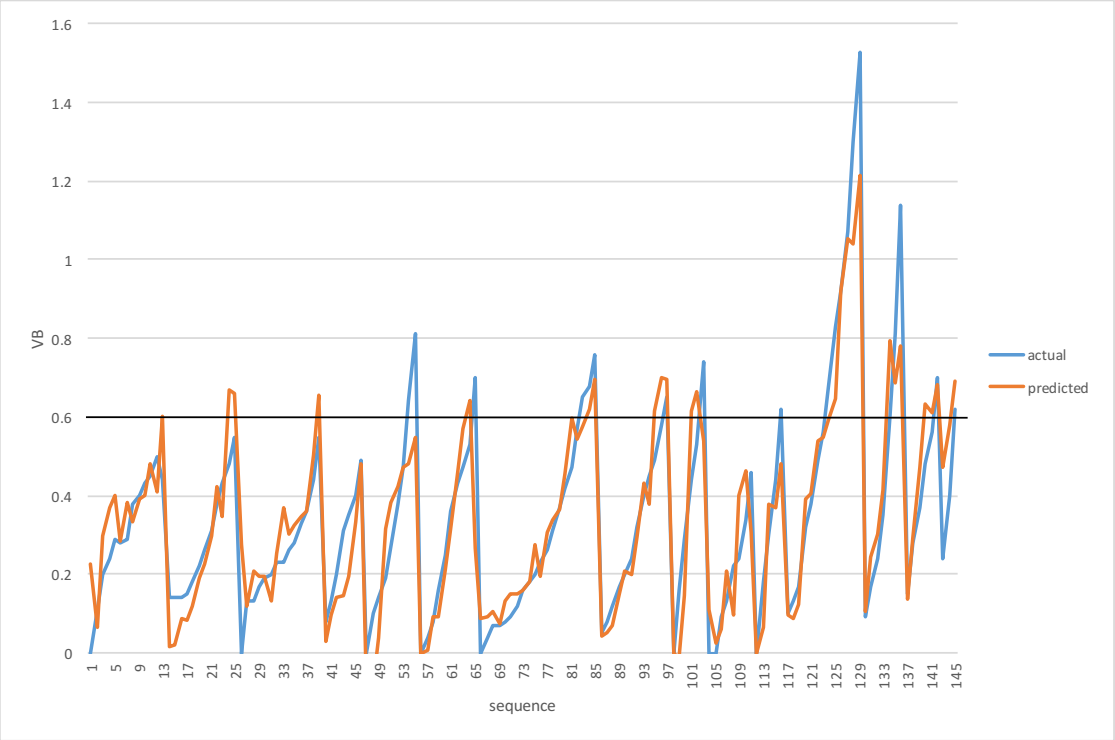


***Figure 62**: Improved Linear Regression (left) and the algorithm parameters settings (right)*

The metrics and the graph (***Figure 63***) show good results, even if the CoD slightly decreases compared to the previous model (from 86% to 82%). This could be explained by looking at the output of validation metrics by fold: ***Table 12*** suggests that the model is a bit susceptible to variations in data, especially with respect to R-squared.



Mean Absolute Error	**0.082575**
Root Mean Squared Error	**0.11092**
Relative Absolute Error	**0.418968**
Relative Squared Error	**0.182504**
Coefficient of Determination	**0.817496**

***Figure 63**:Improved Linear Regression model output. On the top, the graphical representation. On the bottom, overall performance of the regression by standard metrics*

Fold Number	Number of examples in fold	Model	Mean Absolute Error	Root Mean Squared Error	Relative Absolute Error	Relative Squared Error	Coefficient of Determination
0	29	Linear Regression	0.074221	0.097901	0.393121	0.146567	0.853433
1	29	Linear Regression	0.088617	0.11877	0.403151	0.190207	0.809793
2	29	Linear Regression	0.068359	0.088211	0.404633	0.136433	0.863567
3	29	Linear Regression	0.096656	0.124969	0.478614	0.179005	0.820995
4	29	Linear Regression	0.085021	0.120114	0.472963	0.322081	0.677919
Mean	145	Linear Regression	0.082575	0.109993	0.430496	0.194859	0.805141
Standard Deviation	145	Linear Regression	0.011321	0.016003	0.04163	0.074505	0.074505

***Table 12**: Improved Linear Regression evaluation by folds of the Cross-validate Model*

6.4.2 Boosted Decision Tree Regression

The second ML algorithm is the *Boosted Decision Tree Regression* (Figure 64).

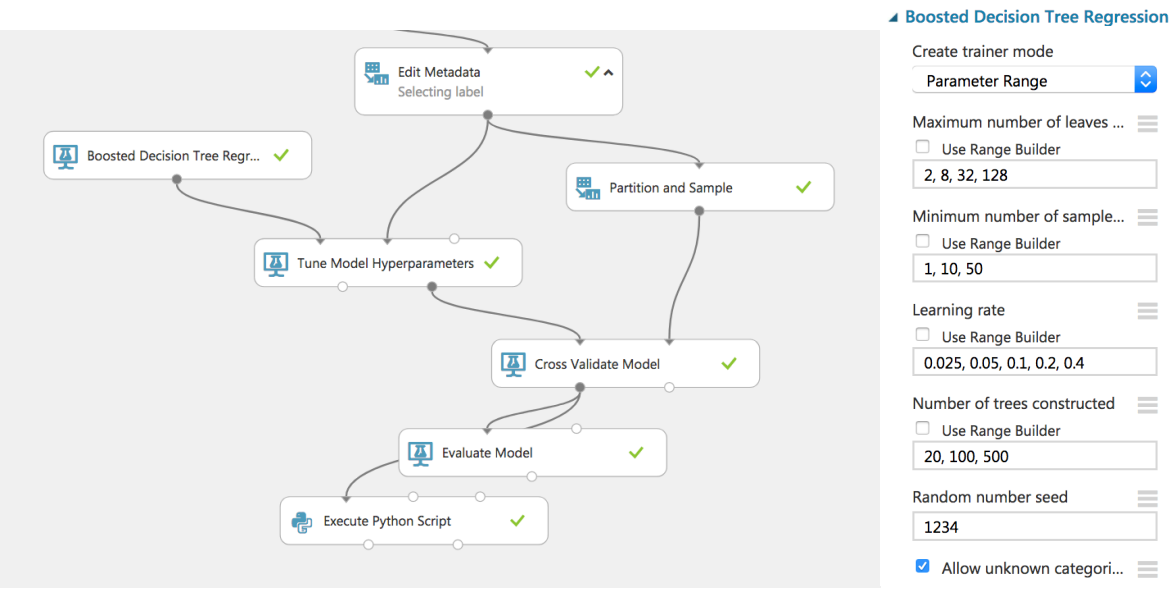
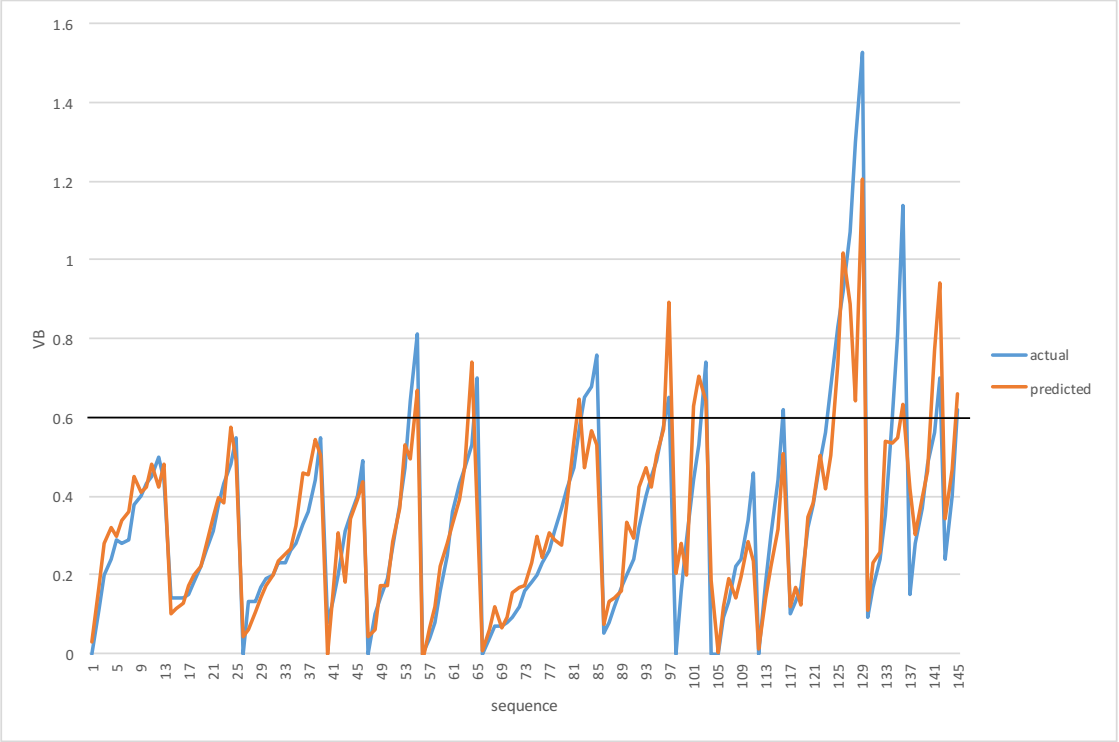


***Figure 64**: Improved Boosted Decision Tree Regression (left) and the algorithm parameters settings (right)*

As illustrated in **Figure 65**, both the general metrics and the plot confirms the performance of the model in ***Section 5.2.3***. However, the standard deviation of the metrics of each fold is not sufficiently low with respect to the MAE (***Table 13***).



Mean Absolute Error	**0.077345**
Root Mean Squared Error	**0.119834**
Relative Absolute Error	**0.392432**
Relative Squared Error	**0.213016**
Coefficient of Determination	**0.786984**

***Figure 65**: Improved Boosted Decision Tree model output. On the top, the graphical representation. On the bottom, overall performance of the regression by standard metrics*

Fold Number	Number of examples in fold	Model	Mean Absolute Error	Root Mean Squared Error	Relative Absolute Error	Relative Squared Error	Coefficient of Determination
0	29	Boosted Decision Tree Regression	0.054752	0.079595	0.350376	0.134865	0.865135
1	29	Boosted Decision Tree Regression	0.065527	0.087296	0.463253	0.280669	0.719331
2	29	Boosted Decision Tree Regression	0.088669	0.151765	0.390953	0.258744	0.741256
3	29	Boosted Decision Tree Regression	0.107042	0.156357	0.460331	0.248732	0.751268
4	29	Boosted Decision Tree Regression	0.070734	0.10181	0.428709	0.281754	0.718246
Mean	145	Boosted Decision Tree Regression	0.077345	0.115365	0.418724	0.240953	0.759047
Standard Deviation	145	Boosted Decision Tree Regression	0.020639	0.036251	0.048105	0.060979	0.060979

***Table 13**: Improved Boosted Decision Tree evaluation by folds of the Cross-validate Model*

6.4.3 Decision Forest Regression

The third ML algorithm is the *Decision Forest Regression* (***Figure 66***).

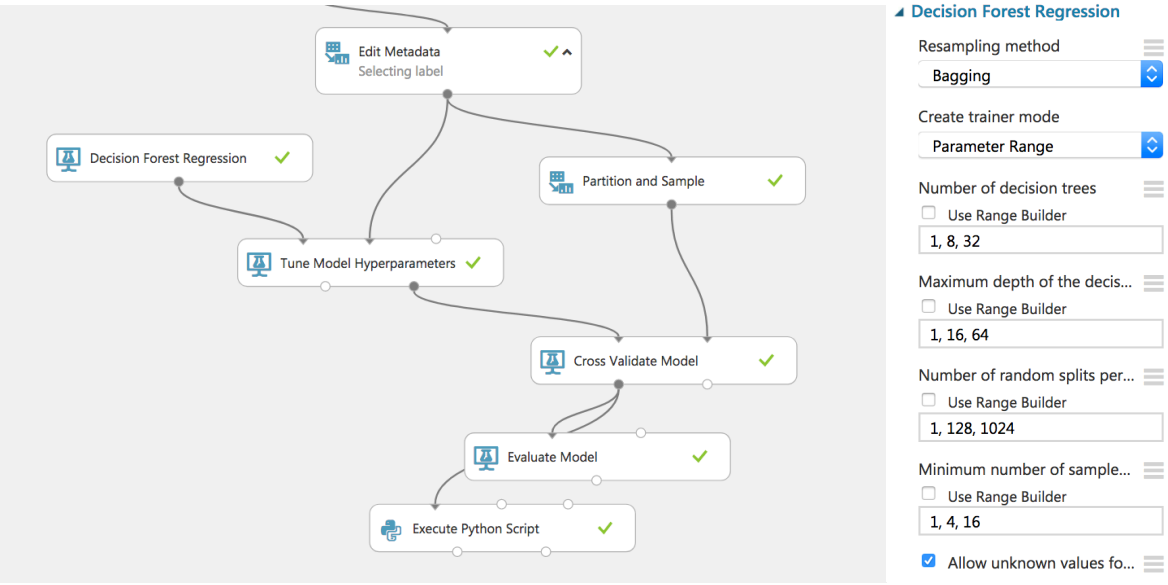
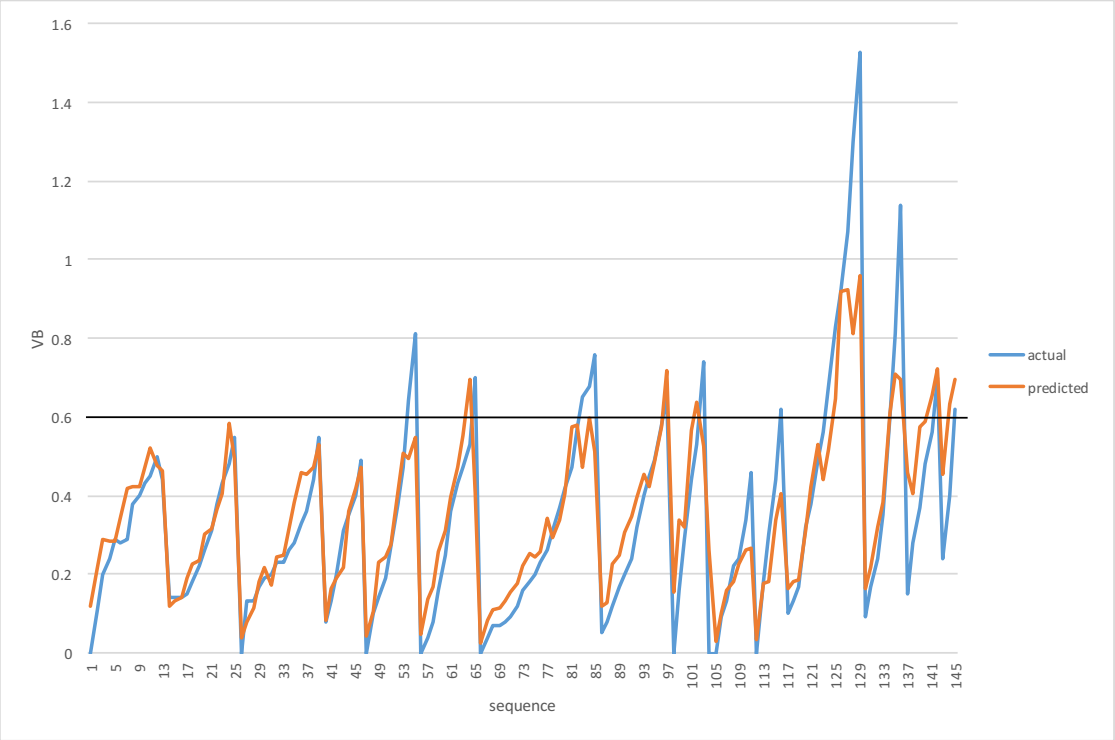


Figure 66*: Improved Decision Forest Regression (left) and the algorithm parameters settings (right)*

The model is enhanced in MAE and CoD (***Figure 67***), respectively 0.08 and 78% (compared to 0.09 and 71% obtained without improvement). In ***Table 14***, the output of the Cross-validate Model's evaluation by folds is reported. As can be seen, the model generalizes better than the previous algorithm.



Mean Absolute Error	**0.080905**
Root Mean Squared Error	**0.120484**
Relative Absolute Error	**0.410494**
Relative Squared Error	**0.215331**
Coefficient of Determination	**0.784669**

***Figure 67**: Improved Decision Forest Regression model output. On the top, the graphical representation. On the bottom, overall performance of the regression by standard metrics*

Fold Number	Number of examples in fold	Model	Mean Absolute Error	Root Mean Squared Error	Relative Absolute Error	Relative Squared Error	Coefficient of Determination
0	29	Decision Forest Regression	0.072589	0.094873	0.464523	0.191607	0.808393
1	29	Decision Forest Regression	0.060716	0.080211	0.429237	0.236959	0.763041
2	29	Decision Forest Regression	0.089872	0.132591	0.396261	0.197493	0.802507
3	29	Decision Forest Regression	0.10341	0.162862	0.44471	0.269858	0.730142
4	29	Decision Forest Regression	0.077937	0.114205	0.47236	0.354534	0.645466
Mean	145	Decision Forest Regression	0.080905	0.116948	0.441418	0.25009	0.74991
Standard Deviation	145	Decision Forest Regression	0.016374	0.032393	0.030371	0.066442	0.066442

***Table 14**: Improved Decision Forest Regression evaluation by folds of the Cross-validate Model*

6.4.4 Bayesian Linear Regression

The fourth ML algorithm is the *Bayesian Linear Regression* (***Figure 68***). As for the basic Linear Regression, this learner does not support setting a range of values to use in a parameter sweep. The regularization weight is therefore decided through some iterations by manually changing it; the best result is achieved whit 0.5.

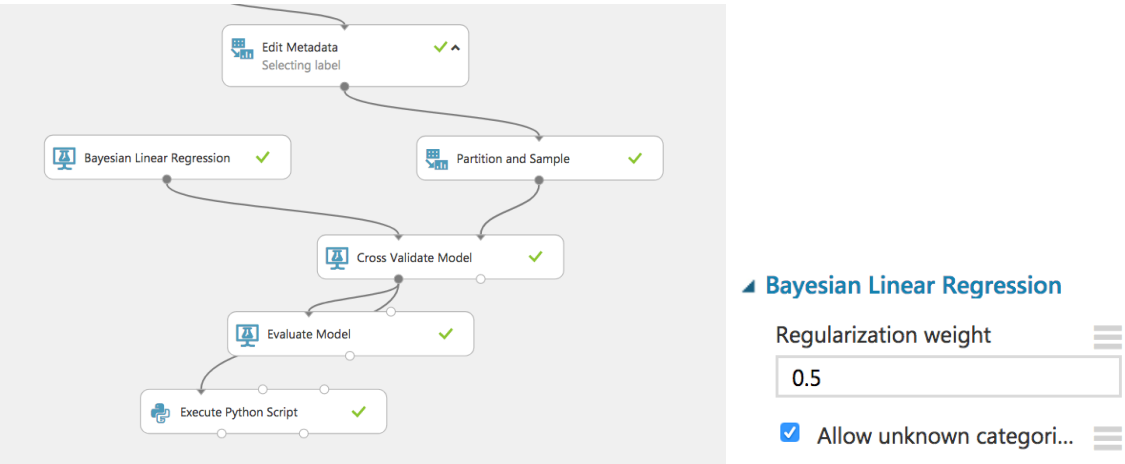
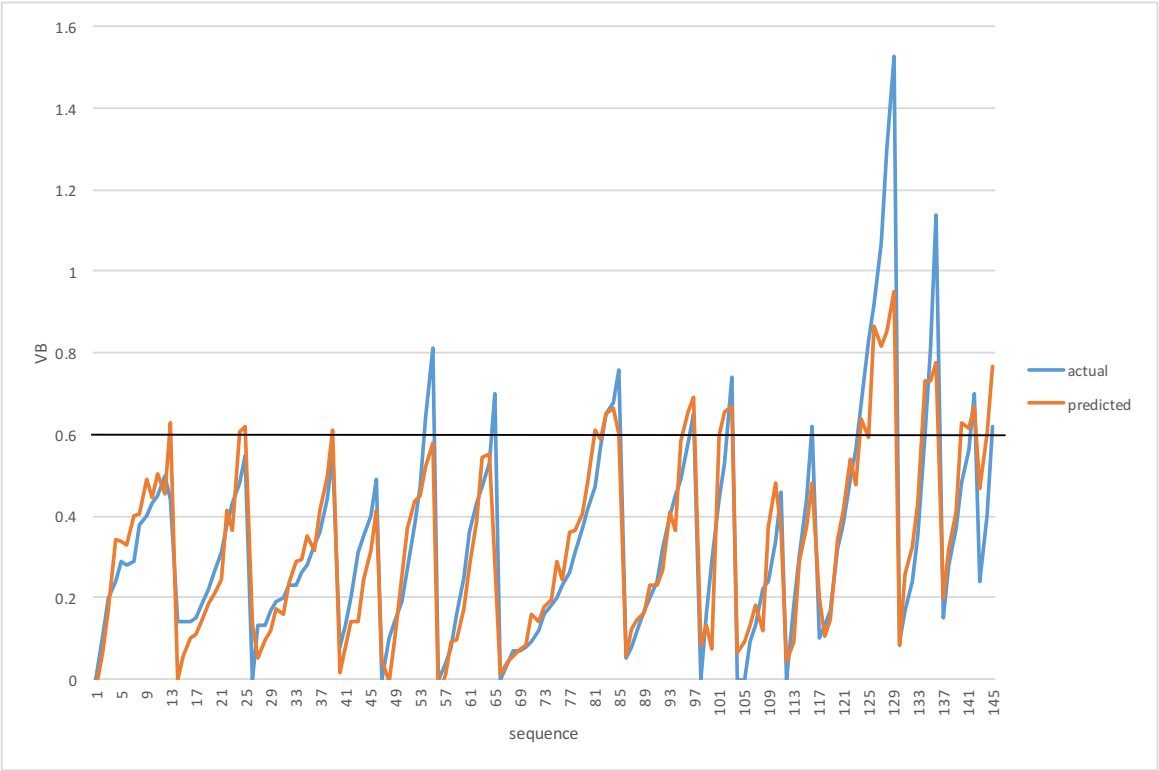


Figure 68*: Improved Bayesian Linear Regression (left) and the algorithm parameters settings (right)*

The graph shows great results, confirming the already good performance achieved without improvement (***Figure 69***). In ***Table 15***, the output of the Cross-validate Model's evaluation by folds is summarized. As can be seen, the model generalizes better than the previous algorithm.



Mean Absolute Error	**0.078617**
Root Mean Squared Error	**0.114071**
Relative Absolute Error	**0.398888**
Relative Squared Error	**0.19302**
Coefficient of Determination	**0.80698**

***Figure 69**: Improved Bayesian Linear Regression model output. On the top, the graphical representation. On the bottom, overall performance of the regression by standard metrics*

Fold Number	Number of examples in fold	Model	Mean Absolute Error	Root Mean Squared Error	Relative Absolute Error	Relative Squared Error	Coefficient of Determination
0	29	Bayesian Linear Regression	0.076258	0.101172	0.53605	0.251871	0.748129
1	29	Bayesian Linear Regression	0.104485	0.169537	0.379739	0.212316	0.787684
2	29	Bayesian Linear Regression	0.060857	0.073214	0.328041	0.106079	0.893921
3	29	Bayesian Linear Regression	0.080826	0.114673	0.382439	0.207041	0.792959
4	29	Bayesian Linear Regression	0.070661	0.087022	0.416203	0.195167	0.804833
Mean	145	Bayesian Linear Regression	0.078617	0.109123	0.408494	0.194495	0.805505
Standard Deviation	145	Bayesian Linear Regression	0.016266	0.037154	0.077956	0.053817	0.053817

***Table 15**: Improved Bayesian Linear Regression evaluation by folds of the Cross-validate Model*

6.4.5 Neural Network Regression

The last ML algorithm is the *Neural Network Regression* (***Figure 70***). As for the classification task, some hyperparameters still need to be manually specified. The choice for the number of hidden nodes falls on 64 after performing some iterations.

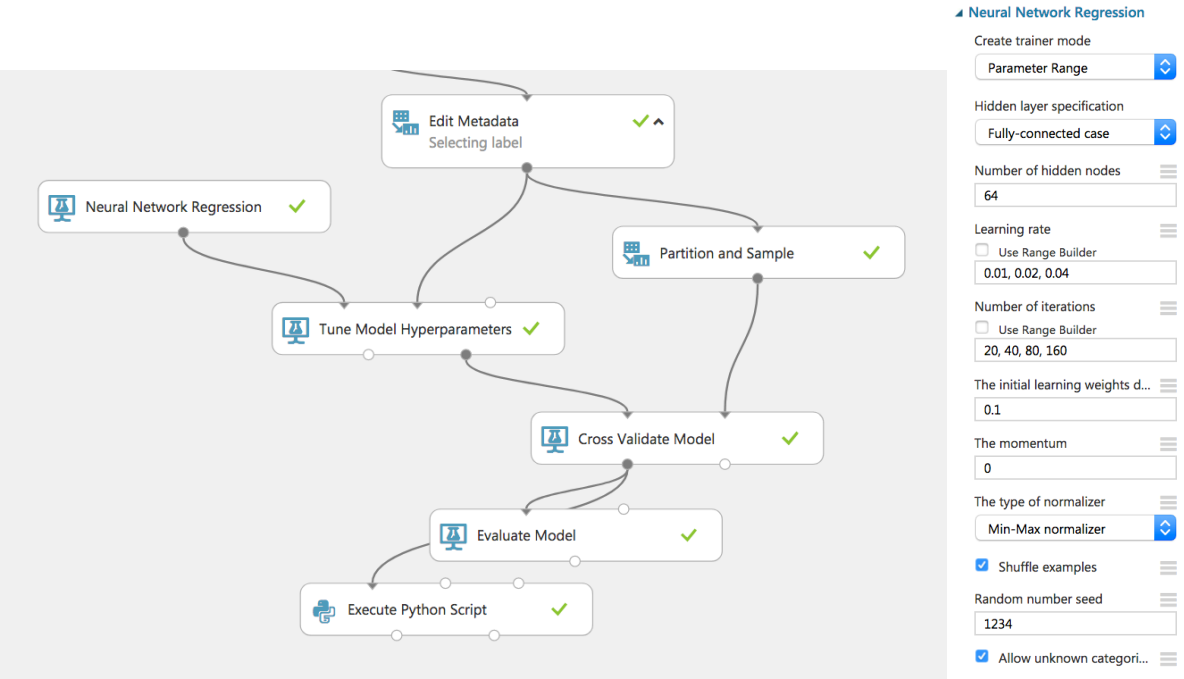
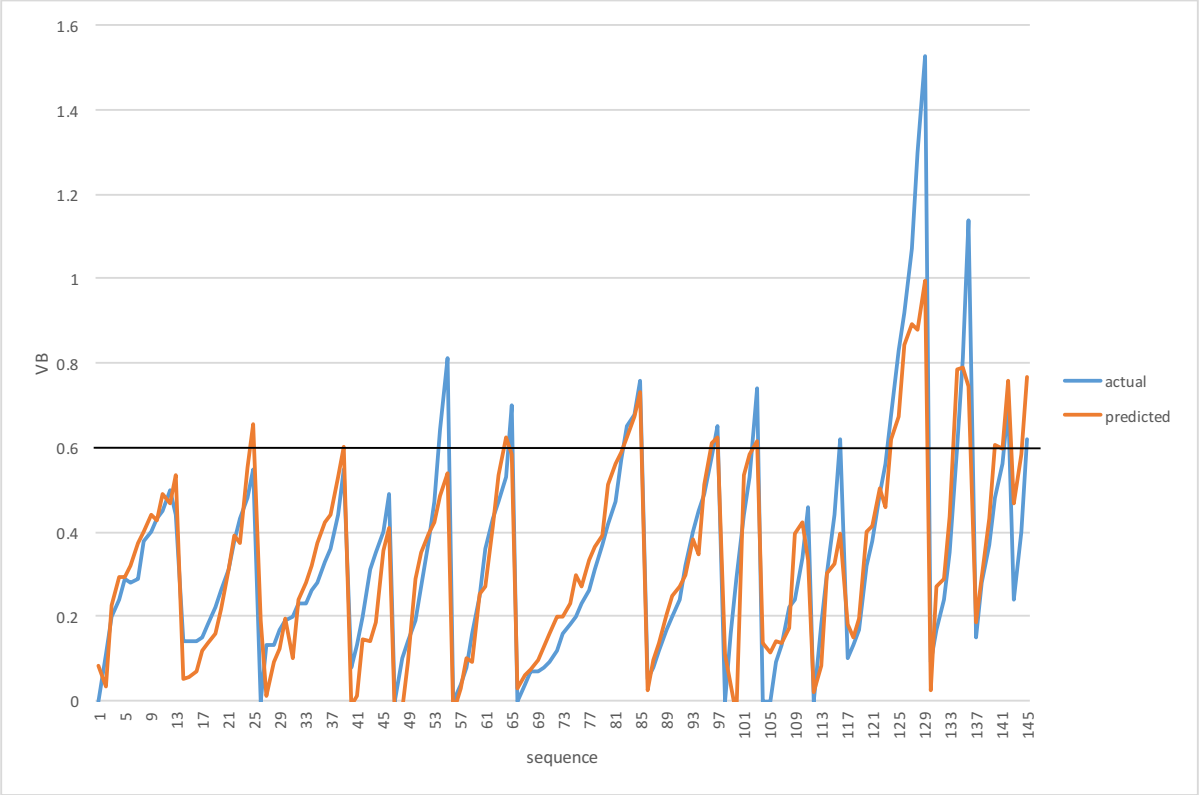


***Figure 70**: Improved Neural Network Regression (left) and the algorithm parameters settings (right)*

The obtained performance is highly improved compared to the one achieved in ***Section 5.2.6*** (***Figure 71***): the MAE is 0.777 and the R^2 is over 82%. This huge difference could be due to the change in configuration settings of the Neural Network Regression model. The determinist definition of parameters might have negatively influenced the previous results, while the Tune Model Hyperparameters allows us to identify the best combination of settings and to enhance the performance.

As regard the reliability of the new model, the evaluation metrics computed for each fold (***Table 16***) look consistent with a low variance. Thus, we can conclude that the model is also able to generalize well on unseen data.



Mean Absolute Error	**0.077787**
Root Mean Squared Error	**0.109908**
Relative Absolute Error	**0.394677**
Relative Squared Error	**0.179188**
Coefficient of Determination	**0.820812**

Figure 71*: Improved Neural Network Regression model output. On the top, the graphical representation. On the bottom, overall performance of the regression by standard metrics*

Fold Number	Number of examples in fold	Model	Mean Absolute Error	Root Mean Squared Error	Relative Absolute Error	Relative Squared Error	Coefficient of Determination
0	29	Neural Network Regression	0.075414	0.115535	0.399441	0.204123	0.795877
1	29	Neural Network Regression	0.082058	0.111723	0.373312	0.168306	0.831694
2	29	Neural Network Regression	0.059328	0.07857	0.351176	0.108238	0.891762
3	29	Neural Network Regression	0.098798	0.143417	0.48922	0.235756	0.764244
4	29	Neural Network Regression	0.073339	0.088471	0.407976	0.174739	0.825261
Mean	145	Neural Network Regression	0.077787	0.107543	0.404225	0.178232	0.821768
Standard Deviation	145	Neural Network Regression	0.014369	0.025355	0.052514	0.047406	0.047406

***Table 16**: Improved Neural Network Regression evaluation by folds of the Cross-validate Model*

6.5 Summary of results

The results are summarized in ***Table 17*** for Classification task and in ***Table 18*** for Regression task.

The most performant algorithm for the classification task is the *Two-Class Boosted Decision Tree*. It shows the highest evaluation metrics, with an accuracy of almost 96% and a precision of 92%. The misclassification is the lowest for both the "safe" and the "worn" class, respectively 5% and 4%.

Metric	Two-class Decision Forest	Two-class Logistic Regression	Two-class Decision Jungle	Two-class Boosted Decision Tree	Two-class Neural Network
TP rate	91.67%	90.00%	91.67%	95.00%	90.00%
TN rate	93.76%	96.00%	95.20%	96.00%	93.60%
Accuracy	0.930	0.941	0.941	0.957	0.924
Precision	0.873	0.915	0.902	0.919	0.871
Recall	0.917	0.900	0.917	0.950	0.900
F1-Score	0.894	0.908	0.909	0.934	0.885
AUC	0.972	0.982	0.979	0.995	0.981

***Table 17**: Summary of evaluation metrics for the Classification task*

As regard the regression task, the most performant algorithm is the *Neural Network Regression*. This model returns the minimum error values (for instance a 0.077 MA), as well as the greatest R^2 of 82%, meaning that it is able to approximate a good prediction of VB.

Metric	Linear Regression	Boosted Decision Tree Regression	Decision Forest Regresion	Bayesian Linear Regression	Neural Network Regression
MAE	0.082575	0.077845	0.080905	0.078617	0.077787
RMSE	0.11092	0.119834	0.120484	0.114071	0.109908
RAE	0.418968	0.397432	0.410494	0.398888	0.394677
RSE	0.182504	0.213016	0.215331	0.19302	0.179188
R-squared	0.817496	0.786984	0.784669	0.80698	0.820812

***Table 18**: Summary of evaluation metrics for the Classification task*

7. Remaining Useful Life Predictive Model

The models implemented so far perfectly address the issue of determining when the cutting tool should be replaced: the classification model flags the tool as "worn" when it needs to be changed, while the regression returns the estimation of VB from which the milling operator can derives the real state of the inserts. However, these models are not able to exactly specify the Remaining Useful Life of the tool, i.e. how long before it needs to be changed. To further improve the analysis, this section discusses an additional predictive model to address this question.

First of all, the Remaining Useful Life (RUL) must be defined for our case study. Generally, the Remaining Useful Life (RUL) refers to the expected period of time during which an asset or system is likely to operate in accordance with its intended purpose before it requires repair or replacement. The estimation of RUL is based on the real useful life of an asset. However, this information is not always provided: in many problems, the useful life is random and unknown, and as such it must be estimated from available sources of information such as the information obtained from the tool condition monitoring [67].
According to our dataset, the RUL could be represented as the remaining Runs that can be performed with the same cutter before it wears. Once again, the criteria used for the wear is the critical flank wear (VB) value of 0.6. Since the wear was measured after each run, we assume (that the useful life of the inserts given the specific cutting operations to be performed corresponds) to last practicable Run for each Case is the first one corresponding to a VB>= 0.6. Therefore, we can estimate the remaining runs for each observation as the difference between this maximum number of feasible runs, varying for each case, and the current run.

The procedure for developing the RUL Predictive Model follows similar steps as for regression task implemented for the continuous value VB: dataset is prepared by cleaning it and selecting relevant features and, after creating the new label feature, regression algorithms are trained and tested to evaluate their performance in order to identify the most suitable one for our analysis.

7.1 Data preparation

Figure 72 illustrates the configuration chart of the data preparation phase in the Azure MLS canvas. The execution of this stage takes almost the same steps as the Regression task for estimating VB, with two additional modules highlighted below.

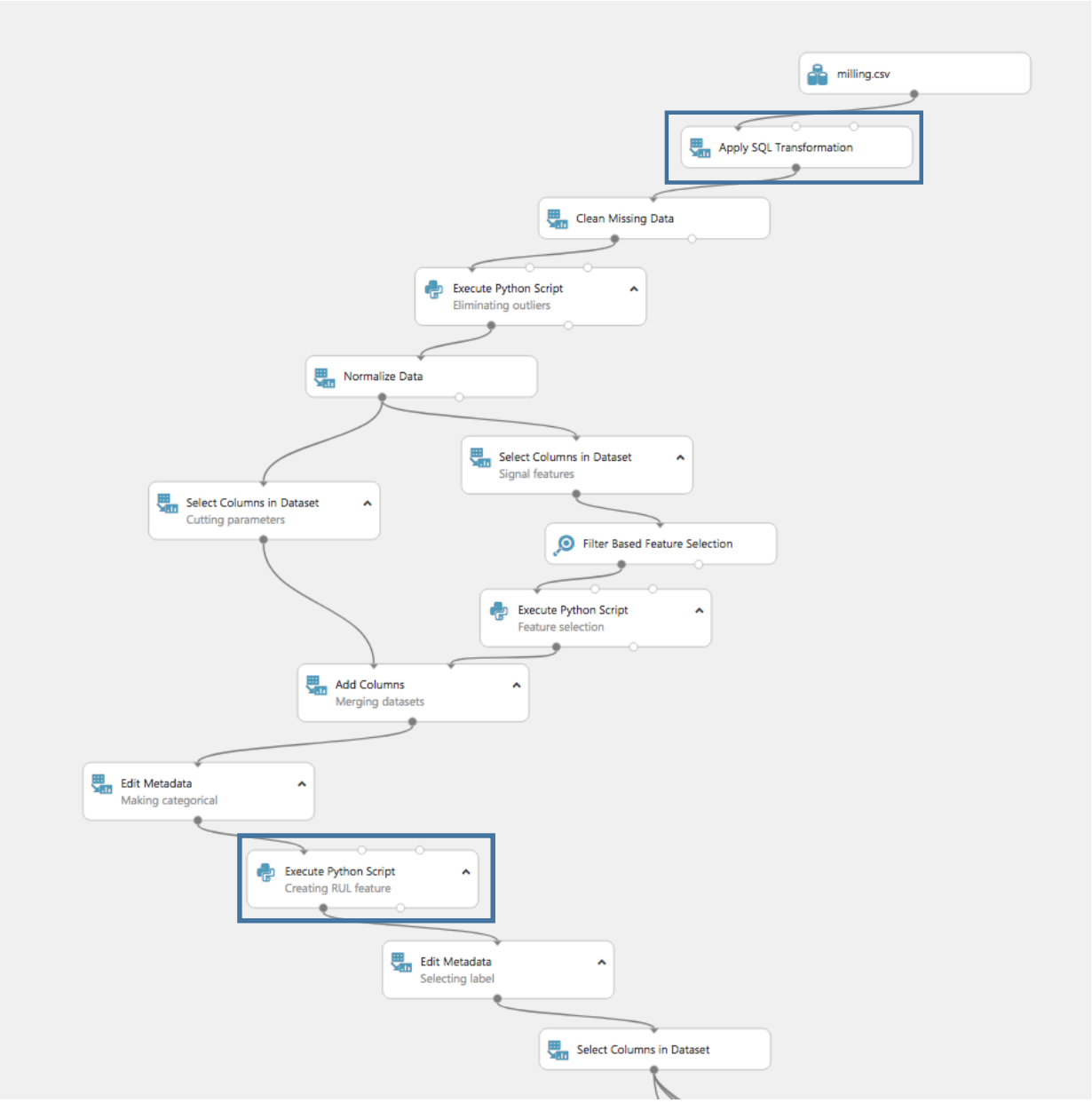


***Figure 72**: Configuration of the Data preparation step for the RUL model in Azure MLS canvas*

The first box imports the milling dataset. In this circumstance, not all observations are relevant for the purpose of our analysis. In fact, during the experiment, the critical value VB=0.6 was not achieved in every Case. Hence, in order to proceed with the subsequent construction of the RUL feature as intended in our implementation of the model, it is essential to skim the dataset and take into consideration only those cases where the maximum reported VB was actually greater than 0.6. The *Apply SQL Transformation* can be used for this task (***Figure 73***): it identifies the cases having a maximum VB>=0.6 and then selects all the observations related to these cases.

```
1 select * from t1
2 where caso  in (select caso from t1
3                 group by caso
4                 having max(VB)>=0.6)
```

***Figure 73**: SQL code for selecting only the case whose maximum VB is greater than 0.6*

Data cleaning involves *Clean Missing Data* and *Execute Python Script* modules, which respectively remove rows where VB is not available and observations representing clear outliers. After applying data normalization to signal features, feature selection is performed as already carried out in ***Section 4.4***. The resulting dataset from *Add Columns* module contains all cutting parameters and the identified significant signal features. *Edit Metadata* module simply marks the "material" feature as categorical, while the new *Execute Python Script* module has the role of creating the RUL feature (***Figure 74***). The function rmax(case) returns the maximum number of feasible runs given a specific case and then iterates over the dataset to compute the RUL (maximum run-current run) for each observation.

```
import pandas as pd

def azureml_main(df):

    df['RUL'] = 0
    rul = []

    def rmax(case):
        rmax = 0
        for index,row in df.iterrows():
            if int(row['case']) == case:
                if row['VB']>= 0.6:
                    rmax = int(row['run'])
                    break
        return rmax

    for index,row in df.iterrows():
        case = int(row['case'])
        rm = int(rmax(case))
        run = int(row['run'])
        rul.append(int(rm-run))

    se = pd.Series(rul)
    df['RUL'] = se.values

    return df
```

The created column is flagged as label feature by the second *Edit Metadata* module and then the *Select Columns in Dataset* module excludes the VB feature from the dataset, since the RUL is determined based on it; moreover, it would be costly for new data having such measurement.

7.2 Model Training and Testing

Five ML algorithms are chosen to be evaluated: Linear Regression, Bayesian Linear Regression, Boosted Decision Tree Regression, Decision Forest Regression and Neural Network Regression (***Figure 74***).

The way of proceeding is the same as for the regression task applied for VB in ***Section 5.2***. For the first two algorithms, which do not support parameter sweeping, the dataset is split into 5 folds by the *Partition and Sample* module and then the model is trained and tested according to the cross validation method. As regard the other three algorithms, the procedure will involve the *Tune Model Hyperparameters*, which takes the entire dataset and the untrained ML

algorithm and performs a parameter sweep, set on **Entire Grid** mode, and optimizing **Mean Absolute Error** metric. The Evaluation Model module returns the standard metrics for a regression task, both as overall performance and by fold.

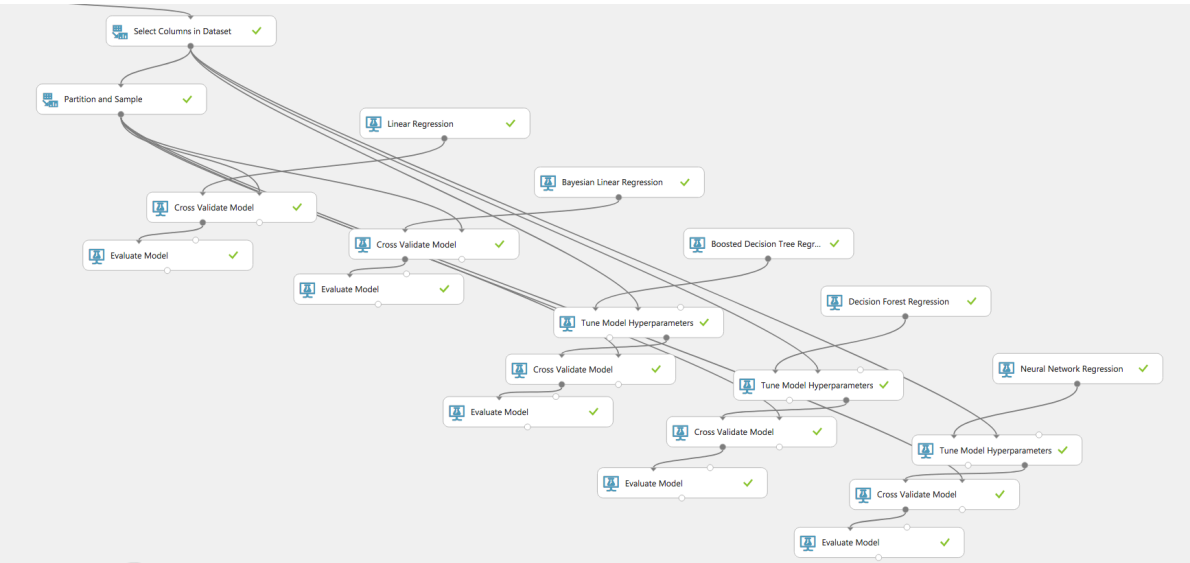


***Figure 74**: Model training and testing phase for the RUL model in Azure MLS canvas*

The results of the training and testing phase are summarized in **Table 19**.

Metric	Linear Regression	Bayesian Linear Regression	Boosted Decision Tree Regression	Decision Forest Regression	Neural Network Regression
MAE	1.671182	1.640172	1.614642	1.516941	0.581028
RMSE	2.158402	2.134108	2.021055	1.873197	0.746141
RAE	0.429251	0.421286	0.414729	0.389634	0.138989
RSE	0.178105	0.174119	0.15616	0.134146	0.022034
R-squared	0.821895	0.825881	0.84384	0.865854	0.979086

***Table 19**: Summary of results for the RUL model*

Neural Network Regression proves to be the most performant algorithm, showing the lowest MAE, just 0.58, and the highest $R^{2,}$ equal to 98%. As confirmed by the graph of predicted vs actual RUL for each observation, the model excellently fits the data, with a maximum error of 2 runs (***Figure 75***).

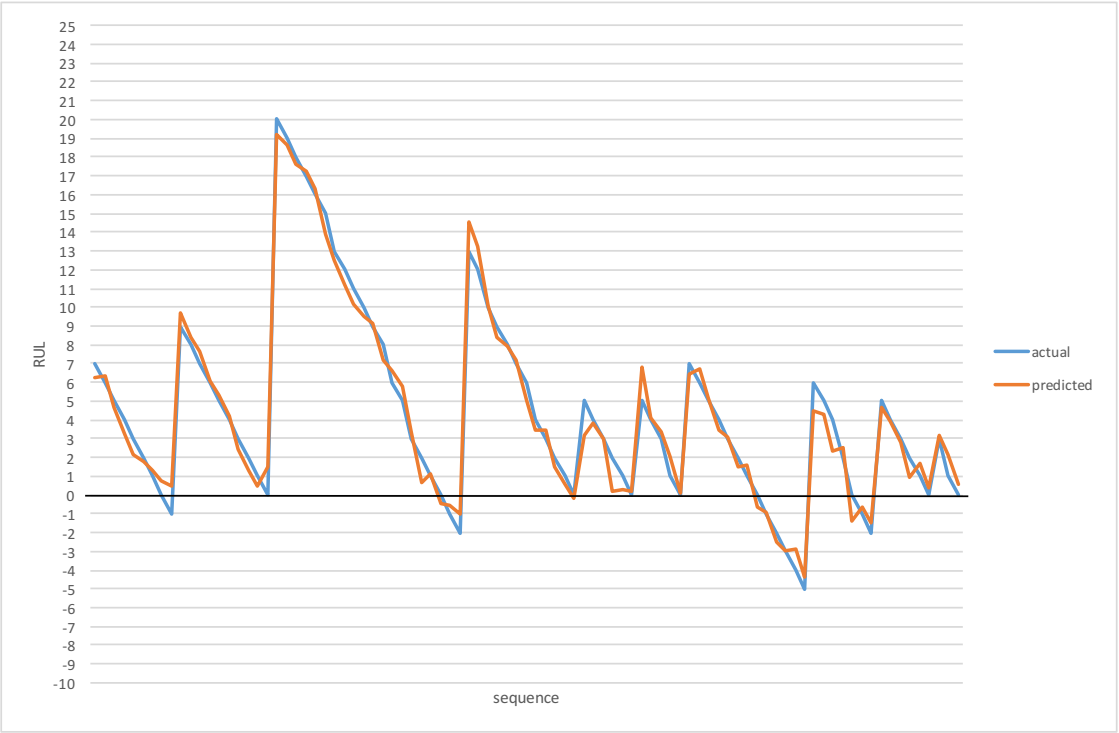


***Figure 75**: Neural Network Regression graphical representation of predicted RUL (red) vs. actual RUL (blue)*

The evaluation metrics for each fold are reported in ***Table 20***. They indicate a low standard deviation compared to the mean value, meaning that the model is reliable and is able to generalize really well in front of new data.

Fold Number	Number of examples in fold	Model	Mean Absolute Error	Root Mean Squared Error	Relative Absolute Error	Relative Squared Error	Coefficient of Determination
0	18	Neural Network Regression	0.595246	0.820685	0.152821	0.030338	0.969662
1	18	Neural Network Regression	0.589325	0.695286	0.153379	0.027141	0.978442
2	18	Neural Network Regression	0.621621	0.704898	0.134449	0.012586	0.987432
3	18	Neural Network Regression	0.557326	0.763811	0.113426	0.016014	0.983986
4	19	Neural Network Regression	0.54162	0.746026	0.140868	0.024091	0.975909
Mean	91	Neural Network Regression	0.581028	0.746141	0.138989	0.022034	0.979086
Standard Deviation	91	Neural Network Regression	0.031755	0.050389	0.016401	0.007496	0.006946

***Table 20**: Neural Network Regression's evaluation metrics by folds of Cross-validate Module*

8. Conclusions

The objective of this Thesis was to develop a Cutting-tool Wear Monitoring and Predictive Maintenance system. So far, three different models have been presented for this purpose:

- Classification of tool inserts as “safe” or “worn” by Two-Class Boosted Decision Tree algorithm;
- Estimation of flank wear measurement (VB) by Neural Network Regression algorithm;
- Remaining Useful Life (RUL) estimation, i.e. feasible runs before the cutter wear, by Neural Network Regression algorithm.

Those models have been demonstrated to perform excellently and can be used for implementing a technology which would help the milling operator in evaluating the replacement of a worn cutting tool. ***Figure 76*** shows a possible layout to visualize the predicted information after a certain run on a device connected to a milling machine.

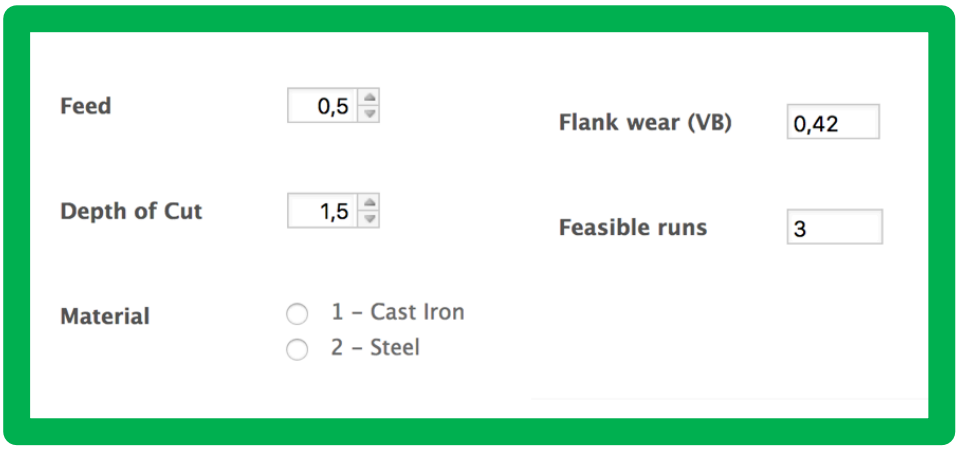


Figure 76*: Sample visualization of the implemented Cutting-tool Wear Monitoring and Predictive Maintenance system*

On the left side, the milling operator could specify the cutting parameters in which the milling operation is carried out: feed rate, depth of cut and material. Till now, the device can accept only two kinds of workpiece material, cast iron or steel, while Feed and DOC can be manually inserted. As the machine works, the ML models return three kind of information. By means of the classification task, the framework of the panel is coloured of green whether the tool is classified as “safe” or red when it is marked as “worn”. An estimate of the flank wear measurement VB is displayed on the right side, together with the number of feasible runs which can still be performed; both values are predicted through the specific regression tasks developed in this paper. The three evidences give multiple indications about the state of the cutting tool, each with a certain level of reliability. It could happen that those predictions are in discordance. This case indicates an ambiguous situation which should be further evaluated by the milling operator, but the system still represents a powerful instrument to increase efficiency and can be improved in future to become of practical use in manufacturing industry.

References

[1] B. Marr, «Forbes,» 2018. [Online]. Available: https://www.forbes.com/sites/bernardmarr/2018/09/02/what-is-industry-4-0-heres-a-super-easy-explanation-for-anyone/#7dfbe2fb9788.

[2] B. Sniderman, M. Mahto e M. Cotteleer, «Deloitte Insights,» 2016. [Online]. Available: https://www2.deloitte.com/insights/us/en/focus/industry-4-0/manufacturing-ecosystems-exploring-world-connected-enterprises.html.

[3] i-Scoop, «i-Scoop,» [Online]. Available: https://www.i-scoop.eu/industry-4-0/.

[4] B. Marr, «Forbes,» 2016. [Online]. Available: https://www.forbes.com/sites/bernardmarr/2016/06/20/what-everyone-must-know-about-industry-4-0/#486c25b2795f.

[5] M. Khaki, «LinkedIn,» 2016. [Online]. Available: https://www.linkedin.com/pulse/iot-allows-objects-sensed-controlled-remotely-across-existing-khaki/.

[6] M. Rouse, «IoT Agenda,» 2016. [Online]. Available: https://internetofthingsagenda.techtarget.com/definition/Internet-of-Things-IoT.

[7] S. Kaul, R. Manes, B. Sniderman, L. McGoff e J. Mariani, «Deloitte,» 2017. [Online]. Available: https://www2.deloitte.com/content/dam/Deloitte/us/Documents/process-and-operations/us-cons-predictive-maintenance.pdf.

[8] A. Gilchrist, Industry 4.0: The Industrial Internet of Things, Apress, 2016.

[9] R. Burke, A. Mussomeli, S. Laaper, M. Hartigan e B. Sniderman, «Deloitte Insights,» 2017. [Online]. Available: https://www2.deloitte.com/insights/us/en/focus/industry-4-0/smart-factory-connected-manufacturing.html.

[10] Wikipedia, «Wikipedia.org,» [Online]. Available: https://en.wikipedia.org/wiki/Artificial_intelligence.

[11] Investopedia, «Investopedia,» [Online]. Available: https://www.investopedia.com/terms/a/artificial-intelligence-ai.asp.

[12] Wikipedia, «Wikipedia.org,» [Online]. Available: https://en.wikipedia.org/wiki/Machine_learning.

[13] M. Varone, D. Mayer e A. Melegari, «Expert System,» [Online]. Available: https://www.expertsystem.com/machine-learning-definition/.

[14] M. Rouse, «SearchEnterpriseAI,» 2018. [Online]. Available: https://searchenterpriseai.techtarget.com/definition/machine-learning-ML.

[15] J. Brownlee, «Machine Learning Mastery,» 2016. [Online]. Available: https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms/.

[16] N. Castle, «Data Science,» [Online]. Available: https://www.datascience.com/blog/supervised-and-unsupervised-machine-learning-algorithms.

[17] R. Y. Zhonga, X. Xu, E. Klotz e S. T. Newmanc, «Intelligent Manufacturing in the Context of Industry 4.0: A Review,» 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2095809917307130.

[18] R. Holubek e P. Kostal, «The Intelligent Manufacturing Systems,» 2014. [Online]. Available: https://www.researchgate.net/publication/260918138_The_Intelligent_Manufacturing_Systems.

[19] Wikipedia, «Wikipedia.org,» [Online]. Available: https://en.wikipedia.org/wiki/Predictive_maintenance.

[20] N. Ambhore, D. Kamble, S. Chinchanikara e V. Wayal, «Science Direct,» 2015. [Online]. Available: Tool Condition Monitoring System: A Review.

[21] S. Pal, P. S. Heyns, B. H. Freyer, N. J. Theron e S. K. Pa, «Tool wear monitoring and selection of optimum cutting conditions with progressive tool wear effect and input uncertainties,» 2009.

[Online]. Available: https://repository.up.ac.za/bitstream/handle/2263/13790/Pal_Tool(2009).pdf?sequence=1.

[22] C. Madhusudana, H. Kumar e S. Narendranath, «Condition monitoring of face milling tool,» 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2215098616302142.

[23] C. Xiaorui, *Industrial Internet of Things: Cutting-tool Wear Monitoring and Predictive Maintenance by machine learning algorithms,* 2017.

[24] Wikipedia, «Milling (machining),» [Online]. Available: https://en.wikipedia.org/wiki/Milling_(machining).

[25] M. P. Groover, Fundamentals of modern manufacturing: Materials, Processes, and Systems, 1996.

[26] Wikipedia, «Milling cutter,» [Online]. Available: https://en.wikipedia.org/wiki/Milling_cutter.

[27] Autodex, «A closer look at vertical and horizontal milling machines,» [Online]. Available: https://www.autodesk.com/industry/manufacturing/resources/manufacturing-engineer/vertical-and-horizontal-milling-machines.

[28] Custompart, «Milling,» [Online]. Available: http://www.custompartnet.com/wu/milling.

[29] Wikipedia, «Speeds and feeds,» [Online]. Available: https://en.wikipedia.org/wiki/Speeds_and_feeds.

[30] G. Schneider, «Cutting Tool Materials,» 2009. [Online]. Available: https://www.americanmachinist.com/cutting-tools/chapter-1-cutting-tool-materials.

[31] Wikipedia, «Machinability,» [Online]. Available: https://en.wikipedia.org/wiki/Machinability.

[32] J. A. Schey, Introduction to manufacturing processes, McGraw-Hil, 1977.

[33] International Organization for Standardization (ISO/TC 29/SC 9), «ISO 8688-1:1989 Tool life testing in milling - Part 1: Face milling,» 1989. [Online]. Available: https://www.iso.org/standard/16091.html.

[34] International Organization for Standardization (ISO/TC 29/SC 9), «ISO 8688-2:1989 Tool life testing in milling - Part 2: End milling,» 1989. [Online]. Available: https://www.iso.org/standard/16092.html.

[35] H. Shapiro, J. Martens, G. Ericson e W. A. Rohm, «What is Azure Machine Learning Studio?,» 2018. [Online]. Available: https://docs.microsoft.com/en-us/azure/machine-learning/studio/what-is-ml-studio.

[36] D. Sarkar, «Understanding Feature Engineering,» 2018. [Online]. Available: https://towardsdatascience.com/understanding-feature-engineering-part-1-continuous-numeric-data-da4e47099a7b.

[37] Wikipedia, «Feature engineering,» 2018. [Online]. Available: https://en.wikipedia.org/wiki/Feature_engineering.

[38] C. Zhang, X. Yao, J. Zhang e H. Jin, «Tool Condition Monitoring and Remaining Useful Life Prognostic,» 2016. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4934221/.

[39] W. Caesarendra e T. Tjahjowidodo, «A Review of Feature Extraction Methods in Vibration-Based Condition Monitoring and Its Application for Degradation Trend Estimation of Low-Speed Slew Bearing,» 2017. [Online]. Available: https://www.mdpi.com/2075-1702/5/4/21.

[40] Wikipedia, «Discrete Fourier transform,» 2018. [Online]. Available: https://en.wikipedia.org/wiki/Discrete_Fourier_transform.

[41] MathWorks, «Practical Introduction to Frequency-Domain Analysis,» 2018. [Online]. Available: https://it.mathworks.com/help/signal/examples/practical-introduction-to-frequency-domain-analysis.html.

[42] Wikipedia, «Missing data,» 2018. [Online]. Available: https://en.wikipedia.org/wiki/Missing_data.

[43] Microsoft Azure, «Normalize Data,» 2018. [Online]. Available: https://docs.microsoft.com/en-us/azure/machine-learning/studio-module-reference/normalize-data.

[44] S. Kaushik, «Analytics Vidhya,» 2016. [Online]. Available: https://www.analyticsvidhya.com/blog/2016/12/introduction-to-feature-selection-methods-with-an-example-or-how-to-select-the-right-variables/.

[45] H. Midi, S. K. Sarkar e S. Rana, 2013. [Online]. Available: https://www.researchgate.net/publication/261667769_Collinearity_diagnostics_of_binary_logistic_regression_model]. .

[46] Microsoft Azure, «Edit Metadata,» 2017. [Online]. Available: https://docs.microsoft.com/en-us/azure/machine-learning/studio-module-reference/edit-metadata.

[47] Microsoft Azure Documentation, «SMOTE,» 2018. [Online]. Available: https://docs.microsoft.com/en-us/azure/machine-learning/studio-module-reference/smote.

[48] Amazon Machine Learning, «Training ML Models,» [Online]. Available: https://docs.aws.amazon.com/machine-learning/latest/dg/training-ml-models.html.

[49] Microsoft Azure Documentation, «Machine Learning - Train,» 2018. [Online]. Available: https://docs.microsoft.com/en-us/azure/machine-learning/studio-module-reference/machine-learning-train.

[50] Microsoft Azure Documentation, «How to evaluate model performance in Azure Machine Learning,» 2017. [Online]. Available: https://docs.microsoft.com/en-us/azure/machine-learning/studio/evaluate-model-performance.

[51] Microsoft Azure Documentation, «Two-Class Decision Forest,» 2018. [Online]. Available: https://docs.microsoft.com/en-us/azure/machine-learning/studio-module-reference/two-class-decision-forest.

[52] Microsoft Azure Documentation, «Two-Class Logistic Regression,» 2018. [Online]. Available: https://docs.microsoft.com/en-us/azure/machine-learning/studio-module-reference/two-class-logistic-regression.

[53] J. Shotton, T. Sharp, P. Kohli, S. Nowozin, J. Winn e A. Criminisi, «Decision Jungles: Compact and Rich Models for Classification,» 2013. [Online]. Available: https://www.microsoft.com/en-us/research/publication/decision-jungles-compact-and-rich-models-for-classification/?from=http%3A%2F%2Fresearch.microsoft.com%2Fpubs%2F205439%2Fdecisionjunglesnips2013.pdf.

[54] Microsoft Azure Documentation, «Two-Class Decision Jungle,» 2018. [Online]. Available: https://docs.microsoft.com/en-us/azure/machine-learning/studio-module-reference/two-class-decision-jungle.

[55] Microsoft Azure Documentation, «Two-Class Boosted Decision Tree,» 2018. [Online]. Available: https://docs.microsoft.com/en-us/azure/machine-learning/studio-module-reference/two-class-boosted-decision-tree#bkmk_research.

[56] Microsoft Azure Documentation, «Two-Class Neural Network,» 2018. [Online]. Available: https://docs.microsoft.com/en-us/azure/machine-learning/studio-module-reference/two-class-neural-network.

[57] Microsoft Azure Documentation, «Linear Regression,» 2018. [Online]. Available: https://docs.microsoft.com/en-us/azure/machine-learning/studio-module-reference/linear-regression.

[58] Microsoft Azure Documentation, «Boosted Decision Tree Regression,» 2018. [Online]. Available: https://docs.microsoft.com/en-us/azure/machine-learning/studio-module-reference/boosted-decision-tree-regression.

[59] Microsoft Azure Documentation, «Bayesian Linear Regression,» 2018. [Online]. Available: https://docs.microsoft.com/en-us/azure/machine-learning/studio-module-reference/bayesian-linear-regression#bkmk_Notes.

[60] Microsoft Azure Documentation, «Neural Network Regression,» 2018. [Online]. Available: https://docs.microsoft.com/en-us/azure/machine-learning/studio-module-reference/neural-network-regression.

[61] Microsoft Azure Documentation, «Tune Model Hyperparamters,» 2018. [Online]. Available: https://docs.microsoft.com/en-us/azure/machine-learning/studio-module-reference/tune-model-hyperparameters.

[62] P. Gupta, «Cross-Validation in Machine Learning,» 2017. [Online]. Available: https://towardsdatascience.com/cross-validation-in-machine-learning-72924a69872f.

[63] S. Ray, «Improve Your Model Performance using Cross Validation,» 2018. [Online]. Available: https://www.analyticsvidhya.com/blog/2018/05/improve-model-performance-cross-validation-in-python-r/.

[64] J. Brownlee, «A Gentle Introduction to k-fold Cross-Validation,» 2018. [Online]. Available: https://machinelearningmastery.com/k-fold-cross-validation/.

[65] Microsoft Azure Documentation, 2018. [Online]. Available: https://docs.microsoft.com/en-us/azure/machine-learning/studio-module-reference/partition-and-sample.

[66] Microsoft Azure Documentation, «Cross-Validate Model,» 2018. [Online]. Available: https://docs.microsoft.com/en-us/azure/machine-learning/studio-module-reference/cross-validate-model.

[67] X.-S. Si, W. Wang, C.-H. Hu e D.-H. Zhou, «Remaining useful life estimation – A review on the statistical data driven approaches,» 2011. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S0377221710007903.

[68] B. Sniderman, M. Mahto e M. Cotteleer, «Deloitte Insights,» 2016. [Online]. Available: https://www2.deloitte.com/insights/us/en/focus/industry-4-0/manufacturing-ecosystems-exploring-world-connected-enterprises.html.

Appendices

Appendix 1 – Classification of milling machines

Milling machines can be further classified according to their orientation to workpiece, their degree of motion and their automation into the following: knee-and-column, bed type, turret type, ram type, planer type, tracer mills, and CNC milling machines.

Knee-and-column

A knee-and-column mill refers to any milling machine with a vertical adjustable worktable resting on a saddle supported by a knee. A knee is casting that moves vertically along a column/up and down on a column and can be regulated (depending on the operation) to allow a customizable workspace.

Ram type

A ram-type milling machine is characterized by a spindle mounted on a movable ram on the column. A ram is an overhanging arm generally used in vertical mills: one end of the arm is attached to the column and other end to the milling head. It can be moved on the column in transverse direction, by enabling to adjust the distance between the cutting tool and the workpiece. Two popular ram-type milling machines are the universal milling machine and the swivel cutter head ram-type milling machine.

Bed type

In a bed mill, the spindle can be raised or lowered by moving it parallel in its personal axis, while the table, directly located on the bed, is allowed to move only horizontally. The machine doesn't not include the knee part and the combination of these movements allows to mill the workpiece to different shapes and depths.

Turret

A turret mill, also known as Bridgeport-type mill, employs a stationary spindle which is fixed in a certain position during the cutting operation, combined with a table moving both perpendicular and parallel to the axis of the spindle to be able to bring the workpiece up to the cutting surface (in order to carry out the cutting operation).

Planer type

The planer-type milling machine is similar to the bed-type mill except it can be equipped with various cutters and spindle heads that allow to perform a wider range of milling actions simultaneously.

Tracer mill

A tracer mill is able to reproduce parts based on a master model by synchronizing the tracing unit. It enables to develop difficult shapes and it is mostly used in automotive and aerospace industries.

CNC

Computer numerical control milling machine is the most modern and versatile milling machine in which the cutter path is controlled by alphanumerical data rather than a physical template. It is characterized by a spindle moving along all three direction and a table with a 360-degree rotation where the movements are commanded by a computer to which the machine is directly connected and are controlled by an ISO program that allows it to perform the operations automatically. CNC milling machines are especially suited in complex operations in which two or three axes of the both cutting tool and table must be simultaneously controlled to achieve the required cutter path, or when exact repetition is needed.

Appendix 2 – Material Removal Rate

Material removal rate (MRR or Q) in milling is determined using the product of the cross-sectional area of the cut and the feed rate. Therefore, if a milling operation is cutting a workpiece with width W (mm) at a depth D (mm) and feed rate FR (mm/min), the rate is measured in mm^3/min and it is computed as follow:

$$MRR = W \times D \times FR$$

Appendix 3 – Detailed Experiment Description

Mounted to the table is an acoustic emission sensor model WD 925 (PHYSICAL ACOUSTIC GROUP, frequency range up to 2MHz). The acoustic emission sensor is glued to a custom made base which in turn is attached to the clamping support. The layout of the sensors on the clamping device is shown in ***Figure 77***.

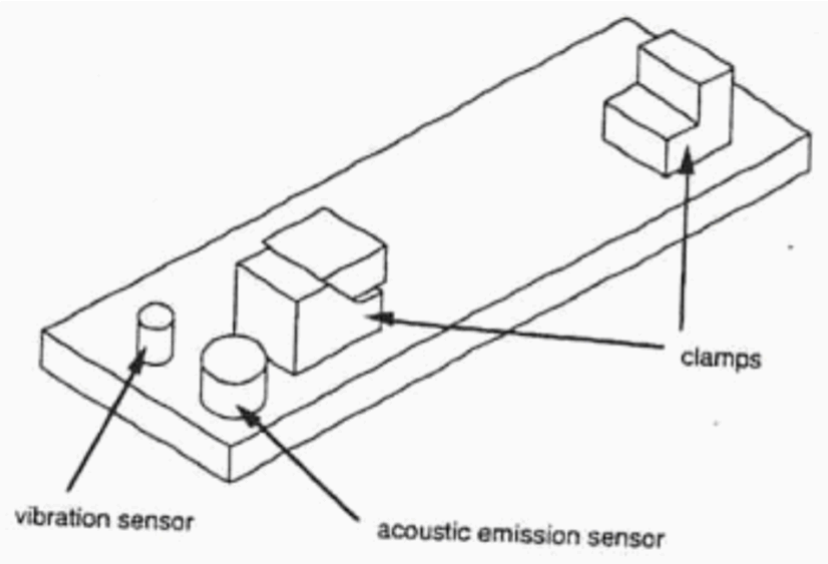


Figure 77*: Clamping device with mounted sensors*

The signal from the acoustic emission sensor goes into the single ended terminal of an acoustic emission preamplifier (DUNEGAN/ENDEVCO, model 1801 with integrated 50KHZ high pass filter). The signal is then amplified by a dual amplifier DE model 302A (DUNEGAN/ENDEVCO). The signal is then fed into the RMS meter which is a custom made device built by the LMA of the University of California at Berkeley. The time constant is set to 8.0ms. The signal is then fed into the PHOENIX CONTACT UMK-SE 11,25 cable which feeds the signal into a MIO-16 high speed data acquisition board (National Instruments). The data acquisition board is mounted in a IBM PC 486DX/2-66.
Also mounted on the clamping device on the table is a vibration sensor, an accelerometer (model 7201- 50, ENDEVCO) with a frequency range up to 13KHz. Its signal is fed into an ENDEVCO 104 charge amplifier with sensitivity 5.71 and 100mV/g output. The signal is then fed into an ITHACO 4302 DUAL 24dB/octave filter with corner frequencies 400 Hz and 1KHz.

Following is a RMS meter same make and settings as described earlier. The signal is then fed then through the PHOENIX CONTACT UMK-SE 11,25 cable connector into the high speed data acquisition board of the computer.
The same vibration sensor that is mounted on the table is also attached to the spindle into a pre-existing threaded hole close to the tool. The signal follows the same path as described for the other vibration sensor except that the signal is fed into the PHOENIX CONTACT cable connector.
Signals from another acoustic emission sensor mounted into another threaded hole on the spindle next to the tool is fed into the differential terminal of an acoustic emission preamplifier model 1801 (DUNEGAN/ENDEVCO) and then follows the same path as outlined for the other acoustic emission sensor.
A OMRON K3TB-A1015 current converter, powered by a HP 6237B triple output power supply providing 15V, feeds the signal from one spindle motor current phase into the cable connector.
A model CTA 213 current sensor (Flexcore Div. of Marlan & Associates, Inc.) which uses the same phase of the spindle motor current is fed into the cable connector.
Terminals 33 (ground) and 38 (+5V) are used with a switch to trigger the data acquisition.
With industrial applicability in mind, a 70mm face mill with 6 inserts (***Figure 78***) was chosen as the tool. The inserts KC710 was selected based on the recommendations for roughing (Kennametal, 1985). KC710 is coated with multiple layers of titanium carbide, titanium carbonitride, and titanium nitride (TiC/TiC-N/TiN) in sequence. These layers retain the toughness of tungsten carbide but have improved resistance to cratering and edge wear. At the same time, they have the advantage of titanium carbide plus reduced face friction. This insert is recommended for heavy roughing.

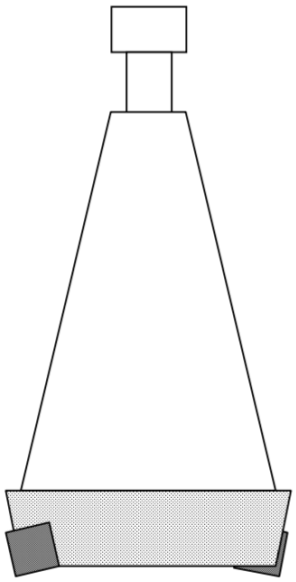
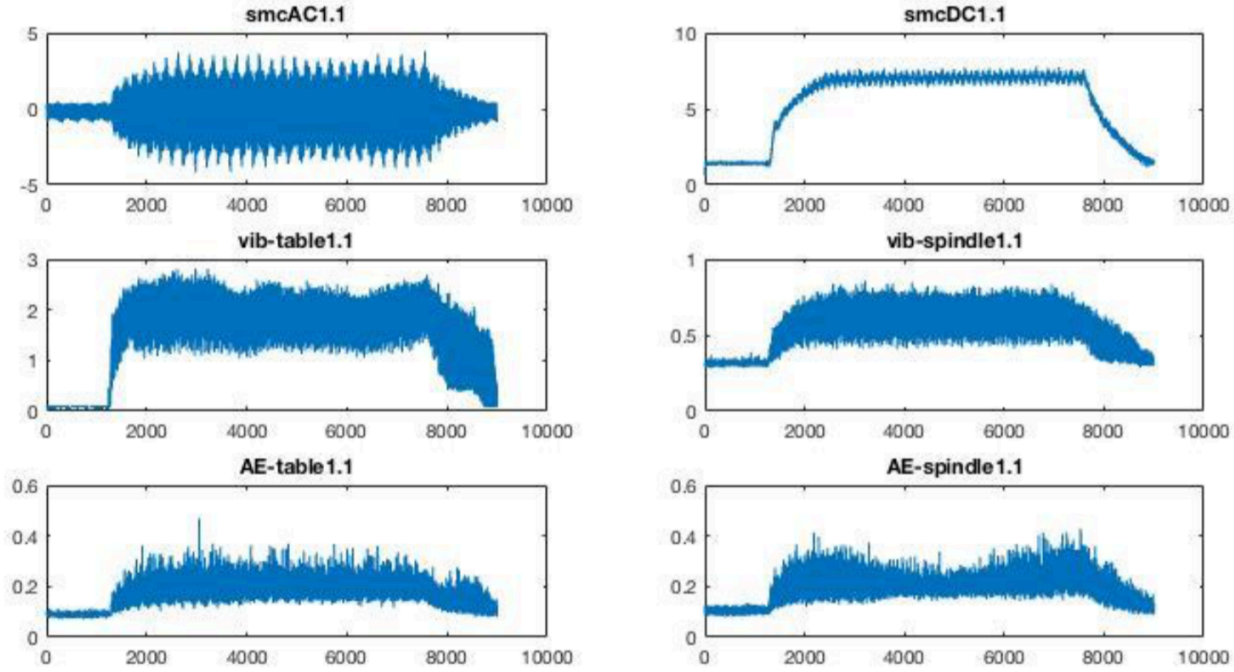


Figure 78 *- Schematic of tool and inserts of face milling*

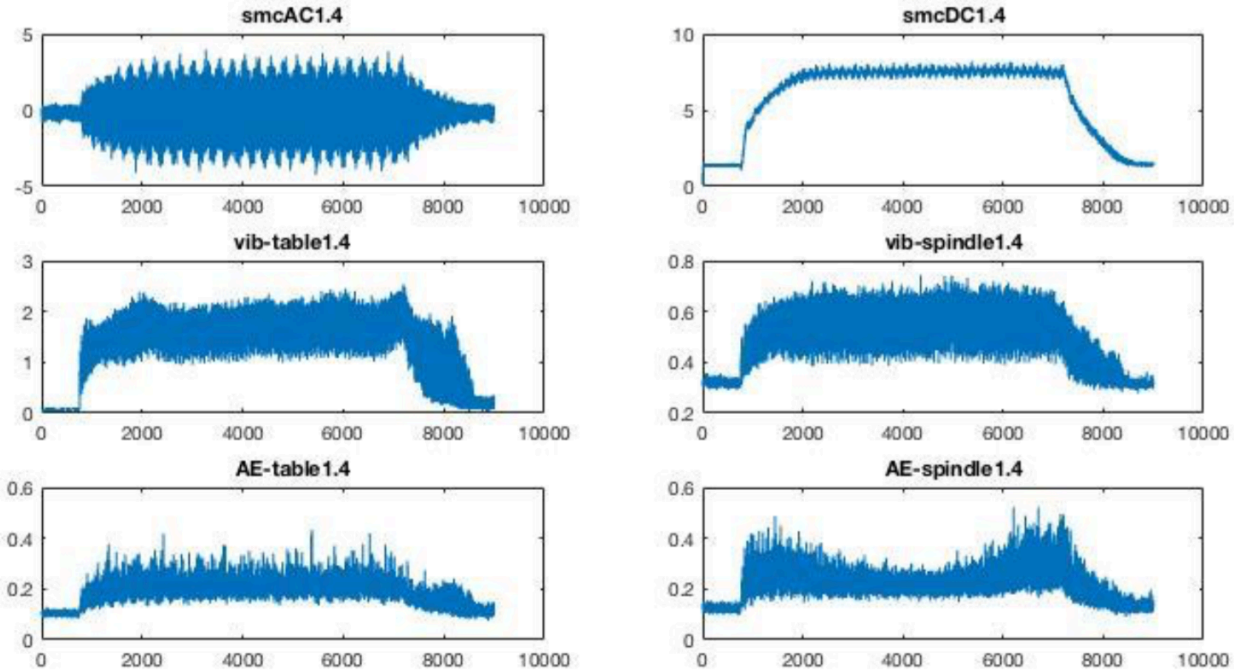
Appendix 4 – Typical signals from different sensors

The following figures show some typical signals from the different sensors.

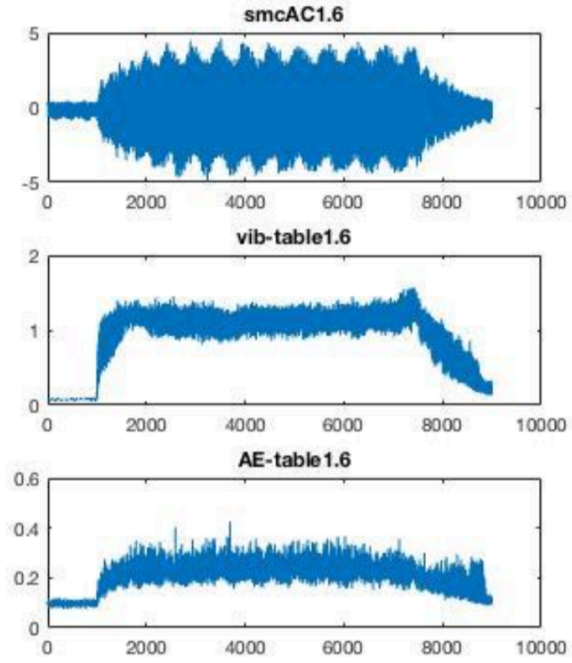
Case 1, Run 1

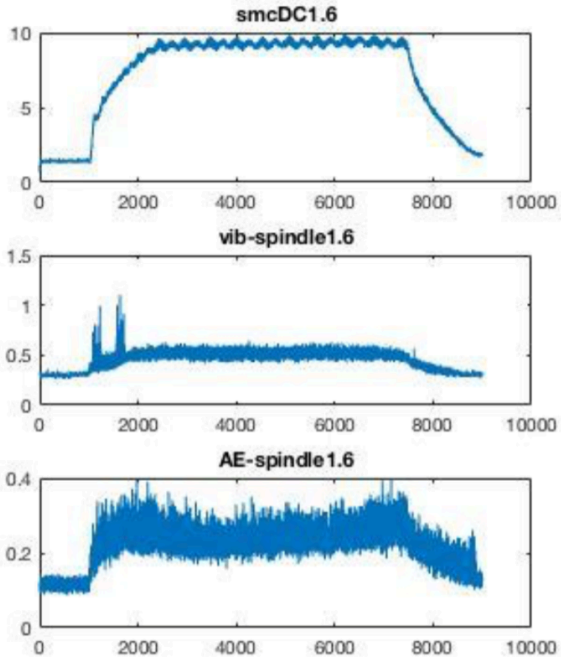


Case 1, Run 4

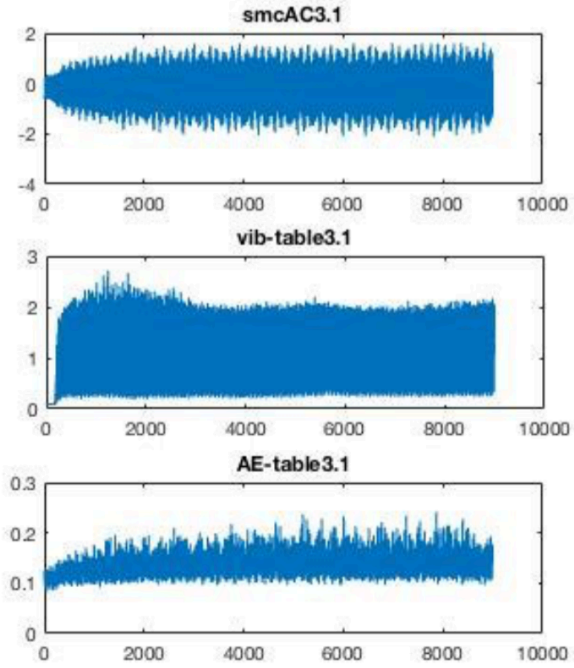


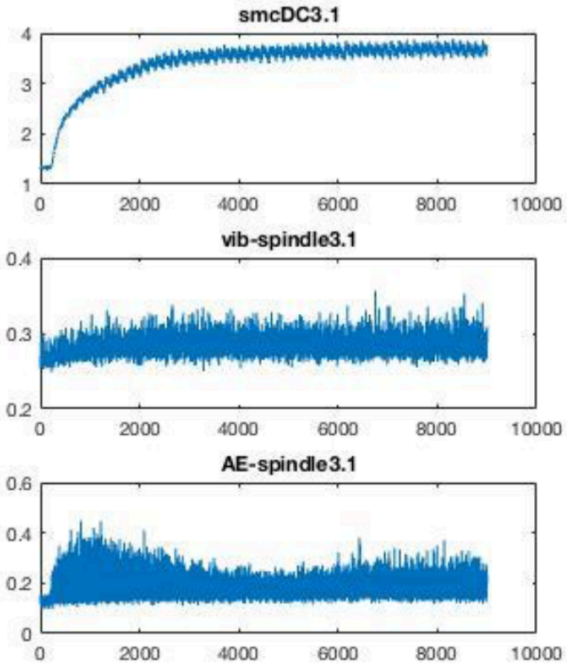
Case 1, Run 6



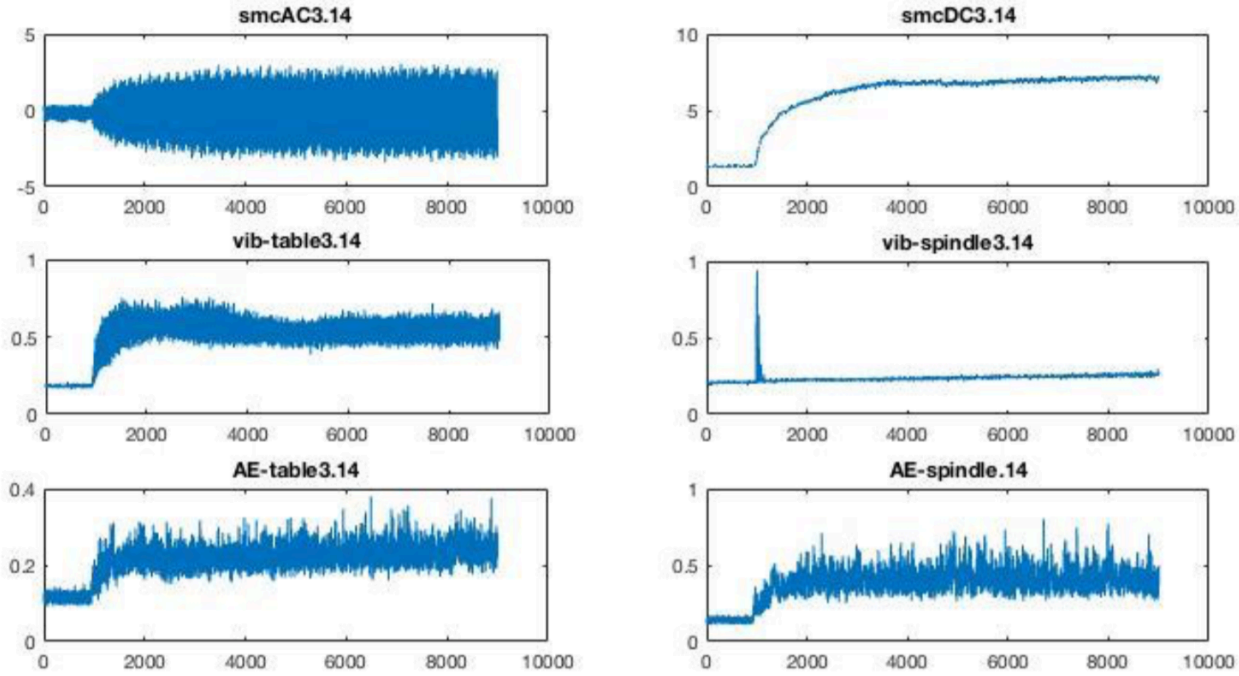


Case 3, Run 1

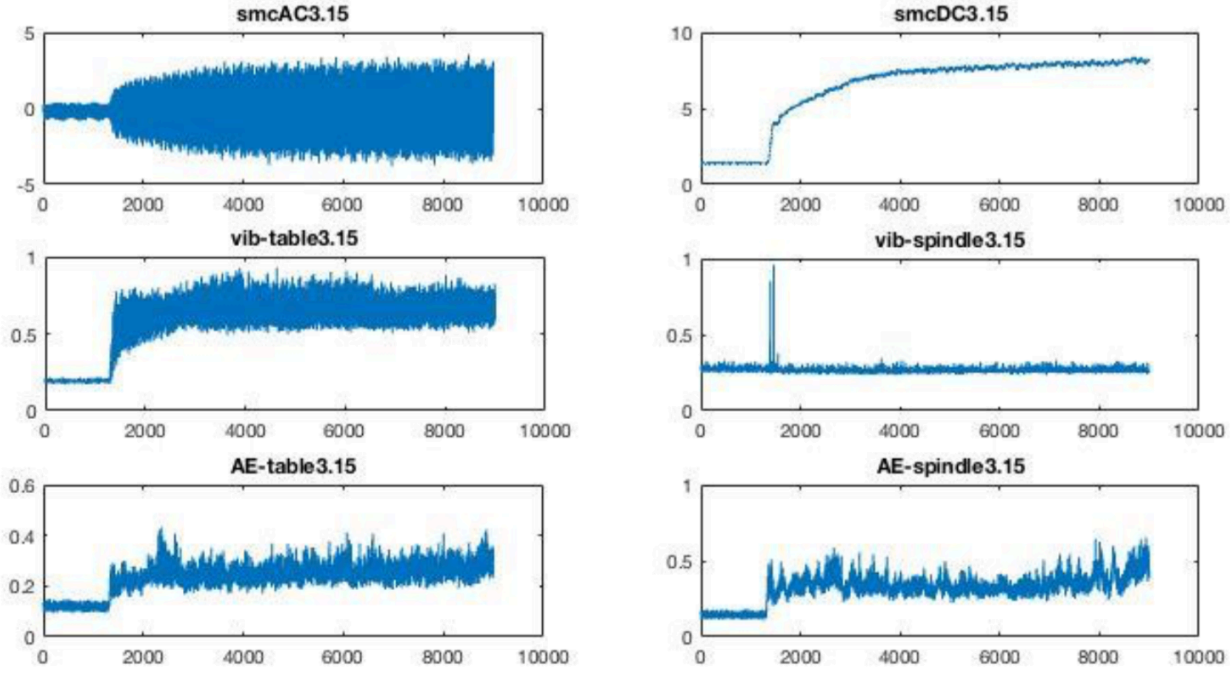




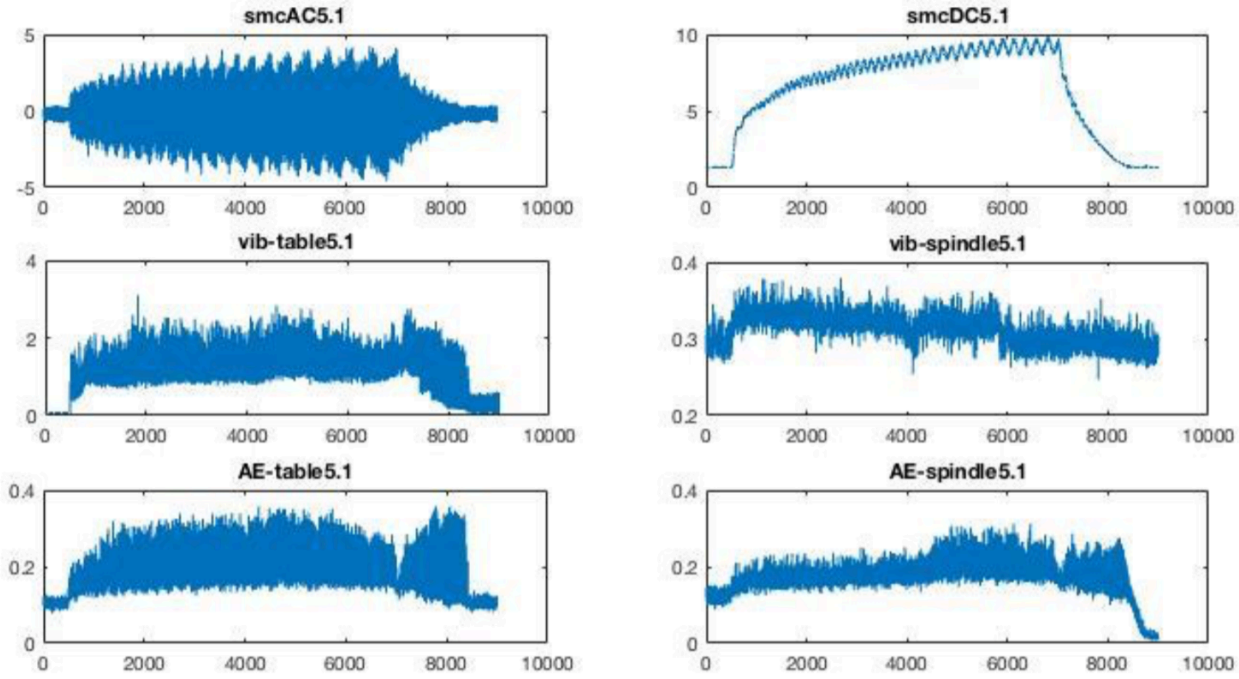
Case 3, Run 14



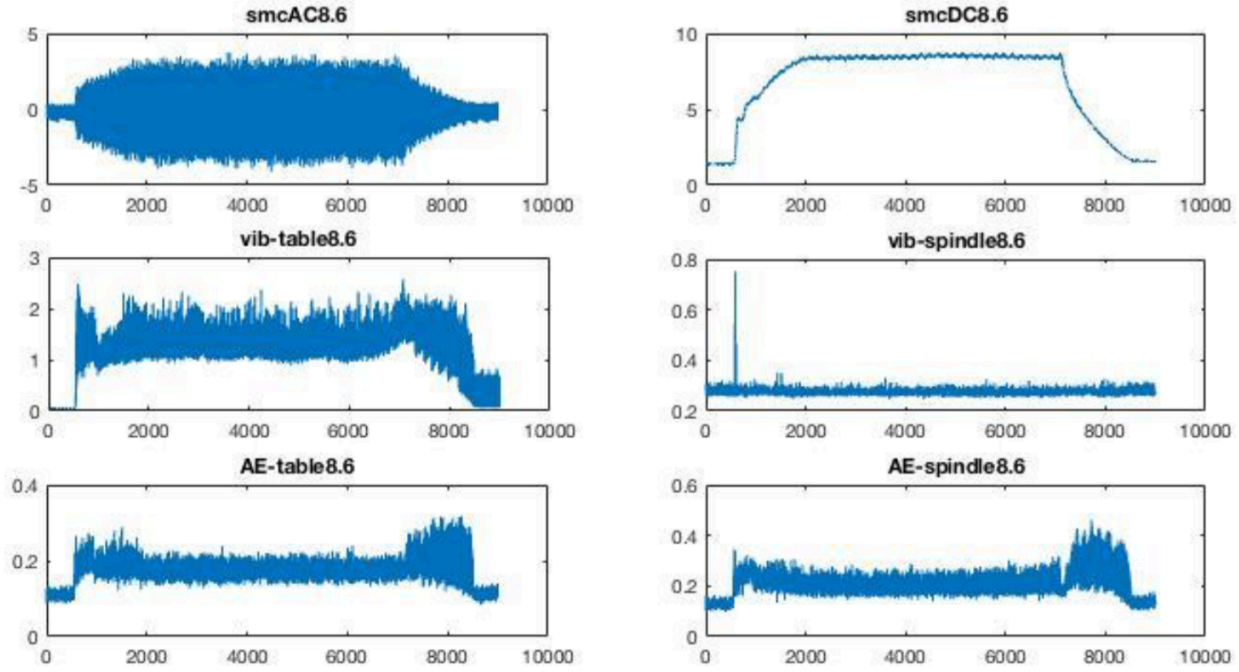
Case 3, Run 15



Case 5, Run 1



Case 8, Run 6



Appendix 5 – Fourier Transform

The Fourier Transform (FT) is a tool that breaks a waveform (a function or signal) into an alternate representation, characterized by sine and cosines. The Fourier Transform shows that any waveform can be re-written as the sum of sinusoidal functions. It is used to decompose a

function of time into the frequencies that make it up. he Fourier transform of a function of time is itself a complex-valued function of frequency, whose absolute value represents the amount of that frequency present in the original function, and whose complex argument is the phase offset of the basic sinusoid in that frequency.

The Frequency transform is defined by the set of equation:

- $F(v) = \int_{-\infty}^{+\infty} f(t) e^{-2\pi i v t} dt$

which is usually known as the ***forward transform,*** where
  - *F(v)* is the complex-valued Fourier Transform, in function of the frequency *v*, also called spectrum of the signal *f(t);*
  - *f(t)* is the signal to be transformed, in function of the time t;
  - *i* indicates the imaginary argument of the exponential.

- $f(t) = \int_{-\infty}^{+\infty} F(v) e^{2\pi i t v} dv$

which is the ***inverse transform:*** it states that the original signal *f(t)* can be reconstructed from its spectrum.

Appendix 6 – Decision Trees

Decision trees in general are non-parametric models, meaning they support data with varied distributions. In each tree, a sequence of simple tests is run for each class, increasing the levels of a tree structure until a leaf node (decision) is reached. Decision trees in general have many advantages for classification tasks:

- They can capture non-linear decision boundaries.
- You can train and predict on lots of data, as they are efficient in computation and memory usage.
- Feature selection is integrated in the training and classification processes.
- Trees can accommodate noisy data and many features.
- They are non-parametric models, meaning they can handle data with varied distributions.

However, simple decision trees can over fit on data, and are less generalizable than tree ensembles.

List of Figures

Acknowledgements

I would like to express my deep and sincere gratitude to my Thesis advisor, Professor Franco Lombardi, for his professional guidance and valuable support. He gave me the opportunity to explore this topic and provided me with constructive recommendations throughout the research.

Special thanks should be given to Dott. Emiliano Traini, my co-advisor who has assisted me during the development of this Thesis and has contributed to it with a major impact. I have very much appreciated his willingness to give his time so generously, his useful advises and especially patience.

Thank you.

Chiara Morresi