POLITECNICO DI TORINO

Automotive Engineering MSc

Master's Thesis

Artificial Intelligence for Vehicle Engine Classification and Vibroacoustic Diagnostics

in collaboration with
DeepTech Lab at Michigan State University
and Fiat Chrysler Automobiles

Academic supervisors
Prof. Giovanni Belingardi
Prof. Daniela Misul
Prof. Joshua Siegel

Candidate
Umberto Coda

14 October 2020

Contents

List of Figures

Abstract

This Thesis aims at solving the initial and fundamental step in the context of vibroacoustic diagnostics applied to vehicles: engine classification. This goal is supported by the creation of a flexible and user-friendly framework to be used for further development. In Chapter 1, the smartphone-based vibroacoustic diagnostics process is outlined, and the need for engine classification in this context is motivated. In Chapter 2, the process of data collection and feature extraction is presented together with examples of framework usage and dataset exploration. In Chapter 3, the Machine Learning procedure is explained alongside with some insights on the functioning of the classification algorithms. In Chapter 4, diagrams present the high-level operational flow of the framework. In Chapter 5, results are evaluated for three different labels: aspiration type (turbo), fuel, and cylinders amount. Those labels are predicted sequentially in order to exploit the correlation among them and to improve performance. This enables AI to reach ROC-AUC higher than 93% in most cases. Finally, I will provide some next steps that may enable the extension of this framework to different diagnostics fields, pursuing a universal vibroacoustic diagnostics vision.

Chapter 1

Introduction and Motivation

The automotive world is changing, and there is increasing concern about vehicles' environmental impact, particularly those with internal combustion engines. As a result, there are increasingly efficiency-improving systems within vehicles. One significant contributor to lifetime efficiency is the availability of in-vehicle diagnostic systems that report faults early and accurately, which motivates owners and operators to seek out preventative or corrective maintenance, enhancing safety and reducing operating costs.

At the same time, mobility is evolving, transitioning from the need to own a car towards mobility-as-a-service (MaaS). Today, vehicle sales are slowing despite continued high mobility demand: the average vehicle age and lifetime miles travelled are increasing, particularly in developing countries [7, 127], and shared mobility services, car rentals, and "*robotaxis*" are emerging. The resultant increased utilization requires enhanced fleet data generation and management capabilities. Diagnostics are also key to fleet management and may infer a vehicle's condition based on observed *symptoms* indicating a technical state [19].

In this introductory chapter, I motivate our work, its context, and long term goals. This Masters Thesis is the first step in a long-term vision. I first explain the need for Vehicle Diagnostics and how the "smartphone world" might enable an acceleration in future innovation. Furthermore, I address one aspect of vehicle diagnostics, namely vibroacoustic diagnostics, and more specifically acoustic diagnostics, as well as some prior work that has been done in this field by other researchers. This is to state that it is an already explored environment, but where the potential to grow is considerable. In fact, I will then present an idea of framework to converge all those optimal projects into a universal concept of diagnostics, where the context in which we observe the system plays a crucial role. Finally, I will explain in which part of this vision this Master Thesis is going to be located, and the secondary goal set for this project.

This chapter is based upon a survey paper article we prepared for submission, the preprint of which is available at `https://arxiv.org/abs/2007.03759` [104]

1.1 Existing Methods for Vehicle Diagnostics

Diagnostics are important for monitoring vehicle, environment, and occupant status (e.g. component wear, road conditions, or driver alertness). Historically, these diagnostics draw upon in-situ sensors and computation to develop "On Board Diagnostics".

1.1.1 On-Board Diagnostics (OBD)

On-Board Diagnostic (OBD) systems present on vehicles sold since 1996 [102] are an automated control system utilizing distributed sensing across a vehicle's embedded systems as a technical solution for measuring vehicle operational parameters and detecting, reporting, and responding to faults.

Sensors may capture signals (e.g. vibration, or noise) and algorithms extract and process features, typically comparing these "signatures" against a library of previously-labeled reference values indicating operating state and/or failure mode [34]. If a "rule" is triggered, an indicator is set to notify the user of the fault, and additional software routines may run to minimize the impact of the fault until the repair can be completed (e.g. by changing fuel tables). OBD data have also been used to enable indirect diagnostics, for example using the measured rate of change of coolant temperature to infer oil viscosity and therefore remaining useful life through constitutive relationships and fundamental process physics [103].

1.1.2 Non-OBD Systems

OBD is effective at detecting many fault classes, particularly those related to emissions [30]. However, some failure modalities may not be detected by OBD, or may be detected with slow response time or poor classification accuracy, because:

- **Incentive misalignment** discourages the use of high-quality (costly) sensors, leading manufacturers to source the lowest-cost sensor capable of meeting legislative standards. Relying upon the data generated by these sensors leads to "GIGO" (Garbage In, Garbage Out) [108]
- Diagnostics may be tailored to **under-report non-critical failures** to improve customer satisfaction, brand perception, and reliability metrics relative to what might be experienced with an "overly sensitive" implementation
- OBD systems are **single-purpose**, meaning they correctly identify the symptoms of the faults for which they were designed, but small performance perturbations may not be detected. For example, a system designed to enhance emissions may monitor engine exhaust gas composition continuously, but will not indicate wear or component failures leading to increased emissions until a legal threshold requiring occupant notification is surpassed [30].

OBD's deficiencies are amplified by an ever-aging vehicle fleet [7, 127], though older cars stand to gain the most from the incremental reliability, performance and efficiency improvements enabled by adaptive and increasingly sensitive diagnostics. While newer vehicles may have the ability to update diagnostic capabilities remotely via over-the-air-updates [109], older vehicles may lack connectivity or the computational resources necessary to implement these advanced algorithms. Further, the sensor payload in the incumbent vehicle fleet is immutable, with no data sources added post-production. Therefore, the vehicles most in need of enhanced diagnostics are the least-likely to support them.

For these reasons, there is a need for updatable off-board diagnostics capable of sensitive measurement, upgradability, and enhanced prognostic (failure predictive) capabilities. A low-cost approach, even if imperfect, will enhance vehicle owners' and fleet managers' ability to detect, mitigate, and respond to faults, thereby improving fleet-wide safety, reliability, performance, and efficiency.

1.2 Diagnostics Opportunities in the Mobile Revolution

As the need for enhanced fleet-wide utility grows, so to does the challenge of monitoring increasingly diverse vehicles and their associated, complex subsystems. The same enhancements driving the growth of in-vehicle sensing and connectivity have simultaneously empowered a parallel advance: namely, the growing capabilities of personal mobile devices. 70% of the world's population is now using smartphones [57] possessing rich sensing, high-performance computation, and pervasive connectivity - capabilities enabling a diagnostic revolution.

1.2.1 Smartphone Sensing Capabilities

While condition monitoring equipment has historically been cost-prohibitive, contemporary mobile devices include more sensors than ever, facilitating inexpensive and performant data capture with minimal complexity (see [62] for an example of pervasive mobile sensing as applied to human activity recognition). Initially, mobile device sensors served to support core device functions, with software libraries easing access to their data and widening their use cases into third-party applications [49] and analytics. Today, mobile device sensor support continues to grow, and even older devices may add external sensors through serial, Bluetooth [50] or Wi-Fi connectivity. Modern devices feature accelerometers, cameras, magnetometers, gyroscopes, GPS, proximity, light sensors and microphones that are accurate, precise, high-frequency, efficient and low-cost [57, 49]. These capabilities have enabled the large-scale use of mobile systems as sensing devices in two-thirds of experimental research studies where such sensors are required [63].

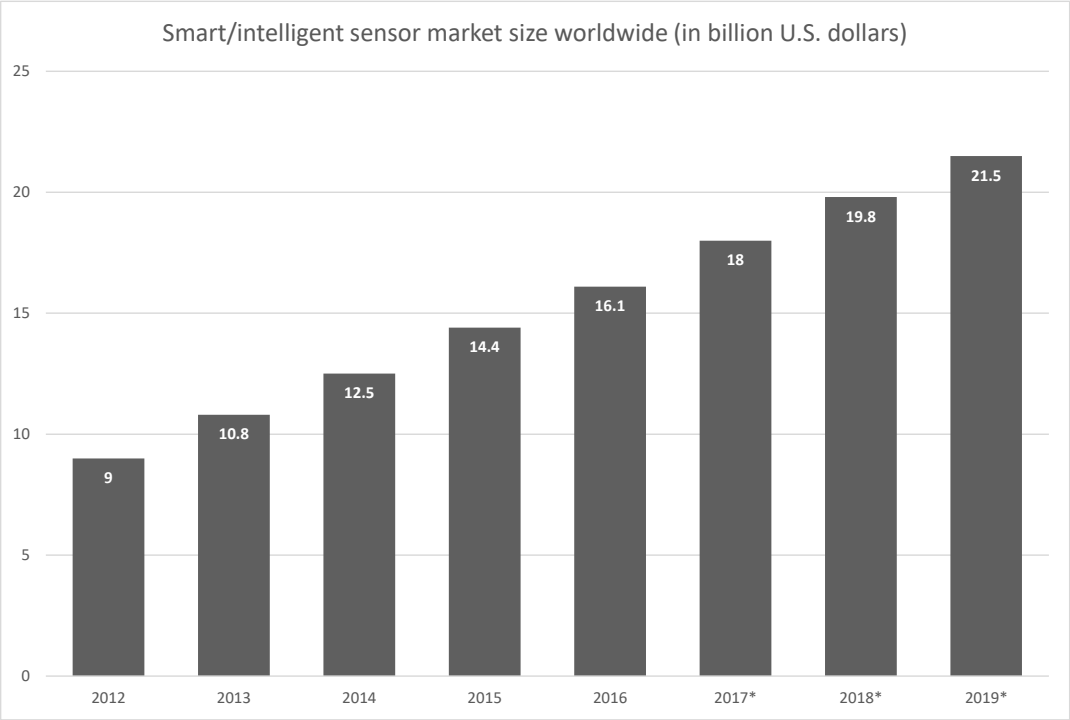


Figure 1.1. Size of sensor market worldwide in billion U.S. dollars. Years with "*" are forecasts. Source: *statista.com* [116]

The use of mobile devices as pervasive sensors has an added benefit of *embodying intelligence*. That is, untrained users with access to the appropriate applications can make technically-sound judgments, identifying even those problems for which the device user has no prior knowledge of its existence – and no awareness that the application is scanning for such faults [105, 110]. This reduces the training burden for mechanics and fleet managers, and makes operating larger and more-diverse fleets feasible.

By shifting intelligence from cost-, energy- and performance-constrained, in-vehicle hardware into third-party devices, the enabled algorithmic models may also be made more computationally-intensive, more easily updated, provided with access to higher-quality (and evolving) sensors and data, aggregable at a fleet level, and airgapped from critical vehicle hardware and software.

This concept has been proven across domains. For example, introducing energy into a physical sample and studying the transient response across diverse sensors has been used to enable an individual to "tap" an object in order to determine its class [47].

1.2.2 Smartphone Computation and Connectivity Capabilities

Pervasive connectivity enables diagnostics to utilize diverse data sources, and supports off-line processing and the creation of diagnostic algorithms capable of adapting over time. This is a result of having access to increased computational resources, enhanced storage capabilities, and richer fingerprint databases for classification and characterization. It also means that "fault definitions" may be updated at a remote endpoint, such that diagnostics may improve performance over time without requiring in-vehicle firmware upgrades, over-the-air (OTA) or otherwise.

To this end, mobile phone computing power has recently increased, with the new mobile GPU Adreno 685 [118] reaching the computational power of Intel's 1998 ASCI Red supercomputer [90]. Networking capabilities have similarly grown, allowing for inexpensive global connectivity.

While some vehicles offer connectivity [102] which may be used to support OBD's evolution, the use of a third-party devices has an additional benefit to manufacturers: with mobile devices, the users, not the manufacturer, pays for bandwidth and hardware capability upgrades over time.

Mobile phones can augment or supplant the data generated by OBD, fusing in-vehicle sensing with smartphone capabilities to enable richer analytics.

Smartphones offer clear benefits over (or in conjunction with) on-board systems, particularly when constraints such as battery life, computation, and network limitations are thoughtfully addressed [57], and present a compelling enhancement over automotive diagnostics' "business as usual".

1.2.3 Off-board Smartphone-based Diagnostics

There is an opportunity to use users' mobile devices as "pervasive, offboard" sensing tools capable of real-time and off-line vehicular diagnostics, prognostics, and analytics. The capabilities of such tools are growing and they may soon supplant on-board vehicle diagnostics entirely, moving diagnostics from low-cost OBD hardware, frozen at time of production, to performant, extensible, and easily-upgradable hardware and adaptive software algorithms capable of improving over time. The advantage of this approach goes beyond performance improvements to increase flexibility, enabling diagnostics that address any vehicle – new or old, connected or isolated – taking advantage of rich data collection, better-characterizable sensors, and scalable computing.

Many effective "pervasive" sensing technologies revolve around the concept of remote sensing of sound and vibration utilizing onboard microphones and accelerometers, sensors core to mobile devices. This class of sensing is termed "vibroacoustic sensing," as it captured vibration and acoustic emissions of an instrumented system. Those consideratios led us to consider such sensors to improve diagnostics of

vehicles in a flexible and revolutionary: it is called Vibroacustic Diagnostics.

1.3 Vibroacoustic Diagnostics

Vibroacoustic diagnostic methods originate from specialists troubleshooting mechanisms based on sound and feel, dating back to well before the time of Steinmetz's famous (and perhaps apocryphal) chalk "X" [114].

The vibroacoustic diagnostic method is non-intrusive, as sound can traverse mediums including air and "open" space and vibration can be conducted through surfaces without rigid mounting. It is therefore an attractive option for monitoring vehicle components [34]. Experientially-trained mechanics may be highly-accurate using these methods, though there may be future specialist shortages [12] leading to demand for automated diagnostics.

There has been work to automate vibroacoustic diagnostics. Sound and vibration captured by microphones and accelerometers, for example, has been used as a surrogate for non-observable conditions including wear and performance level [19]. Low-cost microphones have been used to identify pre-learned faults and differentiate normal from abnormal operation of mechanical equipment using acoustic features, providing a good degree of generalization [126]. Like sound (which itself is a vibration), vibration has been used as a surrogate for wear, with increasing intensity over time reasonably predicting time-to-failure [26]. In fact, accelerometers have also been used to infer machinery performance using only vibration emissions as input [46]. Vibrational analysis may be coupled with OBD systems to improve diagnostic accuracy and precision, [72], or used in lieu of onboard measurements.

Vibroacoustics, counterintuitively, may be more precise than OBD because air-gaps provide a mechanism for isolating certain sounds and vibrations from sensors. While vibration may therefore be used to capture "conductive" time-series data, acoustic signals may be preferable in certain applications as the mode of transmission may serve to pre-condition input data and may transmit information related to multiple systems simultaneously [34].

Some diagnostic fingerprints are developed based on understanding of the underlying physical process, whereas others are latent patterns learned from experimental data collection [34].

1.3.1 Physics-based approach

Real-world systems have inputs including energy, materials, control signals, and perturbation. It is possible to directly-measure inputs, outputs, and machine performance, but indirect measurement of residual processes (heat, noise, etc.) may be less-expensive and equally-useful diagnostically [19].

Vibration and sound are energy emissions stemming from mechanical interactions. Due to inherent imperfections, even rotating assemblies, such as gear meshes, may be modeled as a series of repeated impact events producing a characteristic noise or lateral motion [29].

If one understands these processes, it becomes possible to model them and to engineer a series of features useful for system characterization. Modelling and processing techniques include *frequency analysis*, *cepstrum analysis*, *filtering*, *wavelet analysis*, among others. These generate features that are more-robust to small perturbations and therefore resistant to overfit when used in machine and deep learning algorithms. Other features describing waveforms may provide better discriminative properties. The features selected are informed by the engineer's knowledge of the physical process and what she or he believes likely to be informative in differentiating among particular states. Careful feature selection has the potential to improve diagnostic performance as well as reducing computation time, memory and storage requirements, and enhancing model generalizability.

1.3.2 Vibroacoustic Challenges

Though VA is a compelling solution, it requires significant and diverse training data to achieve high performance and classification or gradation algorithms may be computationally-intensive and tailored to highly-specific systems. Accepting minimally-reduced performance to enhance algorithm generalizability and reduce computational performance, and/or shifting computation to scalable Cloud platforms, has the potential to make VA more powerful as a condition monitoring and preventative maintenance tool for vehicles and other systems.

At the same time, smartphone processing power is increasing, and it may be possible to use a mobile device as a platform for real-time acoustic capture and processing, as demonstrated by Mielke [77], and to do the same for vibration capture and analysis [26].

Algorithms trained on few measurements may be inherently unstable, so multi-device crowdsourcing improves acoustic measurement classification confidence [131]. Diverse, distributed devices lead to better training data and enhanced confidence in diagnostic results, though it is challenging to balance accuracy with system complexity [37] and to ensure samples represent usable input signals rather than background noise [34].

These challenges can be managed with careful implementation, helping pervasively-sensed VA attain strong performance when utilizing system-specific models for diagnostics and proactive maintenance within automotive and other contexts.

1.4 Impact of Artificial Intelligence

Vibration can be described by a mono-dimensional signal expressing the amplitude as a function of time. Sound is a specific component of vibrational signals, characterized by frequencies inside human ear's rage (20Hz-20kHz). Sound is a generated by vibration and/or impact of objects that transmits their motion to a medium (e.g. air), generating a pressure wave. This wave is spread through the medium and can be captured by some sensors and encoded into a digital signal. In digital terms, this signal is composed by a sequence of values defining the amplitude of the signal in that time instant. In this regard, sampling rate plays a crucial role in converting the analogical signal to digital one, as explained by the Nyquist–Shannon sampling theorem. The analysis of a digital signal is generally done with two main general techniques, or a combination of the two:

1. **Signal Processing:** It is a technique which deals with the analysis of digitized and discrete sampled signals. It requires a lot of supervision, and each rule has to be specifically defined by the designer. It may become very complex when the relationships are non-linear, when the amount of data is high or when it is unclear which factors are important to determine the output.

2. **Machine Learning** on the other hand, is a technique allowing the algorithm to improve automatically through experience and to build the mathematical model from the sample data (see Section 2.1). The supervision requires is less, it can handle non-liner relationships, but it is more difficult to understand why the algorithm made some choices.

Since this problem is rather hard for a human to solve, if he had to look at the digital signal alone, the majority of researches done in vibroacoustic diagnostics make use of some kind of machine learning algorithm. Signal processing is still very important as a preprocessing tool to help machine learning algorithms to perform better. The reasons why artificial intelligence has become more mainstream in the last years is due to three main aspects:

- Following some modified version of the Moore's Law, the amount of computing power has risen very fast in the last years, so that nowadays everyone is able to run those algorithms on its personal computer, or even on its mobile phone.

- AI algorithms require a relatively big amount of data, that was not possible to imagine just some years ago. Data are nowadays more available and easier to collect. This is mainly ensured by the amount of mobile devices around equipped with a lot of sensors, as explained in Section 1.2. All data of our work have been collected by smartphones' microphones, for example, and the equipment required for such procedure would have been too expensive and complicated.

- Availability of Open Source software and libraries, allowing everyone to get access to those powerful tools.

1.5 Prior Art

Algorithms of Artificial Intelligence have proven their way into signal analysis thanks to their flexibility and powerful results. Researchers across the world developed systems to monitor the three main players when it comes to drive a car:

- Vehicle identification and component-level diagnostics (Section 1.5.1)
- Vehicle Operating State, e.g. whether it is moving, the position of the throttle, steering, etc. (Section 1.5.2)
- Occupant and driver behavior monitoring and telemetry (Section 1.5.3)
- Environmental measurement and context identification (Section 1.5.4)

1.5.1 Vehicle Condition

Vehicles are increasingly complicated, though their mechanical embodiment typically comprises systems that translate and rotate, vibrating through use. There is a corpus of prior art focused on analysis of such systems.

One example comes from Shen, et al., who developed an automated means of extracting robust features from rotating machinery, using an auto-encoder to find hidden and robust features indicative of operating condition and without prior knowledge or human intervention [101]. Mechanical systems wear down, leading to different operating states that a diagnostic tool must be able to detect in order to time preventative maintenance properly. To address this need, a "sound detective" was created to classify the different operating states of various machines [75].

Another approach to vibrational analysis utilizes constrained computation and embedded hardware. A Raspberry Pi was used to diagnose six common automotive faults using deep learning as a stable classification method [67].

Engine and Transmission

Automotive engines, as with other reciprocating machinery, are difficult to diagnose because of the coupling among subsystems. Engines generate sound stemming from intake, exhaust, and fans, to combustion events, valve-train noise, piston slap, gear impacts, and fuel pumping. Each manifests uniquely and transmits across varied transmission-pathways, as examined in this comprehensive survey related to the use of vibroacoustic diagnostics for ICE's [34]

For this reason, audio may be more suitable than vibration for identifying faults as the air-transmission path eliminates some system-coupling, making it easier to disaggregate signals [34].

In many studies, complete and accurate physical fault models are not available, so signal processing and machine learning techniques help improve classification performance. There are techniques for signal decomposition to better-highlight and associate features with significant engine events, and it is possible to guide classification tools through curated feature engineering including time-frequency analysis, or wavelet analysis [34].

Sensing engines can be done on resource-constrained devices and still enable continuous monitoring, with hardware-agnostic algorithm implementations [69]. Another example used an Android mobile device to record vehicle audio, create frequency and spectral features, and detect engine faults by comparing recorded clips with reference audio files, where the authors could detect engine start, drive belt issues, and excess valve clearance [80].

Engine misfiring is a typical within older vehicles due to component wear. Misfires have been detected in a contact-less acoustic method with 94% accuracy, relative to 82% accuracy attained from vibration signals. Without opening the hood and recording at the exhaust, the authors reached 85% classification accuracy from audio (which again outperformed vibration) [113]. While some algorithms have been developed without physical process knowledge, others make use of system models to improve diagnostic performance. Use of aspects of the physical model help reduce algorithm complexity, requiring a feature engineering work before analysing the input data.

Siegel used feature extraction to reach 99% fault classification accuracy in another study of misfires, well exceeding the prior art. This work demonstrates that feature selection and reduction techniques based on Fisher and Relief Score are effective at improving both algorithm efficiency and accuracy. Data were collected from a smartphone microphone [111]. Similar acoustic data and engineered features have been successfully used to monitor the condition of engine air filters, helping to precisely time change events [106].

Some feature engineering techniques, such as wavelet packet decomposition used in Siegel's misfire and air filter work, have found application in other engine diagnostic contexts such as identifying excessive engine valve clearance [43] and combustion events [91]. Other common faults relating to failed engine head gaskets, valve clearance issues, main gearbox, joints, faulty injectors and ignition components can also be detected thanks to vibrational analysis [60]. Transmissions, too, may be monitored, and a damaged tooth in a gear can be diagnosed capturing sound and vibration at a distance [29]. Even high-speed rotating assemblies, such as turbochargers, can be monitored – turbocharging is increasingly common to meet stringent economy and emissions standards, and engine compression surge has been identified and characterized by sound and vibration [73].

Non-automotive engines and fuel type can also be identified using Vibroacoustic approaches. Smartphone sensors were used to classify normal and atypical tappet adjustments of tractor engines with 98.3% accuracy [96], and fuel type can be determined based on vibrational mode – with 95% accuracy [8].

Other studies have used physics to guide feature creation for indirect diagnostics, e.g. measuring one parameter to infer another. In [103] for example, the authors originally used engine temperature over time as a surrogate measure for oil viscosity and found promising results relating $\frac{dT}{dt}$ to viscosity. As it turns out, vibration may be used as further abstraction. By measuring engine vibration one may determine the engine speed (RPM) and it is possible to determine whether the car is in gear [107] to identify when the car is at rest. As an extension of our previous work, we now note that using knowledge of the car's warm-up procedure (which typically involves a so-called "fast idle" until the engine warms up to temperature, to reduce emissions), is therefore possible to time how long it takes to go from fast idle (where the engine runs quickly to warm up and therefore reduce emissions) to slow idle and infer temperature from vibration, thereby creating a means of inferring oil viscosity from vibration alone and without the use of onboard temperature data.

For the scope of minimize the knowledge gap between vehicle operators and expert mechanics, a mobile application called OtoMechanic has been designed. It uses sound to improve diagnostic precision relative to that of untrained users. Intelligence is embedded in a mobile application wherein a user uploads a recording of a car and answers related questions to produce a diagnostic result. The application works by reporting the label of the most-similar sample in a database as determined by a convolutional neural network (VGGish model). Peak diagnostic accuracy is 58.7% when identifying the correct class from twelve possibilities [79].

Algorithms have the most value when they are transferrable, as they can be trained on one system and applied to another with high performance. In one study, transferrability across similar engine geometries of different cars was considered in the context of detecting piston and cylinder wear, and measuring valve-train and roller bearing state [12].

Powertrain diagnostics are important, but it is equally important to instrument other vehicle subsystems. We look next to how offboard diagnostics have been applied to vehicle suspensions as a means of improving performance, safety, and comfort.

Wheel, Tire and Suspension

As with powertrain diagnostics, suspensions may be monitored using vibroacoustic analysis, optical and other methods, or a combination of both.

In terms of vibroacoustic analysis, wireless microphones have been used to monitor wheel bearings and identify defects based on frequency-domain features [88],

and vibration analysis has been implemented o detect remaining useful life of mechanical components such as bearings [124]. Similar data sources and algorithms have been exploited to identify the emergence of cracks in suspension beams [13].

Other vibroacustic approaches have been implemented, using accelerometers and GPS to measure tire pressure, tread depth, and wheel imbalance [112, 107], primarily using frequency-based features. Such solutions could be extended to instrumenting brakes, using frequency features and low-pass acceleration to measure specific pulsations occurring only under braking, or gyroscopes, to measure events taking place only when turning (or driving in a straight line).

As noted earlier, researchers have demonstrated a means of diagnosing six vehicle component faults using vibration and Deep Learning Diagnostics algorithms running within constrained compute environments. Some of these diagnostics target wheels and suspensions, specifically at wheel imbalance, misalignment, brake judder, damping loss, wheel bearing failure, and constant-velocity joint failure. Each fault was selected as manifesting with characteristic vibrations and occurring at different frequencies. This research required vehicle to be driven at particular speeds in order to maximize signal.

Bodies / Noise, Vibration, and Harshness

Recent studies have utilized MEMS accelerometers (micro electro mechanical systems devices) to investigate vehicle vibration indicative of vehicle body state and condition. Specifically, MEMS accelerometers allow the diagnosis of articulation events in articulated vehicles, e.g. buses. In one study, sensors were placed within the vehicle, with one located within each of the two vehicle segments in order to detect articulation events and monitor changes in bearing play resulting from wear and indicating a need for maintenance [123].

Vehicle occupants value fit and finish and a pleasant user experience while riding in a vehicle. To this end, there is an unmet need for realtime noise, vibration, and harshness (NVH) diagnostics. VA and other offboard techniques may find application in identifying and remediating the source of squeaks, rattles, and other in-cabin sounds in vehicles after delivery from the factory.

1.5.2 Vehicle Operating State

Beyond monitoring vehicle condition and maintenance needs, offboard diagnostics have the potential to identify vehicle operating state in realtime, e.g. to identify whether a vehicle is moving or not, the position of the throttle, steering, or braking controls, or in which gear the selector is currently placed. To this end, mobile devices can be used to enable sensitive classification algorithms making use of accelerometers and cameras.

At their simplest, mobile devices may be used to detect whether someone is

in a car and driving [99]. Some context-aware applications use sensor data to detect whether a vehicle is moving, and if so, to undertake appropriate actions and adaptations to enhance occupant safety, e.g. by disabling texting while in motion [5]. The aforementioned study made use of accelerometers to supervise and eliminate false positive events from the training dataset, ultimately yielding a performance with 98% specificity and 97% sensitivity [5].

Others have used similar data to detect the operating state of a vehicle in order to identify lane changes or transit start- and end-points, using smartphones [119].

Vehicle operating state is an ongoing area of research, with new developments exploring:

1. Accelerometer-based accident detection and response [37], including one research project wherein smartphones were used to detect and respond to incidents taking place on all-terrain vehicles and capable of differentiating "normal" driving from simulated accidents with over 99% confidence [74]. Some approaches use these data to automate rerouting [37]

2. Using K-means clustering with acceleration data to identify driving modes, such as idling, acceleration, cruising, and turning as well as estimating fuel consumption [68] (there are multiple methods for using mobile sensors as surrogate data to indirectly estimate fuel consumption) [58].

This area of research is fast-evolving, particularly as context-sensitive applications gain prominence. Another fast-emerging application of pervasive sensing and offboard diagnostics is to occupant state and behavior monitoring.

1.5.3 Occupant Monitoring

Many automotive incidents resulting in injury or harm to property result from human activity. It is therefore essential to monitor not only the state and condition of a vehicle, but also to supervise the driver's state of health and attention in order to reduce unnecessary exposure to hazards and to promote safe and alert driving [2].

Occupant monitor (including drivers and passengers) may be grouped broadly into three categories:

1. Occupant *State*, namely health and the capacity to pay attention to and engage with the act of driving

2. Occupant *Behavior*, namely the manner of driving, including risks taken and other parameters informing telemetry, e.g. for informing actuarial models for insurers or for usage-based applications [108]

3. Occupant *Activities*, namely the actions taken by occupants within the vehicle (e.g. texting), with particular application to preventing or mitigating the effects of hazardous actions

Occupant State

Vehicle occupant state may be monitored for a variety of reasons, e.g. related to drowsiness, drunkenness, or drugged behavior. Mobile phones have been used to detect and report drunk driving behavior, with accelerometers and orientation sensors informing driving style assessments indicative of drunkenness [37, 31]

The main issue with occupant state is related to drunk driving state. With mobile phones placed in the vehicle there is the opportunity to detect that particular condition observing both the driving style [37] (using accelerometers and orientation sensors) [31] and the driver alertness monitoring the eye state with mobile device camera [32]. As with vehicle diagnostics, multiple sensor types may be used to monitor driver state [57].

Counter-intuitively, as highly automated driving grows in adoption, there will be growing demand for occupant metrics - at first, to ensure that drivers are "safe to drive," and later, to make judgments as to how much to trust a driver's observations and control inputs relative to algorithms, e.g. to trust a lane keeping algorithm more than a drunk driver, but less than a sober driver.

Occupant Behaviors and Telemetry

Smartphones have been widely deployed in order to develop telematics applications for vehicles and their occupants as a form of "off board supervision" [132, 133]. These data have been used by insurance companies to monitor driver behaviors and to develop bespoke policies reflecting real-world use cases, risk profiles, and driver attitudes.

Pervasively-sensed data are used in three main insurance contexts, helping to:

1. Monitor a driver and/or vehicle's distance traveled, supporting usage-based insurance premiums [130].

2. Supervise eco-driving [37], using metrics such as vehicle use or driver behavior (including harshness of acceleration and cornering, with demonstrated performance achieving more than 70% accurate prediction [82]) to guide more-conservative behavior. Related to this, vehicle speed can be monitored with smartphone accelerometers alone, with an accuracy within 10MPH of the ground truth [129].

3. Observe driver strategy and maneuvering characteristics, to assess actuarial risk [37] and feed models with real-world data [130] to inform premium pricing. This information may be used as input into learned statistical models representing drivers, vehicles, and mobile devices to detect risky driving maneuvers [17]. Notably, driving style and aggression level can be detected with inexpensive multi-purpose mobile phones [85, 53] and vehicles or drivers may

be tracked to identify the potential for high risk operation [51], in cases with no additional sensors installed in the vehicle [39].

Other behavior monitoring and telemetry use cases relate to safety, providing intelligent driver assistance by estimating road trajectory [85], using smartphones to measure turning or steering behavior (with 97.37% accuracy [81]), classifying road curvature and differentiating turn direction and type [141], or offering even-finer measure of steering angle to detect careless driving or to enhance fine-grained lane control [21]. In [142], the authors were able to identify straight driving, stationary, turning, braking, and acceleration behaviors independently on the orientation of the device. These approaches may use several learning approaches, though many use end-to-end deep learning framework to extract features of driving behavior from smartphone sensor data.

Occupant Activity

Human activity recognition has been widely studied outside vehicular contexts, and the performance of such studies suggest a likely transferrability to vehicular environments, with pervasive (ambient) or human monitoring gaining prominence. We consider in-vehicle and non-vehicular activity recognition in this survey, as the techniques demonstrated may inspire readers to reapply prior implementations or to adapt their methods to automotive contexts.

In this study, we consider three categories of "off-board" sensing for human activity recognition.

1. **In vehicle activity recognition:** Similarly to the use of pervasive sensing for drunk driver detection, mobile sensing has been applied to the recognition of non-driving behaviors within vehicles, for example distracted driving and texting-while-driving. Detecting texting-while-driving is based upon the observation of turning behavior, as measured by a single mobile device [11]. Mobile sensing solutions making use of optical sensors have also been demonstrated to detect driving context and identify potentially-dangerous states [65]. A survey of smartphone-based sensing in vehicles has been developed, describing activity recognition within vehicles including driver monitoring and the identification of potentially-hazardous situations [37].

2. **Workshop activity recognition:** Human-worn microphones and accelerometers have been used to monitor maintenance and assembly tasks within a workshop, reaching 84.4% accuracy for eight-state task classification with no false positives [70]. In another study, similar sensors were used to differentiate class categories included sawing, hammering, filing, drilling, grinding, sanding, opening a drawer, tightening a vice, and turning a screw driver using acceleration and audio data. For user-independent training, the study attained recall and precision of 66% and 63% respectively [134]. The methods

demonstrated in identifying different work- and tool-use contexts may provide the basis for identify human engagement with various vehicle subcomponents, e.g. interaction with steering wheels, pedals, or buttons, helping create richer "diagnostics" for vehicle occupants and their use cases.

3. **General activity recognition:** Beyond identifying direct human-equipment interactions, mobile sensing has been applied to the creation of context-predictive and activity-aware systems [25]. Wearable sensors and mobile devices with similar capabilities have been used to detect user activities including eating, drinking, and speaking, with a four-state model attaining in-the-wild accuracy of 71.5% [139]. In another study, user tasks were identified over a 10-second window with 90% activity recognition rate [62]. In vehicles and mobile devices, computation is often constrained. Researchers have demonstrated activity classification using microphone, accelerometer, and pressure sensor from mobile devices in a low-resource framework. This algorithm was able to recognize 15-state human activity with 92.4% performance in subject-independent online testing [59].

   Related to tailoring user experience, acoustic human activity recognition is an evolving field aimed at improving automotive Human Machine Interfaces (HMI) suitable across contexts. In one study, 22 activities were investigated and a classifier was developed reaching an 85% recognition rate [117]. Acoustic activity recognition may also be applied directly to general activity detection.

   In consumer electronics, activity or context recognition may be used to detect appliance use or to launch applications based on context, or used as sound labeling system thanks to ubiquitous microphones. Sound labeling and activity/context recognition helps augment classification approached by defining a context (environment) in order to limit the set of classes to be recognized before classifying an activity based on available mined datasets. In one sample application, 93.9% accuracy was reached on prerecorded clips with 89.6% performance for in-the-wild testing. The demonstrated system was able to attain similar-to-human levels of performance, when compared against human performance using crowd-sourcing service Amazon Mechanical Turk [64] In [36] human feedback is used to provide anchor training labels for ground truth, supporting continuous and adaptive learning of sounds.

Detecting activities within a vehicle - using acoustic sensing or other approaches - may help to tailor the vehicle user experience based on real-time use cases. Studying existing techniques for general activity recognition and applying this to an automotive context has the potential to improve the occupant experience as well as vehicle performance and reliability.

Of course, monitoring vehicles and their occupants alone does not yield a comprehensive picture of a vehicle's use case or context: the last remaining element to be monitored is the environment.

1.5.4 Environment

Environment monitoring is a form of off-board diagnostic that may help to disaggregate "external" challenges from problems stemming from the vehicle or its use, e.g. in separating vibration stemming from cracks in the road from vibration caused by warped brake rotors. Environment monitoring is also a crucial step towards autonomous driving, helping algorithms understand their constraints and operate safely within design parameters.

Already, smartphones can be used as pervasive sensors capable of complementing contemporary ADAS implementations. In one study, vehicle parameters recorded from a mobile device accelerometer have been used to measure road anomalies and lane changes [48]. Vibroacoustic and other pervasively-sensed measurements have also been used for environment analysis. These may be used to calibrate ADAS systems by monitoring road condition, to classify lane markers or curves, to measure driver comfort levels, and as traffic-monitoring solutions. Some example pervasively-sensed environment monitoring approaches are described as follows:

- Pavement **road quality** can be assessed by humans, though mobile-only solutions [115] may be lower-cost, faster, or offer broader coverage. Accelerometers may be used for detecting defects in the road such as potholes [37, 100, 40, 28] or even road surface type (e.g. gravel detection, to adapt antilock braking sensitivity) [6]. Road-surface materials and defects may also be detected from smartphone-captured images using learned texture-based descriptors [27]. It is also relevant to consider the weather when monitoring the road surface condition for safety, and microphone-based systems have demonstrated performance in detecting wet roadways [1]. Captured at scale, smartphone data may be used to generate maps estimating road profiles, weather conditions, unevenness, and mapping condition more precisely and less expensively than traditional techniques [143, 98], with enhanced information perhaps improving safety [122]. These data may be used to report road and traffic conditions to connected vehicles [95].

- **Curve** data and road classification may integrate with GPS data to increase the precision of navigation system. Mobile phone IMU's have been used to differentiate left from right and U-turns [141], and it is reasonable to believe that combining camera images with IMU data (and LiDAR point clouds, if available), may help to generate higher-fidelity navigable maps for automated vehicles.

- The **comfort level** of bus passengers has been investigated with mobile phone sensors, attaining 90% classification accuracy for defined levels of occupant comfort [23].

- Mobile sensing has been used to detect parking structure occupancy [22].

- Acoustic analysis of **traffic** scenes with smartphone audio data has been used to classify the "busyness" of a street, with 100% efficacy for a two-state model and 77.6% accuracy for a three-state model. Such a solution may eliminate the need for dedicated infrastructure to monitor traffic, instead relying on user device measurements [131]. In [93], the authors implemented a 10-class model, classifying environments based on audio signatures indicating energy modulation patterns across time and frequency and attaining a mean accuracy of 79% after data augmentation. Audio may also be used to estimate vehicular speed changes [61]

- Offboard sensors lead many lives - as phones, game playing devices, and diagnostic tools - so it is important for devices to be able to identify their own **mobility use context**. One approach uses mobile device sensors and hidden markov models to detect transit mode, choosing among bicycling, driving, walking, e-bikes, and taking the bus, attaining 93% accuracy [138], which may be used to create transit maps and/or to study individuals' behaviors [3]

Though the described approaches relate primarily to cars, trucks, and busses, many solutions apply to other vehicles as well. Off-board diagnostics for additional vehicle classes are described below.

1.5.5 Non-automobile Vehicles

Off-board and vibroacoustic diagnostics capabilities may be used for non-automotive, truck, or bus-type vehicles, including planes, trains, ships, and more:

- As with cars, **train** suspensions and bodies have been instrumented using vibroacoustic sensing. Train suspensions have been instrumented and monitored using vibrational analysis [24]. Brake surface condition has also been monitored with vibroacoustic diagnostics [97]. Train bodies (NVH) have also been monitored, notably the doors on high-speed trains. Their condition may be inferred with the use of acoustic data [120].

- **Aerial vehicle** propellers are subjected to high rotational speeds. If imbalanced or otherwise damaged, measurement of the resulting vibrations may lead to rapid fault detection and response [52].

- In **maritime environments** vibroacoustic diagnostics has been implemented with the use of virtualized environments and virtual reality to allow remote human experts with access to spatial audio and body-worn transducers to diagnose failures remotely [10].

The applications for off-board, pervasive sensing and vibroacoustic diagnostics for system, environment, and context monitoring will continue to grow across vehicle classes.

1.6 A Need for Context-Specific Models

Often, classification relies upon generalizable models to ensure the broadest applicability of an algorithm, perhaps at the expensive of performance. Occasionally, classifiers - such as activity recognition algorithms - may make use of "personalized" models. Personal Models are trained with a few minutes of individual (instance-specific) data, resulting in improved performance [135]. This approach may be extended from activity recognition to off-board vehicle diagnostics, with the creation of instance- or class-specific diagnostics algorithms. Selecting such algorithms will therefore first require the identification of the monitored instance or class, which is an ongoing research challenge.

We propose the creation of a "context-based model selection system," aimed at identifying the instrumented system precisely such that tailored models may be used for diagnostics and condition monitoring.

Differentiating among vehicle makes, models, and use contexts will allow tailored classification algorithms to be used, with enhanced predictive accuracy, noise immunity, and other factors - thereby improving diagnostic accuracy and precision, and enabling the broader use of pervasive sensing solutions in lieu of dedicated onboard systems.

There are grounds to believe that implementing such a system is feasible. Automotive enthusiasts can detect engine types and often specific vehicle makes and models from exhaust notes alone - and researchers have demonstrated success using computer algorithms to do the same, recording audio with digital voice recorders, extracting features, and testing different classifiers - finding that it is possible to use audio to differentiate vehicles [9].

The more the application knows or infers about the instrumented system, the more accurate the diagnostic model implemented may become.

1.6.1 A Representative Implementation

Though there are a multitude of ways in which to implement such a system, the authors have given consideration to several architectures and identified one promising path forward. The following subsections describe a representative implementation

upon which a contextual identification system and model selection tool may be built in order to improve diagnostic accuracy and precision for vibroacoustic and other approaches.

The concept begins with the notion of Contextual Activation, i.e. the ability for a mobile device to launch a diagnosticsapplication in background when needed, just as it might instead load a fitness app when detecting motion indicative of running.

With the application launched, sensor samples may be recorded, e.g. from the microphone and accelerometer. These data may then be used to identify the vehicle and engine category, perhaps classifying these based entirely on the noise produced, or in concept with additional data sources, such as a connected vehicle's Bluetooth address, its user/company's vehicle management database and so on.

Once the vehicle and variant is identified, this information may be used to identify operating mode, and from this, a "personalized" algorithm may be selected for diagnostic or other activities.

In aggregate, the system might be imagined along the lines of a decision tree – by selecting the appropriate leaf corresponding to the vehicle make, variant, and operating status, it becomes possible to select a similarly-specific prognostic or diagnostic algorithm tailored to the particular nuance of that system. Implemented carefully, the entire system may run seamlessly, such that the sample is captured, the context is identified, and the user is informed of issues worth her or his time, attention, and money to address.

1.6.2 Contextual Activation

This seamlessness is key to the success of the proposed pervasive sensign concept – to maximize the utility of a diagnostic application, it must require minimal user interaction. The use of contextual activation enables the application to operate data capture only when the mobile device is in or near a vehicle, and the vehicle is in the appropriate operating mode for the respective test (e.g. on, engine idling, in gear, or cruising at highway speeds on a straight road). This allows the software (built as a dedicated application inside the mobile device) to operate as a background task or to be launched automatically when the mobile device detects it is being used within an operating vehicle.

Other potential implementations of this automatic, context-based software execution include automatic application launching when the phone is connected via Bluetooth to the car, or when a mapping or navigation application is opened. In this specific situation, the GPS and accelerometer may be utilized to understand the specific kind of road the vehicle is running on, as well as its speed, e.g. to disallow certain algorithms such as those used to detect wheel imbalance from running on cracked or gravel roads.

One possible embodiment of the system may comprise a "context layer" for generating characteristic features and/or uniquely-identifiable "fingerprints" for a

particular system, which then passes system-level metadata (system type, other details, and confidence in each assessment), along with raw data and/or fingerprints to a classification and/or gradation system. This "context layer" may be used both in system training and testing, such that recorded samples may exist alongside related metadata and therefore allow for classification and gradation algorithms to improve over time, as increasing data volume generates richer training information even for hyper-specific and rare system configurations.

The application may therefore capture raw signals and preprocess engineered features to be sent to a server (these fingerprints are space-efficient, easier to anonymize, more difficult to reverse, and repeatable), uploading these data at regular intervals.

1.6.3 Vehicle (and Instance) Identification

The next step after identifying that the mobile device is in or near a vehicle will be vehicle identification, or identification of a grouping of similar vehicle variants. Depending on the system to be diagnosed, similarities may take place as a result of engine configuration, suspension geometry, and so on.

A vehicle "group" may be identified by engine type - that is, configuration, displacement, and other geometric and design factors. For example, we may classify an engine to be gasoline powered, with an inline configuration, having 4 cylinders with 2.0 liters of displacement, turbocharged aspiration, and manufactured by Ford.

If in our database we do not have any available diagnostic algorithm (e.g. a misfiring test [111]) for this engine type, we then look at increasingly less-specific parent class models, such as generic car-maker-independent gasoline I4 2.0 turbo engine. If this is also not available, we go higher- and higher-level until it is necessary to use the least-specific model, in this case, a model trained for all gasoline engines - at the cost of potentially-decreased model performance. Alternatively, we may consider to use a *similar* engine, with slight difference in displacement or powered by LPG fuel. A representative model selection process, indicating a means of identifying a vehicle variant and then selecting the most-specific diagnostic model available in order to improve predictive accuracy is shown in Figure 1.6.3.

Extending this process, it may become possible to identify a particular vehicle instance, particularly based on features learned over time (e.g. indicating wear).

Other subsystems, such as bodies and suspensions, are harder to identify - but may still be feasible. For example, identifying operating context and road condition may be used to identify when a car hits a pothole, with the post-impact oscillations indicating the spring rate, mass, and damping characteristics indicative of a particular vehicle make or model. As with engines, subtleties may be used to identify vehicle instances, e.g. damping due to tire inflation.

If the vehicle is known to the mobile device user and "short list" of vehicles

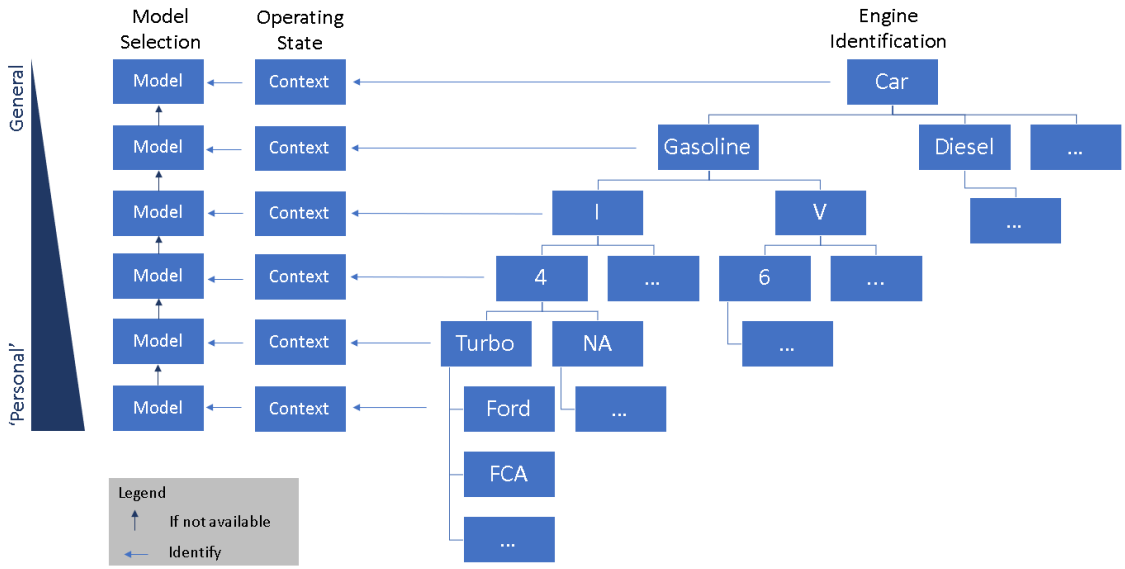


Figure 1.2. Model selection process based on context identification. Once the engine is identified in a tree, and operating state is determined, a more or less "personalized" trained model is selected

frequented by the user, this portion of the classification may be replaced by ground-truth information, or selection may be made among a smaller/constrained subset of plausible options. Moreover, if we activate the application based on the Bluetooth connection indicating proximity to a particular vehicle, we may identify it with near-certainty. In order to reduce the degree of user interaction required, we may use this and other automation tools to identify vehicles and operating context in order to run engine and other diagnostics as a sort of background process.

1.6.4 Context Identification

Once the vehicle is selected, its context must be identified. Context classification uses vibroacoustic cues (and vehicle data, if available) to identify the operating state of the engine, gearbox, and body. For example, is the engine on or off? If it is on, what is the engine RPM? Is the gearbox in park, neutral, or drive – or if a manual transmission, in what gear is the transmission, and what is the clutch state?

Some algorithms will be able to operate with minimal information related to vehicle context (e.g. diagnosing poor suspension damping may require the vehicle simply to be moving as determined by GPS, whereas measuring tire pressure may require knowing the car is in gear [107] and headed straight [112] to minimize the impact of noise and other artifacts on classification performance.

With context selection, we follow a similar process to that used for vehicle type and instance identification, selecting the model with metadata best reflecting the instrumented system to ensure the best fit and performance.

In an example implementation, we might create a decision tree to identify the current vehicle state - with consideration given to engine operating status, gear engagement, motion state, and other parameters - and rather than using this tree to select a model for diagnostics, we may prune this tree to suit a particular diagnostic application's needs (e.g. engine power might not matter for an interior NVH detection algorithm, or a tire pressure measurement algorithm may require the vehicle to be moving to function [112]. The pruned tree may then be used to select the ideal algorithm with the most-specific match between the training data and the current operating context.

With complicated vehicle operating contexts, and with systems measured under uncertainty, binary states may not be sufficient to describe the system status. For this reason, we instead propose the use of a three-state system comprising values of -1, 0, and 1.

If a context parameter is 1, it is true or the condition is met. If it is 0, it is false, or the condition is not met. If an identified context parameter is a negative value (-1) that means it is unnecessary for the diagnostic application, not available, uncertain, or not applicable (e.g. lateral acceleration is not applicable if a vehicle is stationary).

These negative values are removed from the input feature vector, and the corresponding element class is also removed from the reference database. In this way, a nearest neighbor matching algorithm will ignore uncertain or unnecessary data in considering the model to be used for diagnostics or prognostics. This matching algorithm needs a distance metric, which are algorithm-specific weighting coefficients used to define the importance of each context parameter (e.g. state of the engine may be more important than the amount of longitudinal acceleration when diagnosing motor mount condition, assuming both parameters are known).

The model selection process relies on correct identification of both the vehicle variant and the context. Here, we see one proposed method for identifying the vehicle context and using those relevant features to select an appropriate "nearest neighbor" when identifying the optimal diagnostic or prognostic model to choose. Context parameters are identified through distinct, binary classifiers capable of reporting confidence metrics. The context vector comprises entries with three possible states (yes/no/uncertain or irrelevant), and those uncertain or irrelevant entries and their corresponding matches in the reference database are removed such that only confident, relevant parameters are used to select the nearest trained model. A visual overview of the context identification and nearest-neighbor model selection process appears in Figure 1.6.4. Just as Bluetooth connectivity may be used to limit the plausible set of vehicle types, so too may data from sources such as on-board diagnostic systems be used to limit the set of feasible operating contexts,

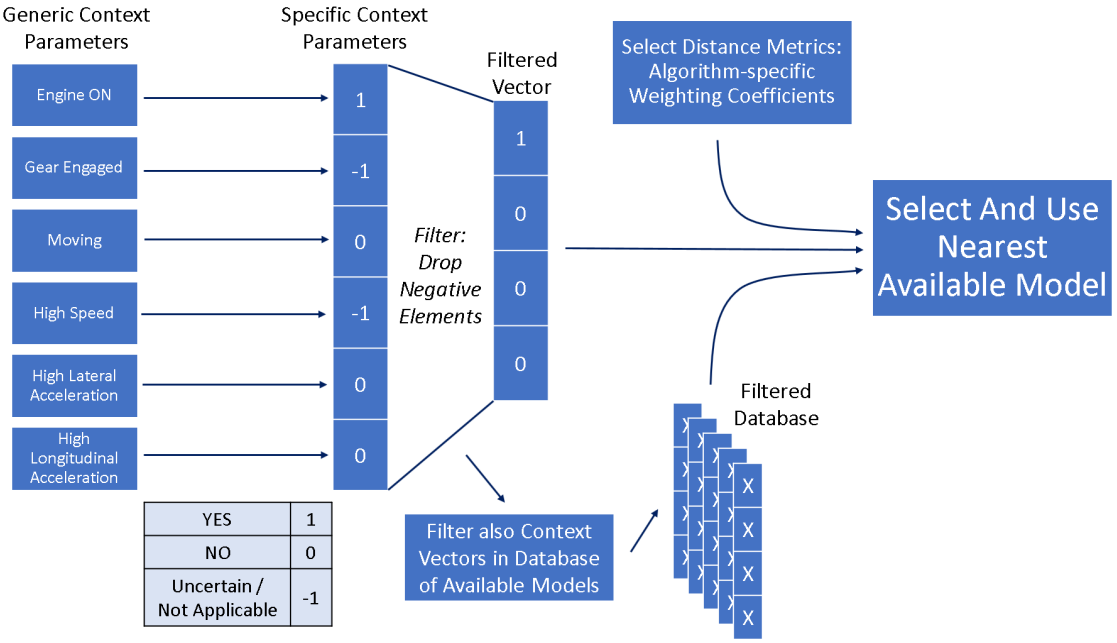


Figure 1.3. Nearest neighbor model selection: after the context is identified and modeled as a vector, this vector is filtered and compared to available context models in the database. The nearest model is used

thereby removing uncertainty from the model selection process.

Combining vehicle identification with context classification, comprehensive vehicle "metadata" may be identified – for example, "light duty, 2.0 liter, turbocharged, Ford, Mustang, Joe's Mustang." With the fullest possible context identified, a list of feasible diagnostic algorithms may then be shortlisted.

1.6.5 Diagnostics

Certain diagnostics will be feasible for each set of vehicle classes and operating contexts. If a vehicle is moving, only algorithms working for moving vehicles will be available. If a vehicle is at idle, only algorithms operating at engine idle will be available. If a vehicle is on a gravel road, only algorithms suitable for rough terrain will be offered.

When the mobile device identifies an appropriate context and short-lists feasible diagnostic algorithms, the most-specific diagnostic model of that type available with sufficient n of training vehicles will be chosen and run on the raw data or engineered features provided by the mobile device (and vehicle sensors, if available). These algorithms will initially start out coarse - is the engine normal or abnormal? Are the brakes normal or abnormal?

Over time, as algorithms become more sensitive, and as training data are generated (with labeled or semi-supervised approaches), more classes may be added. The intent for this system is to transition from binary classification (good/bad), to gradation (80% remaining life, 10% worn), to diagnostics so sensitive that they in fact are *prognostics* – that is, algorithms sensitive enough that faults may be detected and addressed proactively.

The result will be improved efficiency, reliability, performance, and safety, and eased management of large-scale, high-utilization fleets, such as those that will be run by shared mobility services. The algorithms used may over time be adapted to minimize a cost function, e.g. balancing user experience with maintenance cost with the likelihood of having a car break down on the road. This will supplant data-blind proactive scheduled maintenance with data-driven insights sensitive to use environment, risk tolerance and mission-criticality.

1.7 Our Goal: Engine Classification

In view of the vibroacoustic diagnostic vision outlined in this chapter, my thesis work tackles the specific goal of identifying vehicle context as a first, enabling step towards this larger vision. The first and perhaps most informative information about the context is the engine characteristics, because it is one of the most relevant source of noise of a vehicle, as well as one of the main components influencing the behavior and failures of vehicles. My goal is to classify the engine type and configuration based on the sound it emits. I seek to predict some characteristics of internal combustion engines such as:

1. Turbocharging
2. Fuel type
3. Number of cylinders
4. Shape (Inline or Vee)
5. Size (displacement)
6. Power

This is a sequential problem, because one engine characteristic may (and does) impact the others, as I will explain in Section 2.2.4. For that reason, instead of trying to solve a parallel multi-label problem, the attention is focused on one label at time, in a standard engineering of *Divide et Impera* approach.

There are several ways to approach this problem, and different aspects of the engine to be captured. I decided use Artificial Intelligence because of its potential to better generalize (section 1.4) versus traditional machine learning.

1.7.1 The Choice of Acoustic Signals

The reasons why I made the decision of recording sound instead of vibration (or temperature profile) are outlined in this Section. Vibrational analysis is widely used in diagnostics of industrial machinery and equipment, because it is less affected by external conditions, such as background noise, wind or damages, and the sensor is generally embedded in the machine. In this way it provides more robust results, with less need to clean the data. On the other hand, there are several reasons why acoustic analysis may be more powerful in this application:

1. Audio may be **more suitable** than vibration for as the air-transmission path eliminates some system-coupling, making it easier to disaggregate signals [34].

2. **Ease of collecting data:** it is more user-friendly to record an audio signal with a standard smartphone application than capturing the vibration of the smartphone itself or even worse of a sensor embedded into the engine. For this project's purposes the data collection must be rather *crowd-sourced* to ensure enough variability, so the ability to easily deploy software to users is important.

3. If a human can classify some engine characteristics from its sound, then AI should achieve similar or better performance. It is not an easy task for humans, but trained people can easily distinguish a Diesel from a Gasoline engine, or a V8 from a I4, so it is reasonable to assume AI should be able to do the same with access to the same information.

1.7.2 Side Goal: an effective Framework

In our plan, we want not only to identify engine characteristics, but aim also to do so in a manner that is not computationally-intensive such that it might run efficiently on a mobile device. Furthermore, a secondary but still important goal lies in the way the software is built. We aim for it to be:

- **Scalable**, in order to allow future extensions of its capabilities to similar use-cases (e.g. assessing wear, detecting damages) in different contexts (e.g. appliances) without difficult modifications. The output of this Master's Thesis is a framework, not just a piece of a software. In fact, in our vision for the future, everyone may use this software to diagnose each kind of noise, by providing his own functions, performance metrics, and new futuristic custom designed classifiers.

- **Modular**, so that it may be used by other researchers from both Michigan State University and Politecnico di Torino to pursue their own goals. This is achieved by organizing it with modular functions and rich code documentation.

- **User-Friendly**, for the same reason as before, and to allow non-technical people to access the results. Our framework is designed to be controlled easily by modifying cells within an Excel file. This file is divided in different sheets to control different parts of the process:
  - The core is represented by sheet **Load**, where all general control parameters are set by the user. A simple documentation is provided for a better understanding of the meaning of each parameter. This sheet is divided in multiple parts for the different steps (see Figure 1.4).
  - Sheet **Labels to Consider** is also used for loading purposes, and filters the data based on labels and classes we need to use (see Section 2.2.4)
  - Sheet **Features** manages the Feature Extraction process (see Section 2.3)
  - Sheet **Dimensionality Reduction** manages the Feature Selection process (see Section 3.2)
  - Sheets **Classification and Regression Search** manage the Machine Learning parameters (see Section 3.3)

  Furthermore, the folder structure is managed by the software itself, creating a new folder for each simulation and for each feature set extracted from the audio samples, automatically saving results inside that folder.

	A	B	C	D
1	Load	audiopath	/home/remotediag/Audio	Path where audio files are located
2		mainpath	/home/remotediag/OwnClouc	Path where this file is located (and .py ones)
3		storagepath	/home/remotediag/Storage	Path where dataset is stored (outide of OwnCloud to avoid big sincronization, for example)
4		inWindows	0	If the machine we are working on has Windows OS
5	Features	chunk_duration	2.10	Desired duration (in seconds) of the chunks. Shorter --> more samples but less informative
6		developing_mode	0	If we want to use a smaller version of the dataset (for developing purposes only)
7		chunks_per_sample	1	Valid only for developing mode activated, maximum number of chunks per sample to load
8		train_test_stratify	fuel	Stratification label of the dataset for Train-Test split
9		sample_rate	48000	Sample Rate in Hz
10		test_size	30%	Size of Test Dataset
11		feature_scaler	[MinMaxScaler()]	List of Scalers of Matrix X. More of them can be used sequentially
12		features_to_avoid	['DWT', 'FFT']	Which features we are extracting only for extracting subsequent features, but not needed in X matrix
13	EDA	EDA	0	If we want to explore the Dataset after features are extracted (it may take some time)
14		relplot_charts	0	If we want to plot charts with confidence intervals
15		save_chart	1	If we want to save information about the dataset and about mean features trend grouped by class
16		pairplot	1	More insightful statistics on the features
17		features_importances	0	Chart of features importance with XGBoost, targeting label used for stratification (row 8)
18	Fitting	classif_metric	acro', 'f1_macro', 'f1_weighte	Metrics chosen from https://scikit-learn.org/stable/modules/model_evaluation.html#scoring-parameter
19		regr_metric		Metrics chosen from https://scikit-learn.org/stable/modules/model_evaluation.html#scoring-parameter
20		FTid	[31,29,28]	List of Features ID to use (for either Fitting or Results evaluation purposes)
21		all_features	0	If we want to use all Features ID available (it will take some time)
22		cv	3	Number of folds for cross validation
23		label_to_predict	['fuel', 'cyl']	List of labels we want to predict (separately)
24		classes	-1.00	List of classes to consider during training (e.g. [3,4,6,8]). Not working if label_to_predict is a list (of course)
25		variance_threshold	0.00	Threshold for VarianceThreshold applied upstream dimensionality reduction
26		classif1_regr0	1	Set 1 for classification and 0 for regresssion
27		number_of_iterations	50	Number of Iteration of the Randomized Search CV object. The higher, the more computing time and risk of overfitting
28		features_to_drop	[]	List of extracted features to be dropped from X matrix before fitting (e.g. ['FFT Binned', 'PSD'])
29		labels_as_features	[]	List of labels used as features. Useful for sequential process, once one of them is identified (e.g. 'fuel' when searching 'cyl')

Load | Labels To Consider | Features | Dimensionality Reduction | Classification Search | Regression Search | ...

Figure 1.4. A look into Parameters.xlsx: Control file for setting desired parameters. It is divided by categories indicated on the first column

Those characteristics lie under the concept of *Framework*, i.e. a structured and versatile system that may be used in different situations and expanded to more

complex circumstances. This goal has been one of the most challenging part of this Thesis, both in terms of knowledge and of workload, because every different aspect and possible exception are crucial and must be taken into account. I decided to build this software in the most common object-oriented programming language at the time of writing: Python 3.6. It is open source and widely used in very different big tech companies (e.g. Google) [87]. After more than 3000 lines of code and 6 months of hard work, I am proud of introduce my new framework adaptable to similar situations for audio classification.

Chapter 2

Machine Learning Workflow: From Sound to Features

In this chapter, we explain the procedure adopted for this project, starting from the goal that drove our work. We then outline the process, starting from the collection of the data until the generation and visualization of informative characteristics of that sound. We further explain some general concepts of the machine learning procedure, and how we applied them in out specific case, justifying the choices made and their impact on the final result.

2.1 Some common ground on Artificial Intelligence

Before describing the workflow that brought us to our successful results, I want to explore the theory behind the process, so that even a reader not familiar to artificial intelligence and machine learning could have a taste of it. This is not intended to be a complete dissertation on machine learning, but just a simple yet stimulating overview on this complex world. After this introduction, other concepts are clarified by the time they arise in this document in order to simplify the reading and making it easier to follow and understand.

2.1.1 What is Artificial Intelligence (AI)

First of all, what is artificial intelligence? Textbooks define AI as the field of study of intelligence agents, hence a system that perceives its environment and takes actions that maximize its chance of successfully achieving measurable goals. These systems try to mimic cognitive functions associated to human mind, such as learning and problem solving.

A subset of AI is called Machine Learning, which is the study of algorithms that automatically improve through experience [71]. Machine Learning algorithms build a mathematical model based on sample data in order to make predictions or decisions without being explicitly programmed to do so [14]. An emergent form of Machine Learning has become increasingly popular: Deep Learning. Deep Learning is the application of Machine Learning algorithms inspired by the human brain, Artificial Neural Networks. Artificial Neural Networks are composed of multiple layers (hence the name "Deep") connected by nonlinear functions, which are recursively extracting higher level features from the raw input [35]

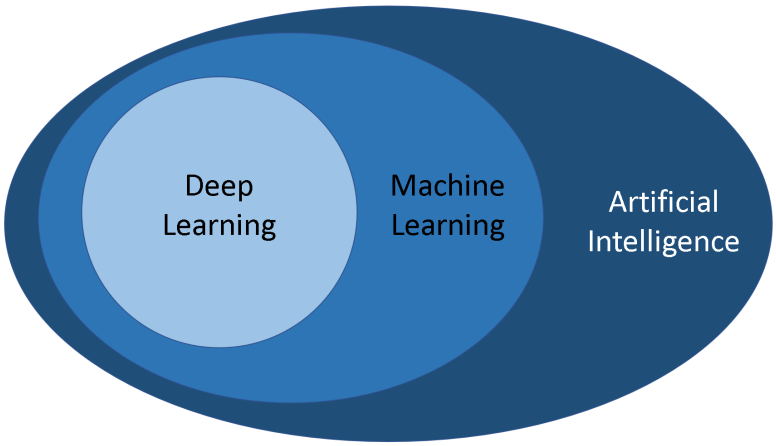


Figure 2.1. Artificial Intelligence, Machine Learning and Deep Learning

I focus primarily on Machine Learning so that we can include process knowledge as a method for enhancing predictive or classification performance. The idea is to help the algorithm to solve problems by providing as input already processed information in the form of engineered "features". Deep Learning, by comparison, tries to learn these features independently, and therefore requires a larger volume of data to attain acceptable performance.

2.1.2 AI and Machine Learning types

Artificial learning approaches are typically divided in 3 classes:

- **Unsupervised Learning** deals with problems with previously undetected patterns (e.g. PCA) or tries to cluster the data based on some similarities.
- **Supervised learning** is a process that maps an input to an output based on some example input-output pair previously linked [92]. In these situations, AI knows the right output of some data and tries to learn the connections

between the two, so that it is able to replicate it in the future with new unseen data.

- **Reinforcement Learning** is a process in which an agent explores its environment and take actions based on the maximization of its reward. Each action is taken based on the balance between exploration (gain more knowledge about the environment for more robust future actions) and exploitation (take the best action and collect the best reward based on the current knowledge).

We will deal with supervised learning, since the characteristics of the recorded engines are known in the training phase.
There are primarily two kinds of supervised learning problems: Classification and Regression (Figure 2.2):

- **Classification** is intended to predict an output based on a discrete number of possible results, namely the classes. A good example of that problem is the prediction of the number of cylinders of the engine, which is in our sample a discrete set of integers between 2 and 8. In our case a problem may arise if we do not have enough training data that are able to capture the variability of the engine models in the market. If we try to predict an engine with 12 cylinders, the algorithm is not able to do it at all, since it can consider only the classes it was trained with.

- **Regression** on the other hand, deals with continuous problems that may have a lot of possible outputs (e.g. engine power), and fit a function that is as close as possible to the data, in order to be able to replicate a measurement lying between previously seen data. For example, if we train our model with engine size of 2.0 liters and 3.0 liters, it could be able to predict an engine with size 2.6 liters, supposed that the problem is continuous and sufficiently smooth between 2.0 and 3.0.

There is not always a hard boundary between the situations in which it is better to use one of the two methods. For instance, when trying to predict the engine displacement in liters, we can treat this problem as a regression (output is a continuous variable between 0.9 and 5.7), or as a classification, by rounding down the output value to the unit, resulting in discrete and limited classes (1.0, 2.0, 3.0, 4.0, 5.0). If classes are few both in training and testing dataset, classification might be the best strategy.

2.1.3 Phases

The work is divided in two phases:

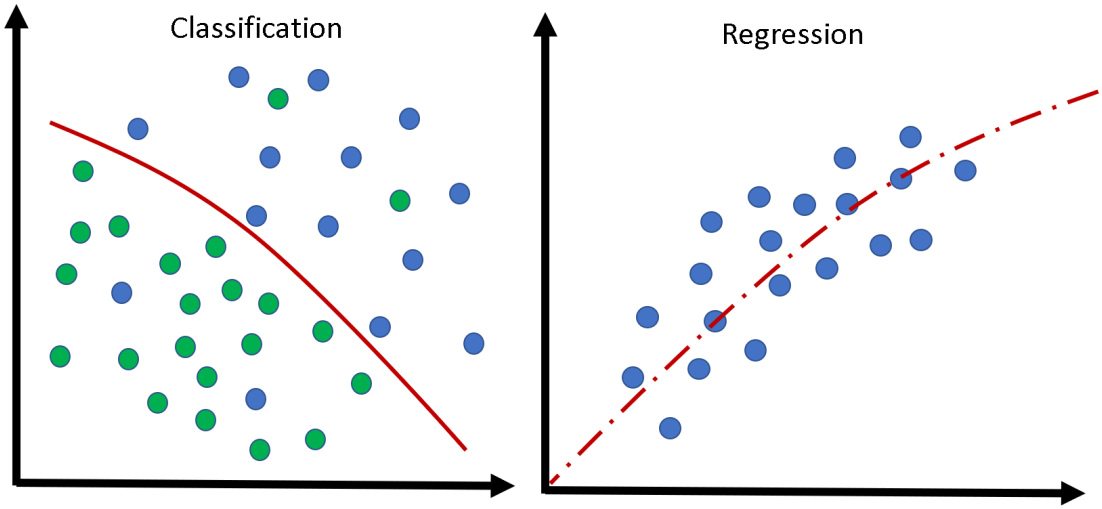


Figure 2.2. Classification and Regression problems

1. **Training Phase:** we provide some labelled data to the algorithm, and let it run through all the samples and its known outcomes to arrange its internal parameters and trying to fit the input to the output in the best way possible.
2. **Testing Phase:** we provide the algorithm with some unseen samples with no output, and we measure the performance of its predictions with real ones. For this reason, we have to pre-store some data with known output and provide only the input to the algorithm.

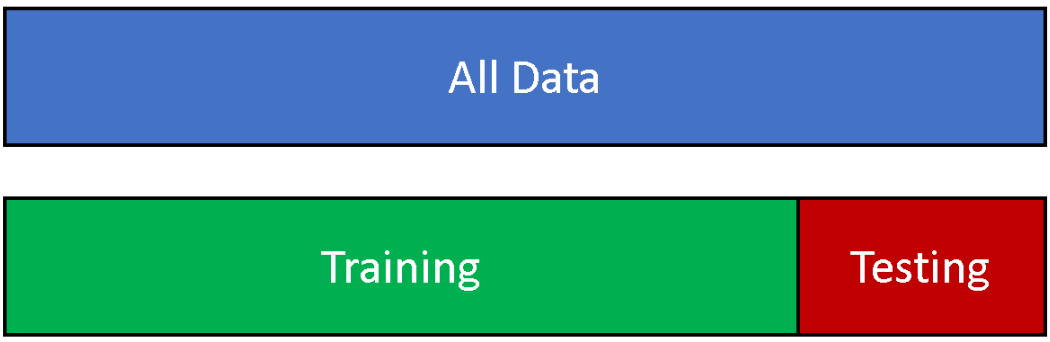


Figure 2.3. Splitting the data in training and testing sets

We need the algorithm to predict the results of field measurements, not of training ones. It is fitting parameters based on training data, but the performance is needed

on test data, which are the sample closest to field data we have. For this reason, we save a part of our data for testing purposes, as in Figure 2.3. In this regard, we want the algorithm to learn from training data, but to be able to generalize its knowledge and hence to predict new unseen test data. In fact, it is counter-intuitive a trade-off: we want that the algorithm learns from the data as best as it can, but not too specifically, otherwise it will face poor performance with unseen samples. This issue is described by the term **overfitting**, which indeed is the production of an analysis that corresponds too closely (or even exactly) to a particular set of data, but that may therefore fail to predict new data coming from future observations. This is the opposite of **underfitting**, which does not fit the data, but gives trivial (constant) outputs (Figure 2.4).

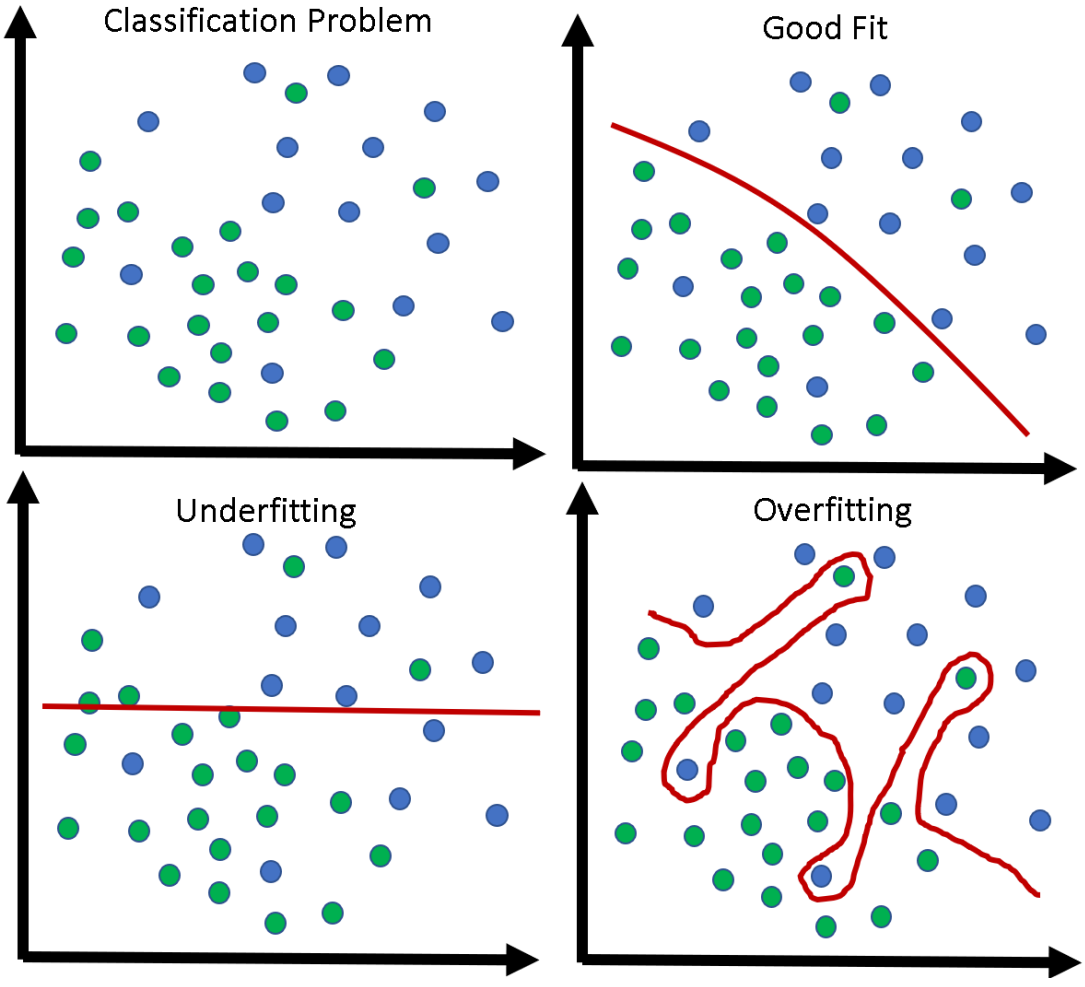


Figure 2.4. Underfitting provides trivial constant output. Overfitting produces an analysis that corresponds too closely to the training set

Overfitting is one of the typical issues in machine learning, and I have experienced it at some point of my work as well. In a more friendly way, we may see it as the algorithm lying to you. It pretends to produce very good results, but when it faces new challenges it is completely unable to perform it is not able to reproduce that performance. Overfitting can be mitigated with different techniques intrinsic in the specific algorithms (e.g. regularization), and its onset can be recognized and avoided with the use of a **validation set** - a portion of the train set not used during training phase but kept aside for temporary judging purposes, as explained in Section 3.1.

2.2 Data gathering

The primary task of this work is data collection. It is a continuous process, since more data usually means more generalizability of the model, and I implemented the software to be able to add more data afterwards and improve performance. An important aspect to keep in mind is that the more data are available, the more accurate the algorithm will be in predicting real-world recordings, and the closer the parameters will be fit to the optimal ones. In my case, I struggled to collect enough data in useful time for the deadlines of this work, but since our goal was also to provide a solid framework for future enhancements, I collected enough data to get acceptable results, and leave the opportunity open to collect more data and use the same software to better generalize the results.

2.2.1 Recording Environment

The main sources for the recordings are:

- YouTube videos
- Personal and friends' cars
- Mechanics' workshops

For this part of the work the data were recorded under defined conditions, in three possible locations:

1. Close to the engine compartment (with open hood),
2. Over the closed hood,
3. From the exhaust at the back of the car

Not in all conditions all recordings are available, and the software must be able to make up for that, considering the input location and engine speed as an additional feature.

2.2.2 Data Preparation and Labelling

Audio were captured from different sources, different microphones, different format and encoding, different sample rates and with some undesirable background noise or interruptions. There are some important considerations to take into account, namely some general concepts that applicable to every work done with data:

- **Garbage In Garbage Out (GIGO):** it is optimistic to expect that good results will come from bad data. Further, often it is impossible to catch a "Garbage Out", because even strong-performing algorithms may provide bad results if it is fed with bad input ("Garbage In")

- **Data Diversity:** As data come from very different environment, they risk to catch some information about the location, the microphone or the wind. If instead a large amount of data come from the same source, the results may be strongly biased.

- **Compression:** Often data come in a compressed way to save space and bandwidth, but strong compression may lead to considerable loss of information, that maybe are not captured by untrained human ears, but could be very relevant for an algorithm.

- **Consistency:** Source matters. We need to take care about the quality of the source, their compression methods and format, their recording purposes and strategies. For example, one may be tempted to get car sounds from websites for video-games developers or cover YouTube videos. Those are unfortunately often synthetic sounds or loopy ones, and may lead to problems. Every recording should be done roughly the same way to discard possible sources of bias.

Since data come often in different formats, I needed to artificially transform them to increase the consistency of the dataset, with both manual and automatic pre-processing techniques:

- **Manually** split recordings in multiple files, when some excessive background noise or interruptions were recorded. This was achieved using Open Source Audacity Software, which allowed to save stereo files in 32bit floating point uncompressed format (.WAV).

  I chose this format because it is one of the most informative ones and it is easily loaded from Python. I made it the standard for my work. It means that all new data added should have the same characteristics in order to be compared with my older data. Often data were presented in int16 format, and it caused troubles when processing them, because int datatype has several limitations in mathematical operations. The different splits of the same file

were saved using similar names (e.g. adding an increasing number at the end) so that I could consider these files as belonging to the same car, and reduced the risk of leakages of data from train to test samples.

- **Automatically** load files in Python using a function from the *scipy* library. Each sample is given an *ID* number to indicate a different recording. This step is needed because during manual pre-processing, the same car's recording is split into different files when some noisy and unwanted interruptions are caught and removed. I needed to group those recordings (called with very similar filenames) under the same *ID*, in order to put them under the same category among test and train split. If this is not done, overfitting problem may occur, because the same recording may be in both train and test. In this case, the algorithm would recognize the samples and give misleading non-representative results with artificially high performance numbers. For that purpose, I implemented an additional label to the file, called *ID*. Its generation is based on the audio filename, and since the WAV files must be ordered alphabetically in the folder, I increased the ID number from one sample to the next one only for non-small difference in filenames. That difference is computed with the Levenshtein distance, with a threshold set to 2. With the array loaded I needed to separate left from right channel, as most recordings were captured in stereo mode. This allowed me to have more data to train on as each channel was treated as independent. If those arrays coming from different channels of same recording were too similar in terms of their L2-norm, I averaged them as it was a mono audio. All data had to be consistent with the others when extracting features, hence each file is re-sampled down or up to the indicated sample rate (default is 48kHz). Python library *librosa* [76] is used for this purpose.

Once all samples were saved to a specific folder, those files needed to be labeled. Before explaining further how it is implemented, some specifications on the terminology are required, such that it is easier for the non-expert reader to follow.

- **Labels** are numeric or textual values corresponding to the characteristics of the engine (or recording, in general terms) we want to try to predict at the end (e.g. number of cylinders, engine displacement, fuel, ...). In supervised learning, labels are known in advance during training (learning) phase but unknown (at least to the algorithm) during real testing. If we (as AI supervisors) know them during testing, we may be able to judge the performance of the algorithm based its predicted labels compared to the real ones, called "ground truth". Different metrics can be used, as explained in Section 3.4.8.

- **Class** is the specific category the engine belongs to; they are the different instances the label can have. For example, considering the label of "number

of cylinders", the different classes are 3 cylinders, 4 cylinders, and so on. Each engine can belong to one class only for each label.

- **Feature** is the information about the engine the algorithm is able to compute in a systematic and deterministic manner. This information is known by the algorithm both in training and testing phases. In fact, they are not *predicted* but rather *extracted*. Some examples of features are the Fourier Transform of the signal, the statistics relating to it, and many more. These are the information the algorithm bases its choices and predictions on. Further details in Section 2.3.

- **Category:** Each feature can take different values, and those are called *categories*. Categories are to Features what Classes are to Labels. If a feature has continuous domain, a category can be considered to be a partition of that domain. This concept will be further developed in Section 3.3.3.

One user-friendly way (one of the goal explained in Section 1.7.2) to label samples is to use an Excel file called *Database.xlsx*, organized in the following manner (an example is shown in Figure 2.5:

- Each **line** is indexed by the filenames, which must be exactly the same as the list of filenames in the specified folder, both in names and amount.

- Each **column** represents a specific label I want to investigate, either as goal (e.g. number of cylinders) or as previously known feature (e.g. engine speed during recording)

- Each **cell** at the intersection between the label column and the filename row represents the class the specific recording belongs to, with respect to the label of the column.

All recordings are loaded, preprocessed and stored inside a dataframe object provided by *pandas* library with columns corresponding to labels, raw audio left channel, raw audio right channel (if present), features, and ID, as shown in Figure 2.6. Afterwards it is stored as *pickle* file [86], so that the user may be able to load it directly to the program, saving time and reducing complexity. The choice of *pickle* was dictated by some constraints of the previously used formats: *.h5* files cannot handle data that are too big, or that have too many vectors inside, while *parquet* has by the time of writing some problems in loading bug *pandas* dataframe. Pickle has some security and portability issues, but has few problems in saving or loading python objects. This feature has been crucial for my work, because since the recordings were all of different length they had to be stored as single array object inside a dataframe cell, and not with each value in a cell.

	A	B	C	D	E	F	G	H	I	J
1	Filename	fuel	disp	cyl	turbo	loc	Idle	cc	hp	OEM
2	1967 Ford Mustang 289ci V8 Hood Open in Garage 271HP 2.wav	G	V	8	0	O	1	4.7	271	Ford
3	1967 Ford Mustang 289ci V8 Hood Open in Garage 271HP.wav	G	V	8	0	O	1	4.7	271	Ford
4	2012 Dodge Ram 1500 - 4.7L V8.wav	G	V	8	0	O	1	4.7	310	Dodge
5	InfinitiQX80 V8 Normal 1.wav	G	V	8	0	O	1	5.6	400	Infiniti
6	InfinitiQX80 V8 Normal 2.wav	G	V	8	0	O	1	5.6	400	Infiniti
7	2010-Dodge-Charger-SXT-3-5-L-V6.wav	G	V	6		O	1	3.5	250	Dodge
8	2017 Chrysler Pacifica Minivan - Pentastar 3.6L V6.wav	G	V	6		O	1	3.6		Chrysler
9	Acadia Normal 1.wav	G	I	6	0	O	1	3.6	288	GMC
10	BMW fuori dal cofano a 3000 giri.wav	D	I	6	1	C	0	3	241	BMW
11	BMW fuori dal cofano in folle 2.wav	D	I	6	1	C	1	3	241	BMW
12	BMW fuori dal cofano in folle.wav	D	I	6	1	C	1	3	241	BMW
13	BMW vicino al motore in folle.wav	D	I	6	1	O	1	3	241	BMW
14	BMW vicino motore a 3000 giri.wav	D	I	6	1	O	0	3	241	BMW
15	BMW vicino scappamento a 3000 giri.wav	D	I	6	1	E	0	3	241	BMW
16	Frontier Normal 1.wav	G	V	6	0	O	1	4	261	Nissan
17	Pacifica Normal.wav	G	I	6	0	O	1	3.6	287	Chrysler
18	Range,aperto,folle.wav	D	V	6	1	O	1	3	245	Range Rover
19	Range,apetro,3000.wav	D	V	6	1	O	0	3	245	Range Rover
20	Range,chiuso,3000.wav	D	V	6	1	C	0	3	245	Range Rover
21	Range,chiuso,folle.wav	D	V	6	1	C	1	3	245	Range Rover
22	Range,scappamento,3000.wav	D	V	6	1	E	0	3	245	Range Rover
23	Range,scappamento,folle.wav	D	V	6	1	E	1	3	245	Range Rover
24	124,aperto,folle.wav	G	I	4	1	O	1	1.4	138	Fiat
25	124,chiuso,folle.wav	G	I	4	1	C	1	1.4	138	Fiat
26	124,scappamento,folle.wav	G	I	4	1	E	1	1.4	138	Fiat

Figure 2.5. Screenshot of Excel file where dataset filenames and labels are stored

	fuel	disp	cyl	turbo	loc	...	OEM	Audio Left	Audio Right	Sample Rate	ID
6	G	V	6.0	NaN	O	...	Acura	[0.006234304, 0.0037317928, 0.013585647, 0.001...	[0.022273397, 0.013643627, -0.0053483434, 0.03...	48000	6
7	G	V	6.0	0.0	O	...	Honda	[-0.009893003, -0.008414985, -0.0132224485, -0...	[-0.016149508, -0.0138136875, -0.024757233, -0...	48000	7
8	G	V	6.0	0.0	O	...	Toyota	[0.00860413, 0.009066186, 0.0045469864, 0.0120...	[0.024591707, 0.015258649, -0.009723709, -0.00...	48000	8
9	G	V	6.0	0.0	O	...	Nissan	[0.016811516, 0.024252981, 0.021591865, 0.0078...	[0.032968078, 0.037770737, -0.00900967, 0.0115...	48000	9
10	G	V	8.0	0.0	O	...	Chevrolet	[0.0053723236, -0.016260846, -0.043432903, 0.0...	[-0.07487101, -0.03856302, -0.027624898, -0.08...	48000	10
11	G	V	6.0	NaN	O	...	Mercedes	[-0.24846615, -0.26956996, -0.18428518, 0.0112...	[0.09333987, 0.08445487, 0.05374777, 0.0438953...	48000	11
12	G	I	4.0	1.0	O	...	VW	[-0.066726476, 0.045024067, -0.0027728472, -0....	[-0.031472325, -0.13565965, -0.088046834, 0.00...	48000	12
13	G	I	4.0	NaN	O	...	Ford	[-0.035479642, -0.033945624, -0.037133064, -0....	[-0.03509334, -0.031082414, -0.03442859, -0.03...	48000	13
14	G	I	4.0	0.0	O	...	GM	[0.08588839, 0.033957355, 0.055056904, 0.12077...	NaN	48000	14
15	G	I	4.0	0.0	O	...	Toyota	[0.05847142, 0.05474603, 0.06202994, 0.0728575...	[0.057044953, 0.05794362, 0.0675782, 0.0790105...	48000	15
16	G	V	6.0	0.0	O	...	Toyota	[-0.034869622, -0.040181503, -0.020789366, -0....	[0.00851955, -0.038091365, -0.06715259, -0.010...	48000	16
17	G	V	6.0	0.0	O	...	Toyota	[0.026978321, -0.0603477, -0.0023329034, 0.075...	[0.077444464, -0.059056602, 0.023858327, 0.136...	48000	16
18	G	V	6.0	0.0	O	...	Toyota	[0.055340573, 0.07721629, 0.027894998, -0.0337...	[0.091469765, 0.15563224, -0.08599947, -0.1806...	48000	16
19	G	V	6.0	0.0	O	...	Toyota	[0.041946843, 0.014312533, 0.018661065, 0.0357...	[0.05224266, -0.01844575, -0.036663923, 0.0283...	48000	16
20	G	V	6.0	0.0	O	...	Toyota	[0.0007497594, -0.028728645, -0.007713001, 0.0...	[0.09604268, 0.06427167, -0.081948474, 0.03976...	48000	16
21	G	V	6.0	NaN	O	...	Dodge	[-0.023011487, 0.007010173, 0.006491542, -0.00...	NaN	48000	17
22	G	I	4.0	1.0	O	...	Audi	[0.09970093, 0.06304932, 0.051971436, 0.046661...	NaN	48000	18
23	G	I	4.0	1.0	O	...	Buick	[-3.0517578e-05, 7.6293945e-05, -0.00010681152...	NaN	48000	19
24	G	I	4.0	1.0	O	...	Buick	[0.0, -1.5258789e-05, 3.0517578e-05, -3.051757...	NaN	48000	19
25	G	I	4.0	1.0	O	...	Buick	[1.5258789e-05, -3.0517578e-05, 4.5776367e-05,...	NaN	48000	19

Figure 2.6. The dataset in Pyhton is organized with the columns shown here thanks to the *pandas* [125]. Note that the audio columns store arrays inside and not values. If the file was mono-channel, "Audio Right" will display a NaN

2.2.3 Train - Test Split and Chunking

At this stage, all audio data were loaded and resampled to the same sample rate (e.g. 48000Hz, so that 1 second of recording correspond to an array of 48000 elements), but each of them have a different length. As outlined in Section 2.1.3 the first next step at this point is the separation between train and test split. I used a function from the *sklearn* library [16], allowing to not only split the dataset, but also to stratify based on the labels, thereby keeping similar same class proportions of class in the both train and test datasets (e.g. 60% 4 cylinders, 20% 6 cylinders, 20% 8 cylinders). Two main precautions must be considered in this phase:

1. Split the dataset based on the *ID* and not as raw dataset, in order to avoid that the same car is in both splits, causing overfitting issues. To address that I passed the vector of *IDs* to the splitting function, and divided the samples based on their *ID*.

2. Each class must be represented by at least two samples, so that one will appear in both the training and testing set, otherwise the stratification would not work.

As explained before, the more abundant data leads to better results for most AI algorithms. I decided to augment the number of samples at my disposal by splitting the audio file in equally long sub-chunks (e.g. 1 second, 48000 elements). This simple data augmentation technique helped to reduce the complexity of dealing with samples of different lengths as well, and reduced per-segment feature computation time. Different lengths were tried, from 10 milliseconds to 10 seconds. For each tested length a new dataframe was created and stored. The file was saved with a name following a pattern allowing for future simulation runs to recognise the specific *pickle* file and load it directly. The three information indicating the way I did the chunking are denoted as follows in the filename:

$$chunked\_dataset\_dur < chunklength > \_dev < developingmode > \_ < chunkspersample > \_ < split > \_ < stratificationlabel > .pck$$

where

- $< chunklength >$ is the duration in seconds of the sample (e.g. 2 seconds).
- $< developingmode >$ is a way to allow the programmer to load a smaller database and test their algorithms on it, without taking too much time and resources (e.g. 1, indicating we are using developing mode).
- $< chunkspersample >$: when using developing mode, I only loaded from the database the first ¡chunks per sample¿ chunks from every sample, so that I still had the variability coming from different samples, but not the burden of a lot of data.

- $< split >$: after splitting *ID*s in train/test, I saved both databases according to where its *ID* ended during the random split. Therefore, for each conditions two disjoint databases were saved.

- $< stratificationlabel >$: As I divided test and train splits with stratified label (same class percentage in both datasets) I needed to keep this information as well.

For example for a $< chunklength >$ of 2 seconds, with $< developingmode >$ and loading only the first 10 $< chunkspersample >$ (20 seconds of audio) from the ID in train $< split >$ stratified based on $< stratificationlabel >$ cyl, the resulting chunked file was called

$$chunked\_dataset\_dur2\_dev1\_10\_train\_cyl.pck$$

The above parameters can be directly controlled by the used by means of the Excel file called *Parameters.xlsx*, in its sheet called *Load*. The first rows (shown in Figure 2.7) are used to manage the loading and chunking procedure. Here I explain the meaning of each parameter:

- *audiopath* is the folder path where raw audio files (.wav) are located. This parameter is useful if the user want change the dataset and has to reload the audio

- *mainpath* is where the *Parameters.xlsx* file *Dataset.xlsx* and all Python files are to be located

- *storagepath* is selected because the user may want to work by syncing all files in a Cloud platform. However, big files are generated and syncing them would require excessive bandwidth and remote storage. For this reason, *storagepath* should be set to a offline position, possible with large space available (e.g. external Hard Disk or SSD)

- *inWindows* is a boolean value used to indicate whether the program is run on a Windows OS machine (= 1) or on Unix based one like Linux and MacOS (= 0). It is important to specify because the file-system is handled differently

- *chunk_duration* is the required duration in seconds of each single chunk the user wants to split the original samples in. It is an important parameter as shorter chunks means more data, but also less informative one (it is difficult to recognize something from 10 milliseconds of audio)

- *developing_mode* is a boolean variable to set whether to use a smaller version of the dataset, in order to save computing power and time. It is used for developing purposes only to test the software. Its effect is coupled with

chunks_per_sample which sets the maximum amount of chunks to split the raw sample in. Generally the number of chunks per sample is equal to the duration of the sample divided by the desired duration of each single chunk, rounded down. In this case a maximum boundary is set, so that the resulting chunked dataset is small and easier to test and debug

- *test_train_stratify* is one of the label to specify so that roughly the same percentage of classes of that specified label is present in train and test. This is basically a way to split train and test dataset, so that they have the same proportions and results evaluation is more representative
- *sample_rate*: each sample is recorded with its own sample rate, but they need to be all sampled with the same one
- *test_size* simply indicate the size of the test dataset. Empirically, it should be between 20% and 40%

	A	B	C
1	Load	audiopath	/home/remotediag/Audio
2		mainpath	/home/remotediag/OwnCloud/DeepTech Lab/U
3		storagepath	/home/remotediag/Storage
4		inWindows	0
5	Features	sample_rate	48000
6		test_size	40%
7		train_test_stratify	cyl
8		developing_mode	0
9		chunks_per_sample	1
10		chunk_duration	2.00
11		features_to_avoid	['DWT', 'FFT', 'MFCC']
12		feature_scaler	[StandardScaler()], [MinMaxScaler()]

Figure 2.7. Control File: Loading and chunking parameters section

2.2.4 Database Exploration

At this stage, we may already want to gain some insights on the database, such as statistics on the labels, including:

1. General information about the data
2. Missing values
3. Intra-label relationships
4. Inter-label relationships

General information about the data

To begin with, I present here a table showing some general information about the data collected.

Information	**Label**	**Quantity**
Audio samples	.WAV	268 files, 19683 seconds (5h 25min), 9.1 GB
Labels	Fuel	*Classes:* Diesel (D), Gasoline (G)
	Turbo	*Classes:* Yes (1), No (0)
	Cylinders	*Classes:* 2,3,4,5,6,8
Car Makers	*OEM*	23 different classes

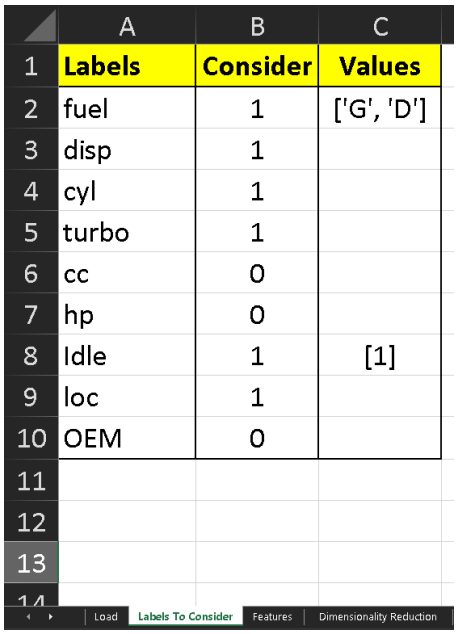
Missing Values and example view of database

After the pre-processing phase, the database results in a long table of n samples of the same length, each with its information stored in a dataset. The database is not complete, because some labels are unknown. There are several ways to infer those missing values, but the all of them may lead to some problems. For this reason, I implemented a function to select from the database only the rows having a defined class for the label(s) I wanted to investigate during that simulation run. A sheet called *Labels To Consider* in the Excel file is used for this purpose. As we can see in Figure 2.2.4, the first column indicates the names of the labels present in the database, then the user sets a boolean value (0 or 1) to indicate respectively if he want to consider that label (1) or not (0). There is a problem that appeared in this phase, namely if there are some classes with only one sample. Here the stratification is not possible, and this sample is present only on train or test. If it is in train dataset, we are not able to see if it can recognize that class among the testing samples, and if it is in testing we are sure that it is not able to recognize it, because the algorithm was not trained on that class. Therefore, there are two possibilities for the third column:

1. To leave the cell blank, so the algorithm will select all samples that have a valid value for that class, dropping only the rows containing empty values (NaNs).

2. To list the classes we want to investigate, so that the function will drop all NaNs and all values not inside that specified list.

In the specific case of Figure 2.2.4, I avoided taking the columns of *cc*, *hp* and *OEM*, and selected only samples recorded at idle (the value of corresponding *Idle* column is equal to 1) and samples with *fuel* labels being G or D, others were dropped.



	A	B	C
1	**Labels**	**Consider**	**Values**
2	fuel	1	['G', 'D']
3	disp	1	
4	cyl	1	
5	turbo	1	
6	cc	0	
7	hp	0	
8	Idle	1	[1]
9	loc	1	
10	OEM	0	
11			
12			
13			
14			

Load | Labels To Consider | Features | Dimensionality Reduction

Figure 2.8. Labels to Consider - Control File

Intra-Label relationships - Class Imbalance

One interesting and important property is the balance of the classes. This considers how each class of a specific labels (e.g. class 4 of label *number of cylinders* is represented inside the database. If we have for example a lot of samples with 4 cylinders (e.g. 90%) and very few with 3 cylinders (e.g. remaining 10%), we encounter a so called *Class Imbalance*. It is important to consider its effect on the results, because sometimes AI appears to perform well but instead provides trivial results. Considering this example (90% 4 cyl, 10% 3 cyl), if the outcome of our performance test results in an accuracy of 90%, it may simply mean that the algorithm is predicting *4 cylinders* for each sample, and it is 90% of the time correct! On the other hand, if the database is a set of representative samples of the engine population, class imbalance may be intrinsic in the problem we want to solve: following the same example before, this is outstanding if the algorithm will face 90% 4 cylinders engines when used in a real environment.

Here I present some pie charts and histograms exploring the class distribution of our sample data. First of all, we explore the appearance of each cars' maker (OEM) in the dataset 2.9. We can see a predominance of Fiat vehicles, and then a rather equal amount of other OEM with fewer samples. Since our data come primarily from the U.S., Diesel engines are poorly represented in the database, resulting in a strong class imbalance. To attempt to rebalance the data, Diesel engine samples were captured and processed from YouTube (Figure 2.10).

Value	Count	Frequency (%)
Fiat	39	24.8%
Buick	17	10.8%
Nissan	11	7.0%
Ford	10	6.4%
Hyundai	9	5.7%
BMW	8	5.1%
Audi	7	4.5%
Mazda	7	4.5%
Citroen	6	3.8%
Peugeot	6	3.8%
Other values (13)	37	23.6%

Figure 2.9. OEM Appearance in the Dataset

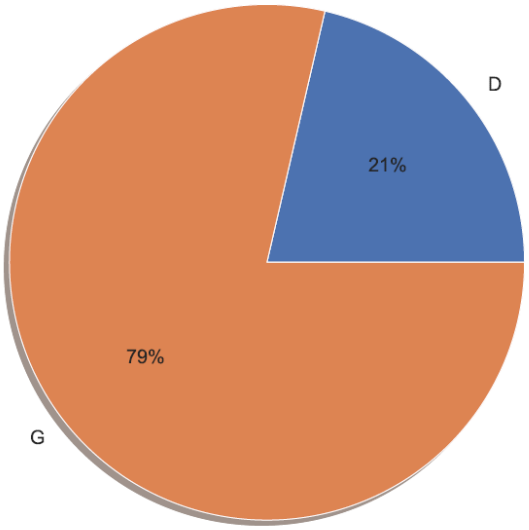


Figure 2.10. Fuel Type - Classes Distribution: most of the engine samples are gasoline powered

As we can see in Figure 2.12, I had only few classes for what the cylinder amount is concerned, and one can observe that the class of 4 *cylinders* is more represented

than the others, but just because they are more common on the market with respect to other cars, perhaps due to increasingly-strict fuel economy and emissions standards globally. One problem of having very few samples of one class (e.g. 5*cylinders*) is that they may not be represented in either test or train database, preventing the performance evaluation to be representative on all classes. The

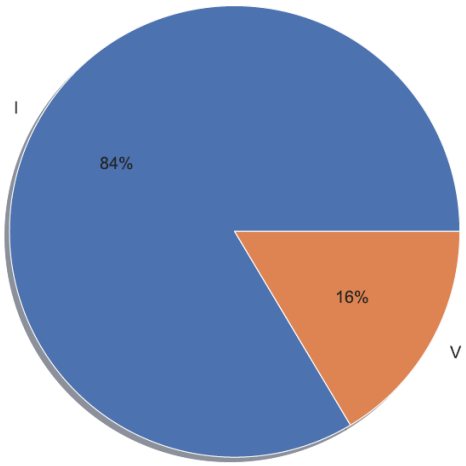


Figure 2.11. Engine Shape - Classes Distribution: most of the engine samples' shapes are *Inline*

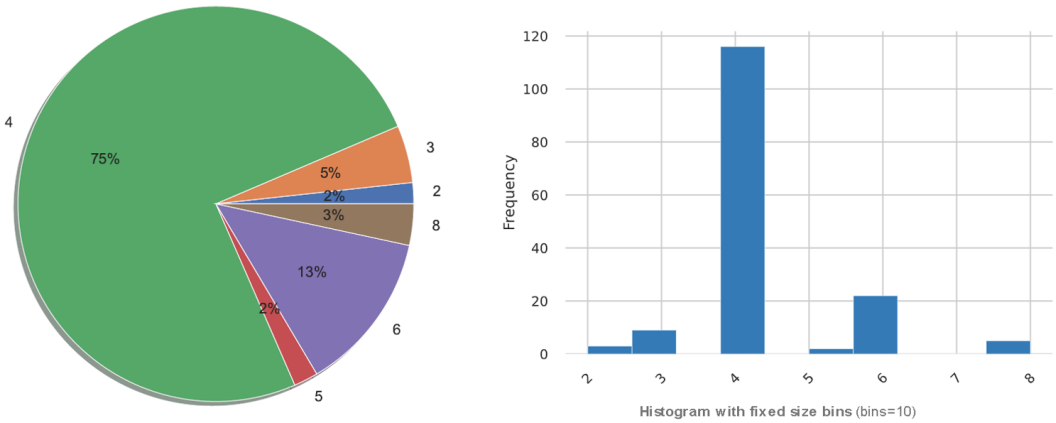


Figure 2.12. Number of Cylinders - Classes Distribution: most of the engine samples have four cylinders

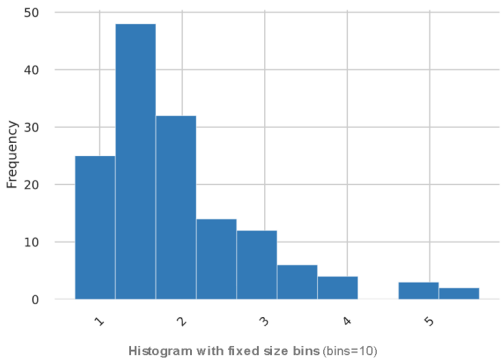
distribution of the engine displacement and engine power is instead more granular

as it is a continuous value (Figures 2.13 and 2.14). I therefore had to make a choice:

1. Keep classes this way and use classification algorithm, if we consider that each class has its own specific characteristics and there is no similarity of continuity from one class to the other: if we misclassify an engine of size 2.4 as 2.5, we are not closer to the truth as if we say 1.0. I tried this approach but granular multi-class classification is a very hard problem, and results were poor.

2. Round the class to the closest integer and use classification algorithms. This was my main approach, but results were not satisfactory at the time of publication.

3. Consider it as a regression problem to eliminate the need for rounding the classes. This is a good approach. However, since this work was mainly focused on classification problems, that option is not considered and is left out for further development of the software.



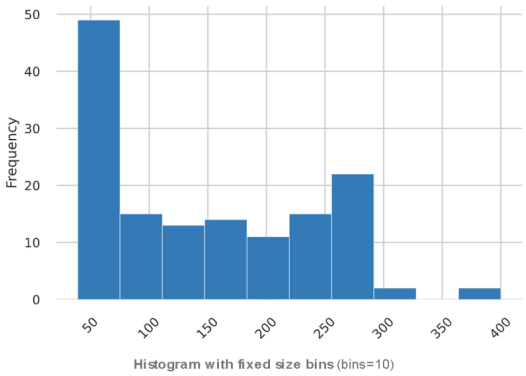
Descriptive statistics

Standard deviation	86.5572879
Coefficient of variation (CV)	0.576430502
Kurtosis	-0.7655458717
Mean	150.1608392
Median Absolute Deviation (MAD)	59
Skewness	0.5587421146
Sum	21473
Variance	7492.164089

Figure 2.13. Engine Displacement - Classes Distribution and Statistics

Inter-Label Relationships - Correlation

It may be also interesting to look at the relationship between labels. A good example is the relationship between the engine displacement and its power. They are of course not redundant information, but the prior information about one of the two can help guessing the value of the other. If we know that our engine is a 1.0 liter displacement, the probability that its power is over 300hp is very low. These insights shown in the following figures helped me to understand that the best process to identifying all the characteristics of the engine is iterative. By using the example above, once you predict the number of cylinders you will use that information as



Descriptive statistics

Standard deviation	0.9750840231
Coefficient of variation (CV)	0.5005705604
Kurtosis	2.282412046
Mean	1.947945205
Median Absolute Deviation (MAD)	0.5
Skewness	1.4422595
Sum	284.4
Variance	0.9507888521

Figure 2.14. Engine Power - Classes Distribution and Statistics

feature to predict the engine displacement more reliably, and afterwards we use those two information (cylinder amount and engine displacement) as features for predicting the engine power, and so on. I came to that idea after looking at the relations among labels in a more detailed way, as shown and explained in Figures 2.15 and 2.16.

To measure the intuition coming from the previous charts, I used a statistical measure called *correlation*: the degree to which two variables move in relation to each other. The kind of relationship among the labels was already clear, but it is always important to answer that question in a quantitative way. I computed the Pearson Correlation Matrix among the labels, obtaining the matrix in Figure 2.17. The Pearson's correlation coefficient is a measure of linear correlation between two variables. It's value lies between -1 and +1, -1 indicating total negative linear correlation, 0 indicating no linear correlation and 1 indicating total positive linear correlation. Furthermore, it is invariant under separate changes in location and scale of the two variables, implying that for a linear function the angle to the x-axis does not affect the correlation. To calculate Pearson Correlation Coefficients for two variables X and Y, one divides the covariance of X and Y by the product of their standard deviations. In Figure 2.18 we can see that the engines of my database are fairly small, that they have a rather linear relationship between cylinder amount, engine displacement and engine power. Label turbo is good represented whereas cylinder amount is quite unbalanced.
In Figure 2.19, on the other hand, we can see how those distributions influence the fuel type of the engine. We can observe that if the engine is turbocharged, it is more likely that it is a diesel powered one. In facts, red curves are higher when "turbo" is equal to 1 (bottom right corner). For what Engine Power is concerned, Diesel engines have a narrower range of operation, and the same is valid for engine displacement and cylinder amount as well. We know from experience that very

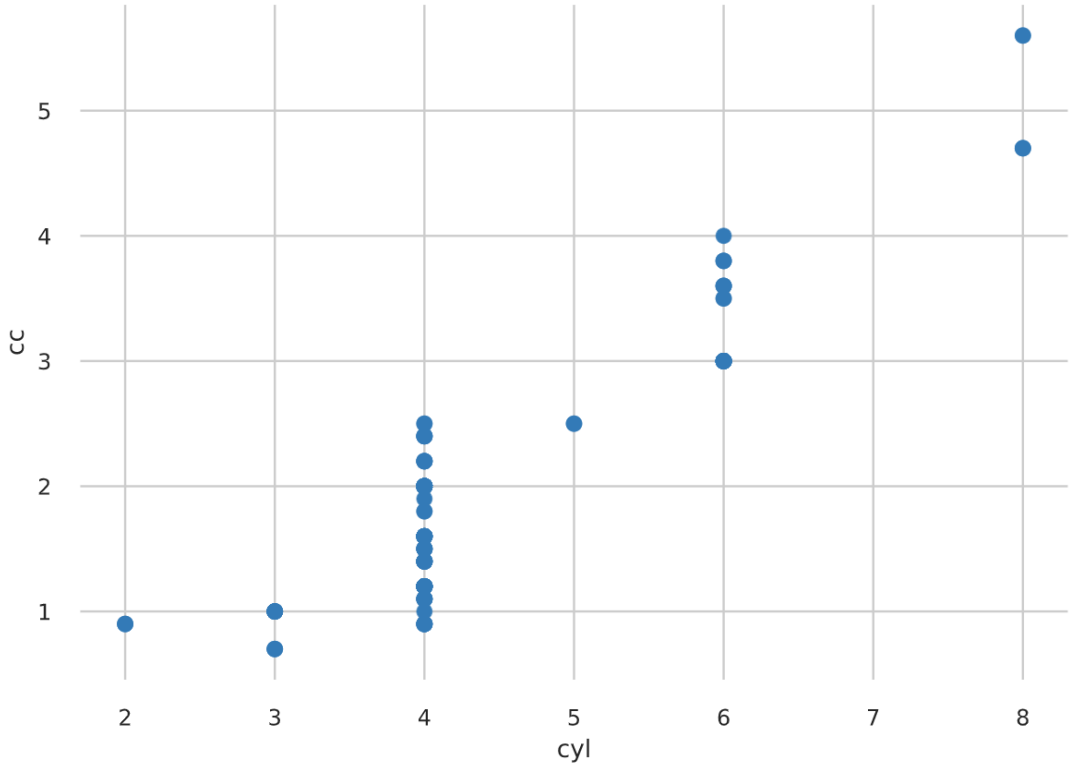


Figure 2.15. Cylinder Amount vs Engine Displacement: Strong correlation observed

small engines are generally gasoline, whereas big ones are diesels. However, I did not record any truck or van engine, so the cars having big engines (especially in the US) are generally "muscle cars" or high performance ones, which tend to be powered by gasoline.

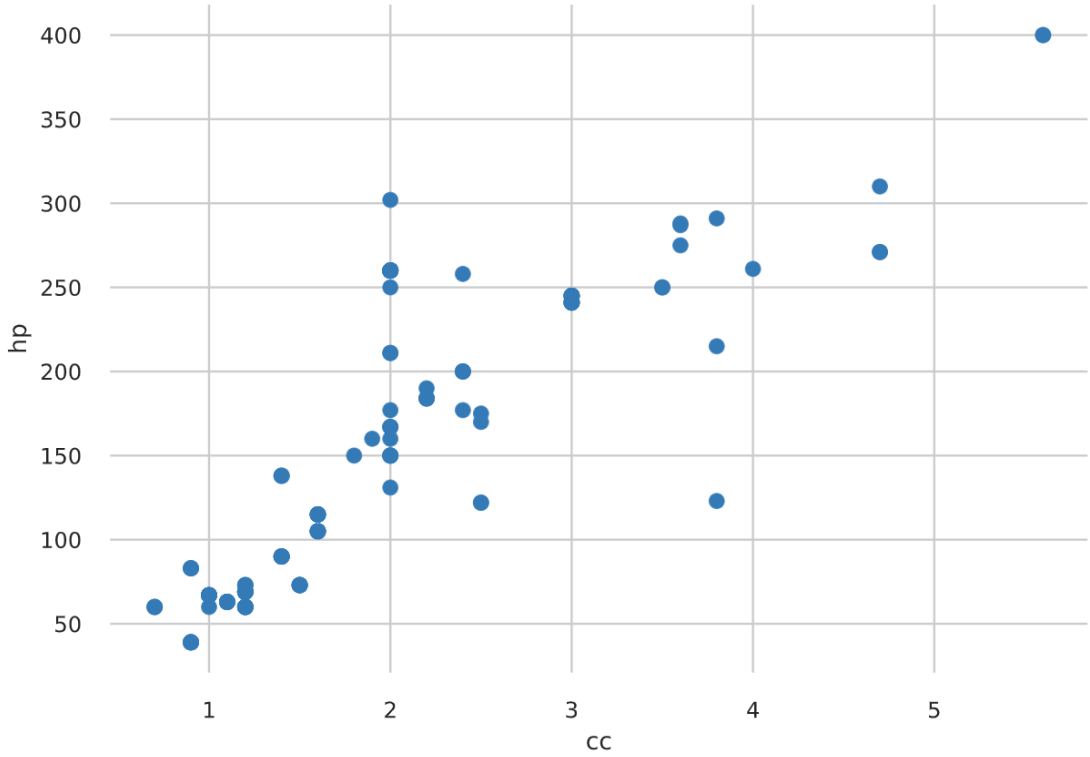


Figure 2.16. Engine Displacement vs Engine Power: Strong correlation observed

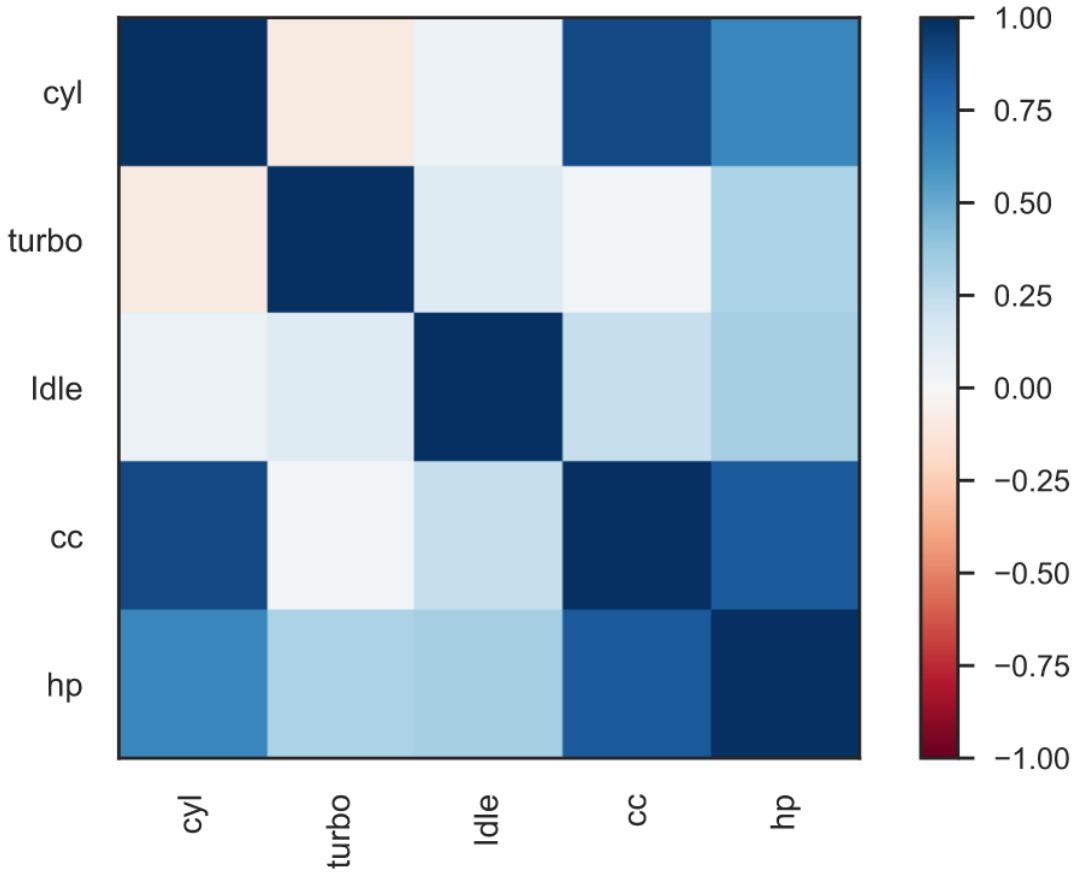


Figure 2.17. Correlation among Labels: the color of the cell corresponding to the interception of two labels is colored based on its value. Diagonals represent the correlation of the labels with themselves, so it is equal to one (dark blue). High correlation can be observed between engine power (*hp*), engine displacement (*cc*) and cylinders amount (*cyl*), whereas *turbo* and *cc* seem to be independent. *cyl* is correlated more to *cc* than to *hp*, and also more that *cc* is to *hp*

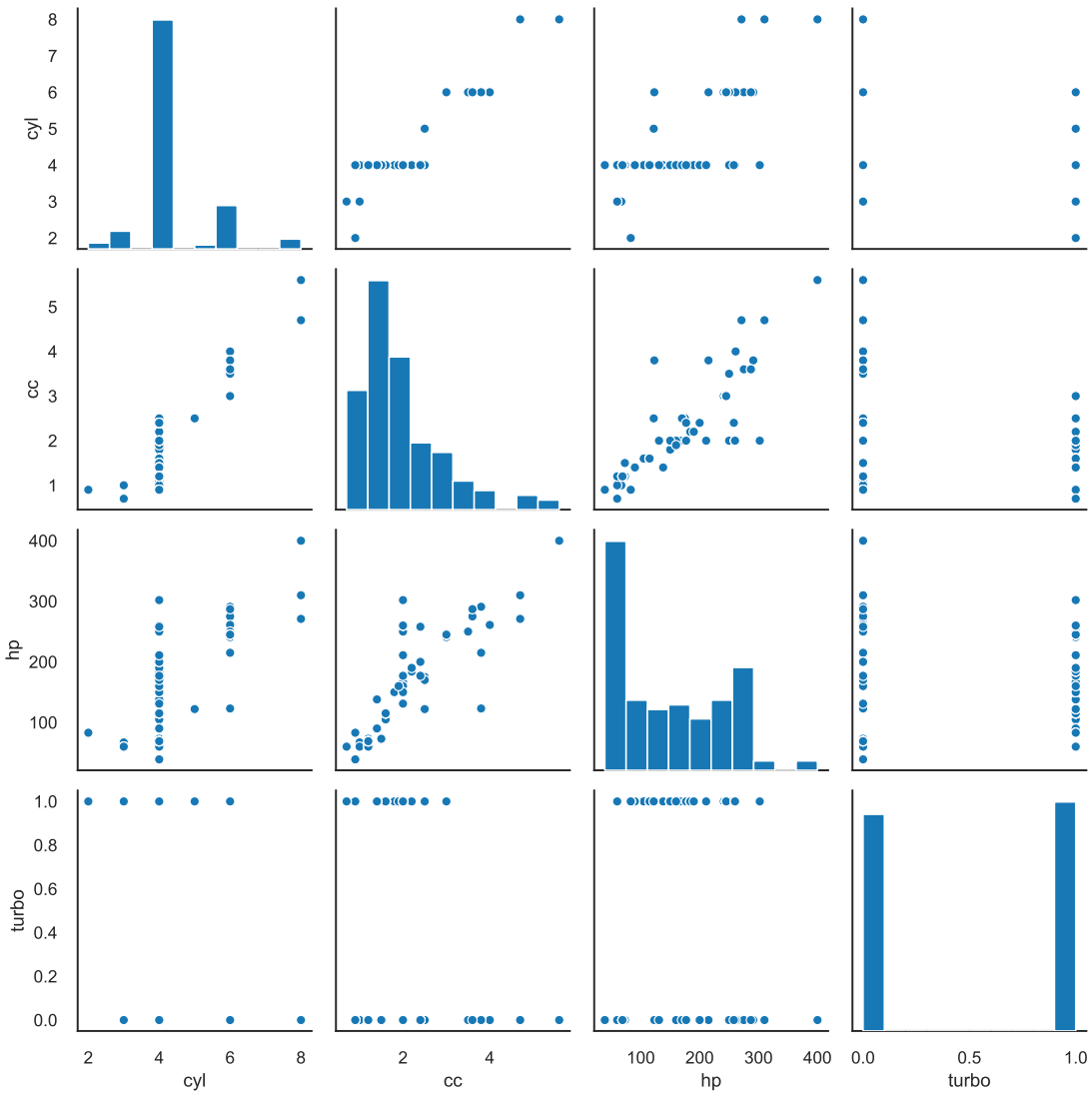


Figure 2.18. Complete view of distribution of labels: Size of the engines (cc) is more frequently small and tends to be linearly correlated with power and cylinder amount. Moreover, larger engines tend to be naturally aspirated. Powerful engines, on the other hand (hp) and evenly spread among turbo and naturally aspirated, but with a higher density in medium power turbocharged engines. 4 cylinder engines are more spread in terms of power, but more compact in terms of size. On the other side, 3 cylinder engines are mostly small and low powered

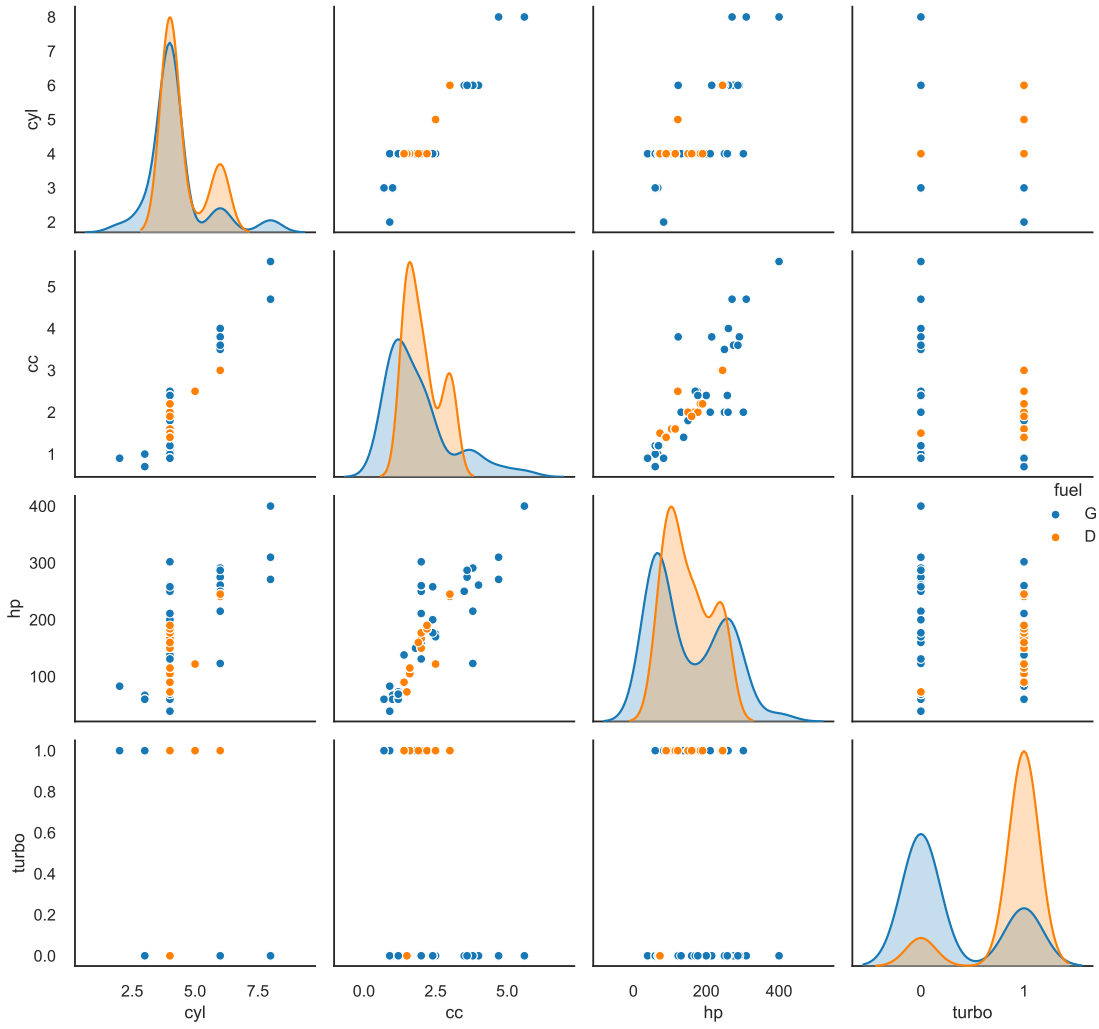


Figure 2.19. More in depth statistical relations among labels, highlighting label *Fuel*. As we can see, most diesel powered engines are turbocharged, and the opposite for gasoline. Then we can observe that big gasoline engines tend to be naturally aspirated, whereas smaller ones are turbocharged

2.3 Feature Engineering

In machine learning, algorithms often benefit from process knowledge. The way process knowledge is "captured" in algorithms is through thoughtful feature engineering. Here, we explore the types of features generated and the method through which we do so. Informative characteristics of the audio sample are generated in a deterministic and repeatable way. The value extracted is called *Feature*, and since we extract many of them from each sample, they are referred as *Feature Vector*. In machine learning, the *feature* is often a scalar (e.g. audio volume, mean, number of peaks, etc.) and *Feature Vector* is the array containing multiple features, but in more complex situations features can be vector themselves (e.g. Fourier Transform). Therefore our *Feature Vector* is a vector containing vectors, causing several problems in applying standard techniques, as we will see later. I tried also to use scalar features only, but when removing vectors such as those created by the Fourier Transform, results suffered.

In the following sections I will provide a short overview on the features I decided to extract, a simple explanation of their characteristics, and some charts representing the mean trend of the feature sorted by class. Additionally, I will present an explanation of the scaling procedure, as well as how the framework is dealing with custom-made functions in a user-friendly way.

2.3.1 Feature Extraction

Feature Extraction is crucial and needs a lot of attention from the AI designer because it influences the performance of the outcome. If we extract the wrong features, with wrong length, or wrong scale, AI may not be able to classify the engine at all. It is as if we want to classify the an animal from a picture, and the only information we are able to extract that it is brown, which is clearly useless even for the most experienced person. The research for the best features to extract from an audio file is an important field of study in machine learning research, especially in the last years due to the rise of Natural Language Processing needs. During the development of the survey on vibroacuostic diagnostics, I had the chance to get exposed to some of the research best practices, incorporating ideas from other researchers to implement my solution. The information about the features to to extract are again provided by the user in the *Parameters.xlsx* file, sheet *Features*. I show a screenshot of an example of the sheet in Figure 2.20.

Discrete Fourier Transform (FFT)

The Fourier Transform is a mathematical operation which decomposes a function in its constituent frequencies. The discrete version is used when dealing with digital

	BA	BB	BC	BD	BE	BF	BG	BH	BI	BJ
1	7.1 Spectral Rolloff		7.15 Spectral Bandwidth		7.16 Spectral Tonnetz		7.2 Spectral Contrast		9 Spectral Centroids	
2	used	1	used	1	used	0	used	1	used	1
3	function	librosa.feature.s	function	librosa.feature.spectra	function	librosa.feature.tonnet	function	librosa.feature.spectra	function	librosa.feature.spectral_centroid
4	sr	48000.00	sr	48000.00	sr	48000.00	sr	48000.00	sr	48000.00
5	hop_leni	100801.00	hop_leng	100801.00			hop_length	100801.00	hop_leni	100801.00
6										
7										
8										
9										
10										
11										
12										

Figure 2.20. Control File: Features

signals, and more specifically the Fast Fourier Transform (FFT) is used in this case.

$$X(f) = \sum_{k=0}^{N-1} x_k e^{-i\frac{2\pi}{N}kq} \qquad \text{with} \quad q = 0,1,\ldots,N-1$$

Since the engine is a rotating machine, frequency is one of the crucial aspects when dealing with cylinder counts and other aspects. The human ear is able to differentiate differences in frequencies, and it may be one of the aspects helping experts humans to recognize the engine characteristics. Since I wanted to put my AI algorithm in the conditions of recognizing the engine potentially as good as human expert (or better), we need to give it the signal decomposed into frequencies. Some workarounds are needed to take care of the borders of the signal to avoid aliasing when the chunk length is not a multiple of the signal period: windowing. This procedure is the multiplication of the signal by a window signal like the Hann's one, shown in Figure 2.3.1

$$w(s) = 0.5 \cdot \left(1 - cos(\frac{2\pi n}{N-1}\right)$$

$$\text{with} \quad n = 0,1,\ldots,N-1 \quad \text{and} \quad N \text{ being the length of the signal}$$

This is not strictly necessary, but it may help the quality of the feature and hence the robustness to overfitting. In fact, the algorithm will not focus on some specific and random boundary effects. In fact, if it focus only on some specific aspect, it may find that pattern repeating in some train samples and use it as indicator for the prediction. But since this pattern is random, it may not be present (or even misleadingly present in test samples). According to the Nyquist Theorem, the useful length of the vector coming out from the FFT is half of the sampling frequency. It means that for an audio chunk of 1 second sampled at 48000Hz, a feature of 24000 elements is created. Many algorithms would struggle with such a high dimensionality. On the other hand, some frequencies may not be caught if the chunk length is too short, because the minimum frequency is inversely proportional

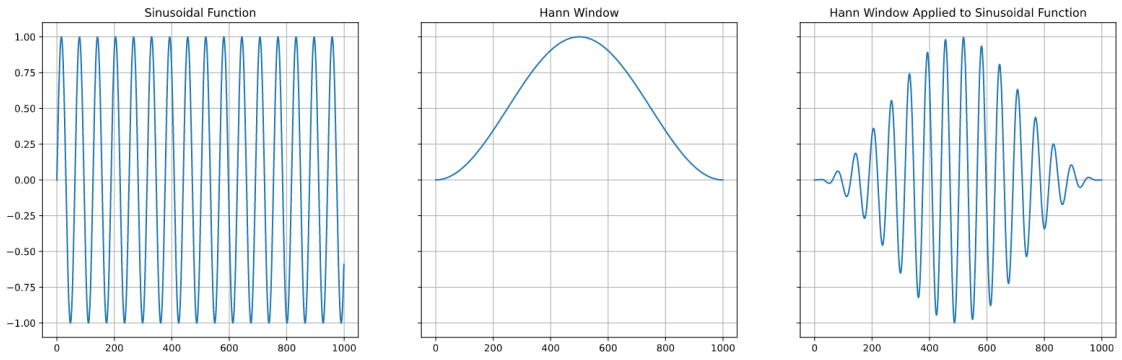


Figure 2.21. Hann Window and its effect

to the length of the chunk. I therefore implemented a function with the aim of binning the feature vector with bins of equal size, with a height equal to the average amplitude of the point inside that interval.

The algorithm will extract other features that will depend on the it (e.g. its

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	1	FFT	1.1	FFT Kurt	1.2	FFT Skew	1.3	FFT Var	1.4	FFT SE	1.5	FFT Mea	1.8	FFT Binned
2	used	1	used	FFT	used	FFT	used	FFT	used	FFT	used	FFT	used	FFT
3	function	my_fft	function	scipy.s	function	scipy.s	function	scipy.s	function	scipy.s	function	scipy.s	function	my_binning
4	w	scipy.signal.windows.hann											sampling_rate	48000
5													bin_len_hz	50
6														
7														
8														
9														
10														
11														
12														
13														
14														
15														
16														

Figure 2.22. Control File: FFT Features. String “FFT” in cell “used” indicates that we extract features from the already extracted FFT

skewness). Since FFT is already computed and stored in a column of our dataset and since we need to optimize the computational resources needed for features extraction, we don’t want to recompute FFt or each other feature dependent on it. For this reason, I developed a system in Excel that allows to use that previously calculated function (in this case FFT) and apply the statistical function on top of that instead of the raw signal. Figure 2.22 shows how this is done in practice. As we can see, we have a table containing:

- Name of the feature (FFT)
- Function to be used (my_fft)

- Parameters to provide to the function (windowing function in this case)
- Flag to set whether it has to be used or not. For normal functions, this flag is boolean, but I introduced a way to save time when extracting features with an additional value. Therefore, the flag may take three possible values:
    1. **1:** The feature vector is computed and stored inside the matrix of the features
    2. **0:** the feature calculation is simply skipped
    3. Name of the **primary** function: the feature is computed starting from a previously extracted feature vector. For example (as in Figure 2.22), the Skewness of FFT will have "FFT" in the "used" field, instead of "1", and only the *skewness* function in the "function" field, instead of $skewness(FFT(signal))$

There may be some situations when we do not need the primary feature (e.g. complete FFT) but only the ones computed from that one (e.g. a binned version of it, skewness, and so on). For that reason, I put a field in *Load* sheet of *Parameters.xlsx* file to provide this opportunity. The list of feature provided in *features_to_avoid* field will be ignored and not put into the X matrix. With X matrix we mean the matrix the algorithm will be trained on, i.e. a matrix where each row is representing a sample and each column is a feature of that sample. For example column 1 may be the skewness of the FFT, columns from 2 to 138 the power spectral density, and so on. It results in a matrix of size $n_{samples} \times m_{features}$ that is generally called X. By y we mean a matrix of size $n_{samples} \times l_{labels}$, and corresponds to the output values we want to predict. As we typically want to predict one label at time, y will be of shape $n_{samples} \times 1$ (figure 2.25). The following figures (2.3.1 and 2.24) show how the features computed from the FFT vary, by grouping the values by class. FFT is plotted averaging over its class, and the standard error is displayed as well, in order to understand if there is substantial statistical difference in the frequency spectrum among classes. FFT Binned is shown as line plot, each point of which is the mean of the points of all samples for that class. Also the standard deviation for each feature point is displayed, so that one can see the variability of those values. We observe higher frequencies in gasoline engines but with high variability, and on the other hand lower amplitude at higher frequencies for diesel engines, with lower variability. On turbo, instead the difference seams to be not appreciable. Remember that all recording were captured at idle conditions. By looking at the number of cylinders, we can see that the behavior at higher frequencies is probably governed by I4 engines, representing the vast majority of the sample. It means that maybe those higher frequencies are not caused by solely gasoline engines but rather by I4 ones, which are mostly gasoline. For features that are represented by single values, a scatter plot is more appropriate, showing the mean value for each

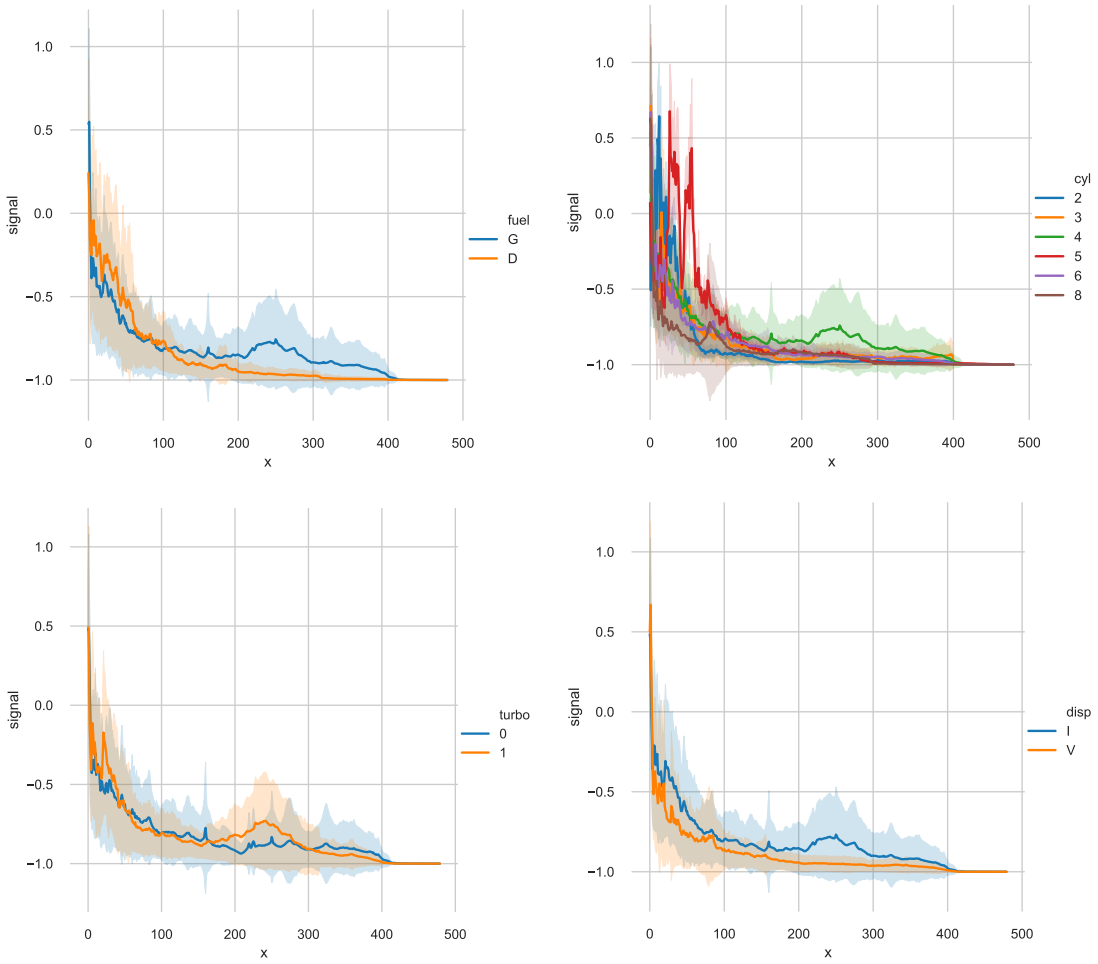


Figure 2.23. FFT Binned and averaged by Fuel, Number of Cylinders and Turbo

feature. Coupled to that, a plot showing the standard deviation is needed, in order to understand the variability of each single value and see if we can differentiate the classes easily, just by drawing a threshold for one single feature.

Wavelet Decomposition

Wavelets address some limitations of the Fourier Transform. FFT outputs a chart of frequencies, but without knowing when that frequency happens, the time resolution of the signal is lost. Short-term Fourier transform attempts to solve this problem by breaking the signal into shorter segments of equal length and computing the Fourier transform of each shorter segment.

However, the problem with this approach is that STFT has a fixed resolution: the smaller we make the window, the more precisely we can identify the time at

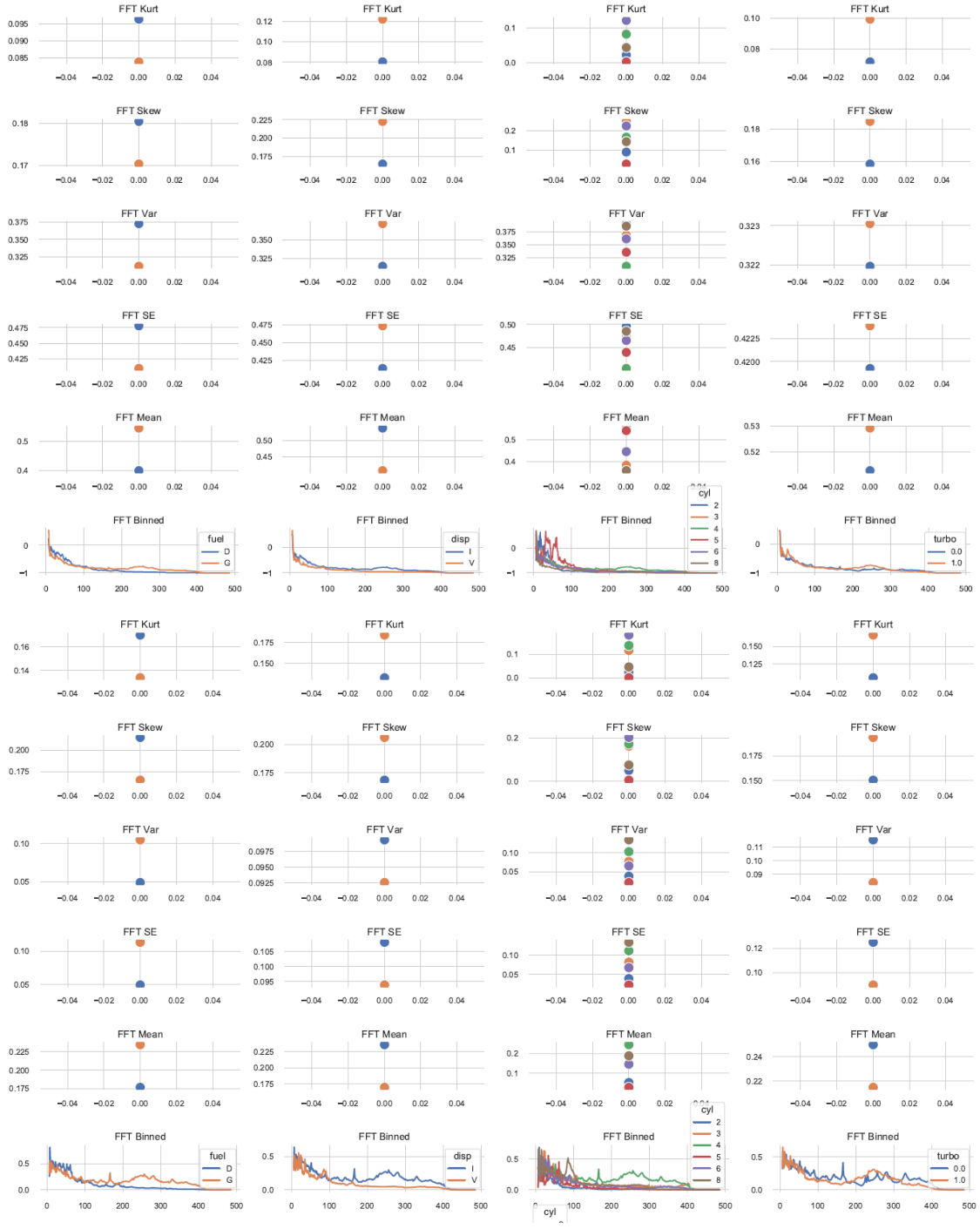


Figure 2.24. FFT-related Features Mean (top) and Standard Deviation (bottom)

"X" MATRIX	Feature 1	Feature 2	...	Feature m
Chunked Sample 1				
Chunked Sample 2				
Chunked Sample 3				
...				
Chunked Sample n-1				
Chunked Sample n				

"y" MATRIX	Label
Chunked Sample 1	*Class C1*
Chunked Sample 2	*Class C1*
Chunked Sample 3	*Class C2*
...	
Chunked Sample n-1	*Class Cx*
Chunked Sample n	*Class Cx*

Figure 2.25. X and y Matrices. y represent the target whereas X represents the input. The size of their vertical axis is the number of samples to train on.

which the frequencies are present but the exact frequencies become difficult to identify. By increasing the window size we can identify frequencies more precisely, but the time at which they happen become less certain. Wavelets came in hand to address this issue.

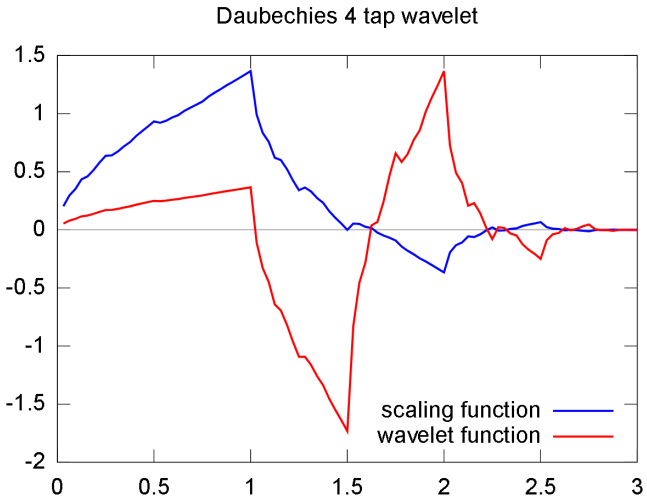


Figure 2.26. Mother Wavelet DB4

Wavelets are mathematical functions that cut up data into different frequency components and then study each component with a resolution matched to its scale. They are more suitable than FFT in analyzing situations where the signal contains discontinuities and sharp spikes. The fact that these wavelets are localized in time gives an advantage over sinusoidal (infinite) waves for what resolution in time domain is concerned. Instead of trying to model the signal with an infinite wave, we are modeling with a finite wave slid across the time domain in a process called convolution. With the Fourier Transform the signal is multiplied with a series of sine-waves with different frequencies. If the peak observed is high, this means

that there is an overlap between those two signals and the selected frequency is observed in that signal. The same process is done with a prototype wavelet (also called mother wavelet, respecting certain properties). The process is repeated with some modifications on the mother wavelet (stretching or squishing) to accommodate lower and higher frequencies. Temporal analysis is performed with a contracted, high-frequency version of the mother wavelet, while frequency analysis is performed with a dilated, low-frequency version of the same wavelet. To summarize, we need a

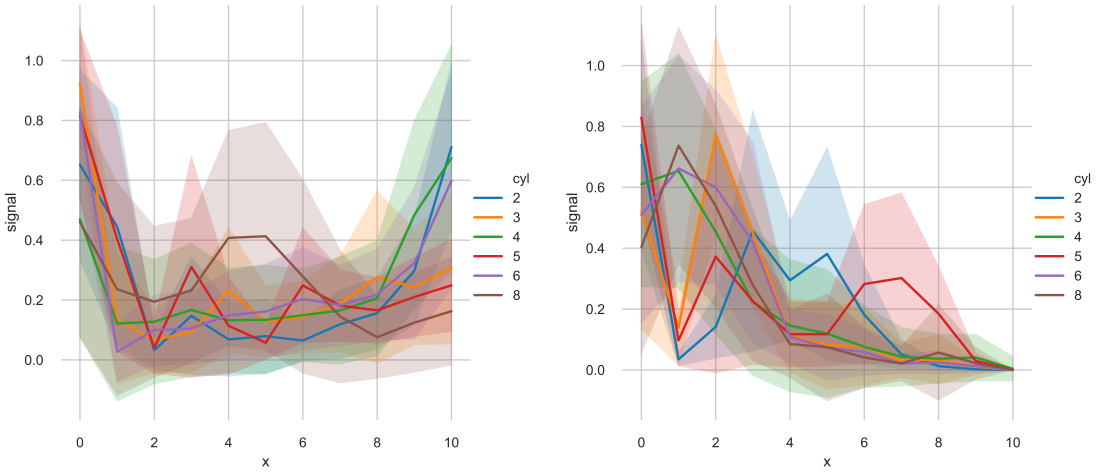


Figure 2.27. DWT Kurtosis and Variance: mean lines are quite different but the variability inside the class is still rather high

bigger time window to catch low frequency and smaller window for higher frequency and that idea is exploited with wavelets analysis. The most commonly used set of discrete wavelet transforms was formulated by the Belgian mathematician Ingrid Daubechies in 1988. This formulation is based on the use of recurrence relations to generate progressively finer discrete samplings of an implicit mother wavelet function; each resolution is twice that of the previous scale [4]. I decided to extract 10 levels of discrete wavelet decomposition with DB4 mother wavelet with the library *PyWavelets* [66]. Then for each decomposed vector separately I extracted some statistics instead of keeping the long array (the same process as for the FFT):

- Skewness: indicating the symmetry of a distribution around its mean value. It is computed from the third moment. A positive value indicates that more energy is distributed on the left with respect to the mean. Opposite if it is negative.

- Kurtosis: giving a measure of the flatness of a distribution around its mean value. It is computed from the fourth moment

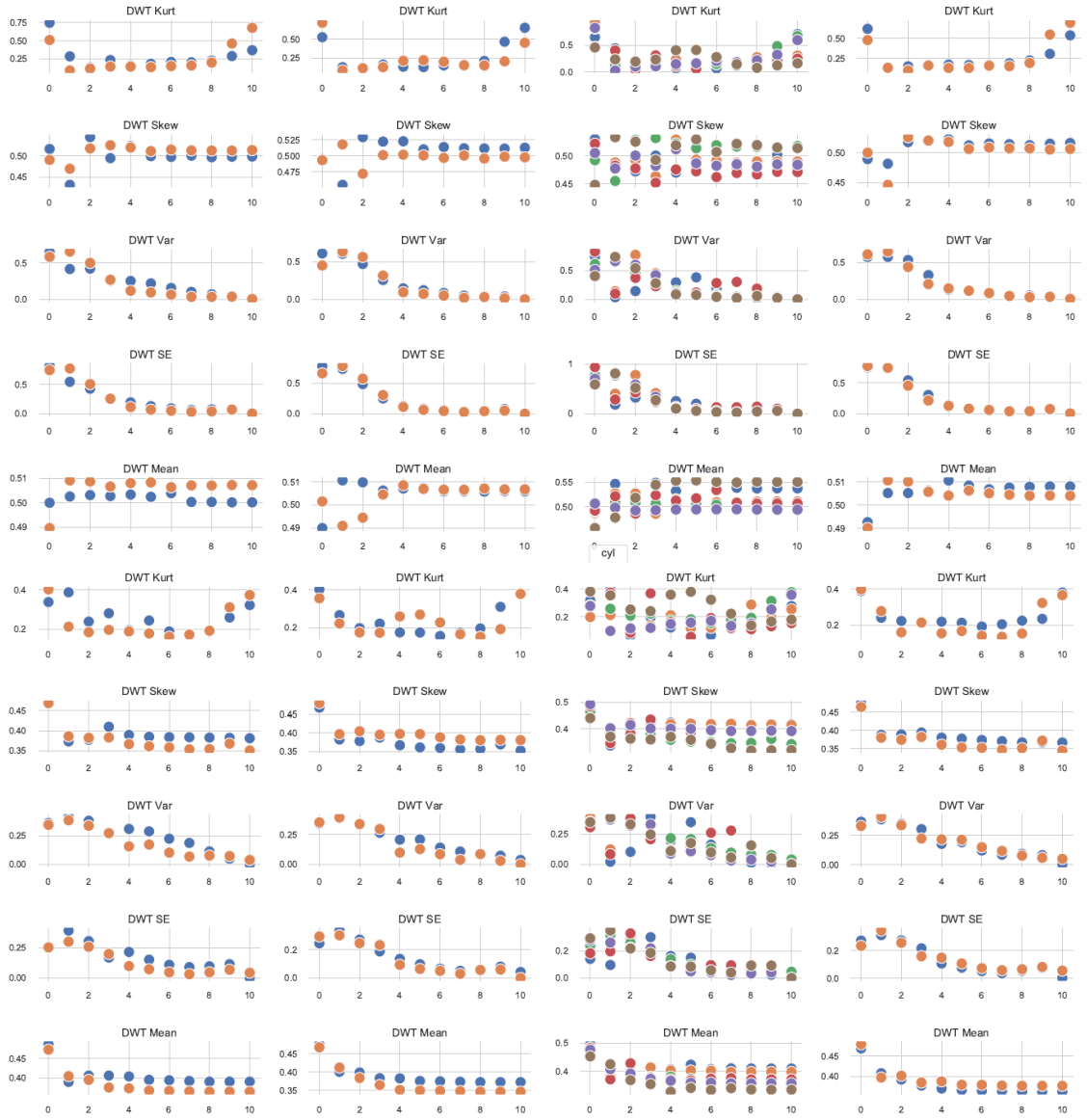


Figure 2.28. DWT-related Features Mean (top) and Standard Deviation (bottom)

- Mean
- Variance

The development of those quantities (their mean and standard deviation) is shown in Figure 2.28 and 2.27, and are indicated in the Excel file the same way as for FFT, as shown in Figure 2.29

AG	AH	AI	AJ	AK	AL	AM	AN	AO	AP	AQ	AR
25	DWT	3.1	DWT Kurt	3.2	DWT Skew	3.3	DWT Var	3.4	DWT SE	3.5	DWT Mean
used	1	used	DWT	used	DWT	used	DWT	used	DWT	used	DWT
function	pywt.waved	function	my_dwt_stat	function	my_dwt_stat	function	my_dwt_stat	function	my_dwt_stat	function	my_dwt_stat
wavelet	db4	stat_fun	scipy.stats	stat_fun	scipy.stats	stat_fun	scipy.stats	stat_fun	scipy.stats	stat_fun	scipy.stats
mode	symmetric										
level	10										
axis	-1										

Figure 2.29. Control File: Discrete Wavelet Transform

Mel Frequency Cepstrum Coefficients (MFCC)

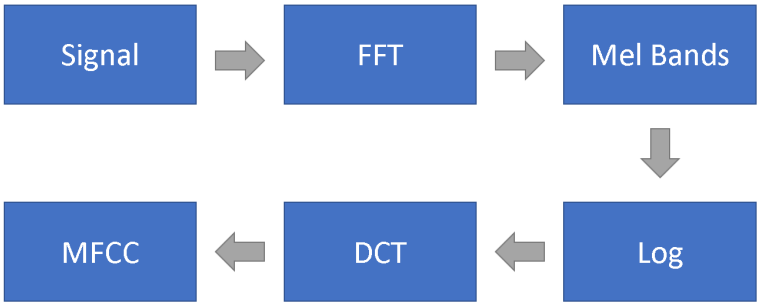


Figure 2.30. How to Compute MFCC

MFCC is a complex transformation made of multiple steps (Figure 2.30), representing the shape of the spectrum with few coefficients. Let's first introduce the Mel frequencies, a set of critical bands filter trying to mimic human ear. Mel frequency scale is linear at low frequencies (until 1000Hz) and logarithmic at high frequencies. Cepstrum is defined as the Fourier Transform (or Discrete Cosine Transform) of the logarithm of the spectrum. MFCC is the vector containing the 12 coefficients of the cepstrum computed on Mel-bands. Since the output of MFCC is a matrix, I decided to average by columns to get a vector. This procedure is acceptable because the signal is considered to be stationary. The mean trend of those vectors is shown in Figures 2.31 and 2.32).

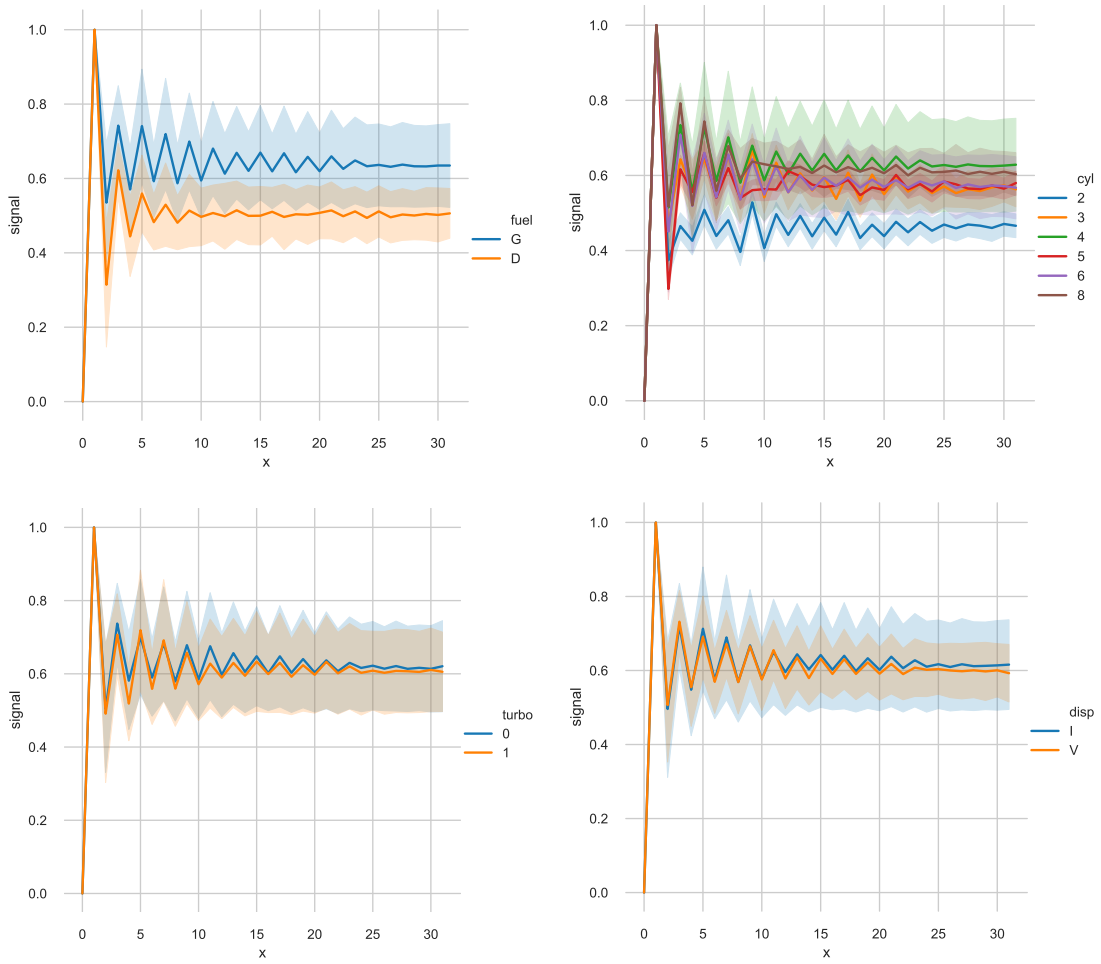


Figure 2.31. MFCC Normalized and averaged by Fuel, Number of Cylinders and Turbo. We can see that MFCC can fairly well differentiate fuel type and some of cylinder amount classes, but not the presence of turbo

Spectral Centroids

Spectral Centroid is the baricenter of the spectrum. It is computed pretending that the spectrum is a distribution, with the probabilities being is the normalized FFT amplitude and the values being the frequencies (figure 2.35) It is computed as the weighted mean of the frequencies present in the signal, determined using a Fourier transform, with their magnitudes as the weights

$$\mu = \frac{\sum_{n=0}^{N-1} f(n)x(n)}{\sum_{n=0}^{N-1} x(n)}$$

where $x(n)$ is the weighted frequency value (amplitude) of bin n, and $f(n)$ the

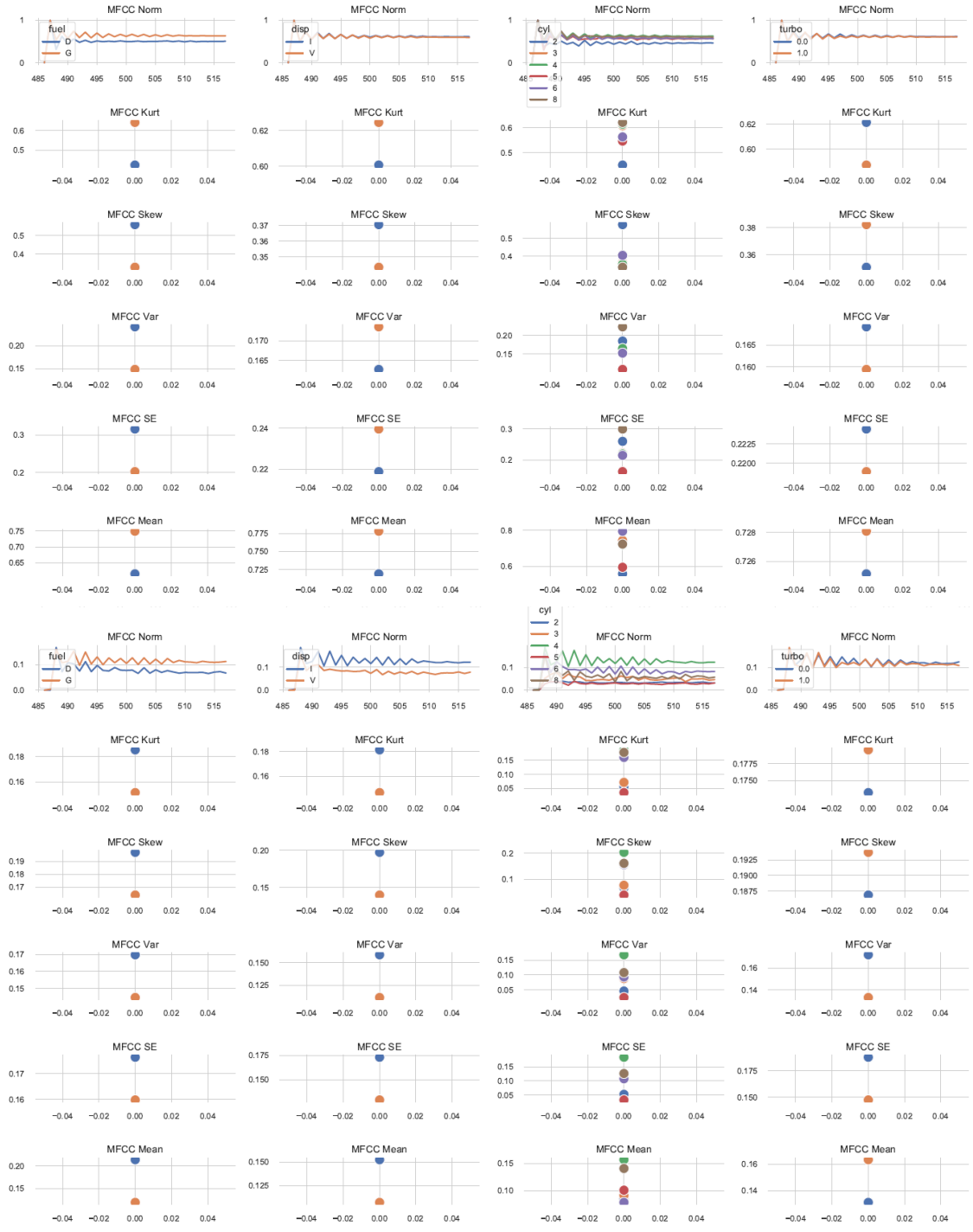


Figure 2.32. MFCC-related Feature Mean (top) and Standard Deviation (bottom)

center frequency of that bin n [84].

Spectral Roll-off

Spectral Roll-off is the frequency value so that 95% of the signal energy is contained below this frequency. It is formulated as

$$\sum_{0}^{f_c} a^2(f) = 0.95 \sum_{0}^{sr/2} a^2(f)$$

where a is the amplitude of the spectrum, f_c is the spectral roll-off frequency and $sr/2$ is the Nyquist frequency (half of the sample rate) [84].The obtained values of spectral roll-off are shown in Figure 2.35.

Zero Crossings

Zero crossing rate is a measure of the number of time the signal value cross the x axis. Periodic sounds tent to have a small value of it, while noisy sounds tend to have a high one. Generally it is computed at each time frame of the signal, but since I already chunked the audio, I used it on the whole length. A plot of this is shown in Figure 2.35 alongside with other features.

Power Spectral Density (PSD)

A Power Spectral Density (PSD) is the measure of signal's power content versus frequency. The amplitude of the PSD is normalized by the spectral resolution employed to digitize the signal. I computed this transformation with the Welch Method. Each word represents an essential component of PSD computation [136, 137]:

- **Power** refers to the fact that the magnitude of the PSD is the mean-square value of the signal. It does not refer to the physical quantity power (Watt), but since power is proportional to the mean-square value of some quantity the mean-square value of any quantity has become known as the power of that quantity.
- **Spectral** refers to the fact that the PSD is a function of frequency. The PSD represents the distribution of a signal over a spectrum of frequencies.
- **Density** refers to the fact that the magnitude of the PSD is normalized to a single hertz bandwidth. For example, with a signal measuring acceleration in unit G, the PSD has units of G^2Hz.

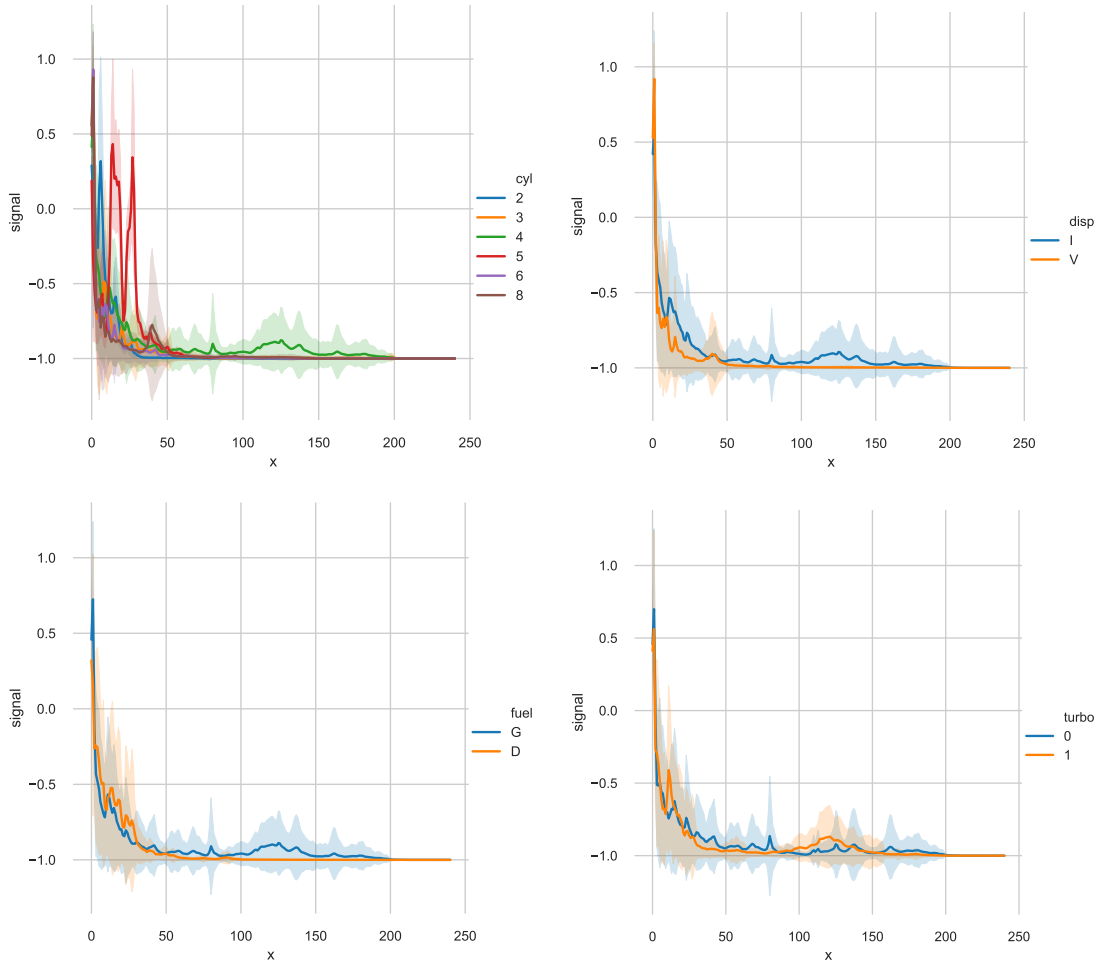


Figure 2.33. Power Spectral Density Trend sorted by class

MFCC Autocorrelated

This feature is extracted following an empirical procedure and was not found in any literature. Instead, I explored some new functions starting from the previously computed MFCC, and applied some sort of autocorrelation and spectral analysis. An interesting aspect of this new feature is that it was selected as "important" by some of our algorithms. It is computed as

$$MFCC_{Autocorr} = Normalization\big(Convolution(MFCC(signal))\big)$$

It is padded with parameters "same" and only the first half of the signal is taken. It is shown in Figure 2.3.1.

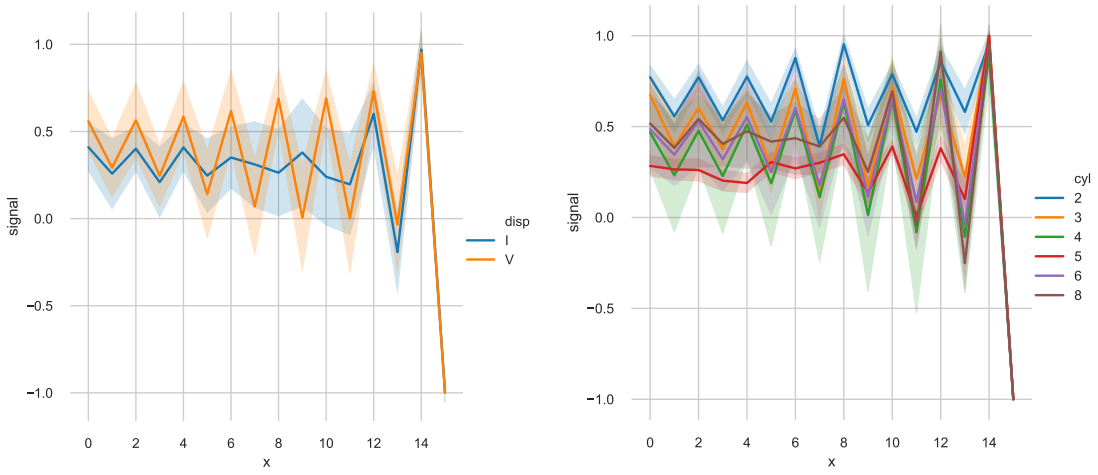


Figure 2.34. MFCC Autocorrelated: Engine disposition (V vs I) and Cylinders amount

Fuel, Location and Turbo

As explained in chapter 1, each predicted label may and should be used for the further prediction of subsequent labels. For example, once successfully predicted the fuel type, that information is used to predict the cylinder amount or the turbocharging. For that reason, I implemented a function to convert label into a feature. If this label was a category or a string (e.g. fuel has "G" or "D"), I converted this string into the integer representing its UNICODE character. Then, after the *MinMaxScaler* function (see Section 2.4), the feature values will be between 0 and 1. Turbo, on the other hand, is already a boolean label, so just a simple function is needed, that takes that label's classes and stacks it to the X matrix.

Others

There are other features we computed, such as Chroma Values, Spectral Bandwidth and Spectral Contrast, but as those are not so relevant for the results, they are just shown in Figure 2.35 without any further explanation. There are features I did not generate by choice, keeping in mind the main side goals explained in Section 1.7.2 the main reason of that choice is that they are very expensive to compute (e.g. Empirical Mode Decomposition) and could not run efficiently on a mobile device. Further development of this work can lead to even better feature selection and bring better results. This framework is perfectly suited for this purpose.

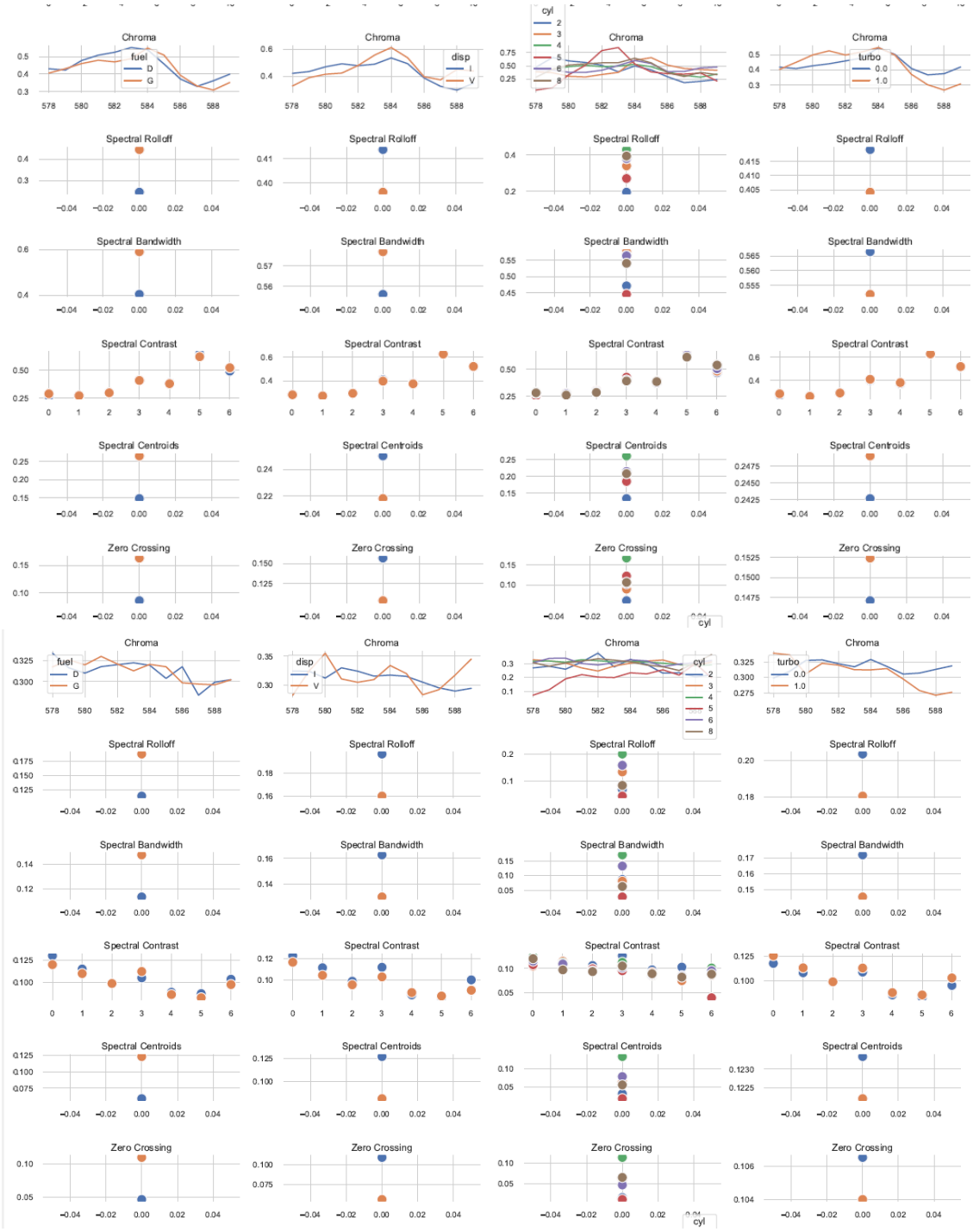


Figure 2.35. Other Features Mean (top) and Standard Deviation (bottom)

2.4 Feature Scaling

When all features are extracted and stored into the X matrix, there is one aspect that is worth taking into account, namely that the values got out from the extraction process have a wide range. Kurtosis values are generally very low, whereas number of zero crossings can be considerably big. Several classification algorithms, as well as dimensionality reduction tools seen in Section 3.2 require the feature values to have similar range, and work better if their distribution is close to normal. For instance many elements used in the objective function of a learning algorithm such as the RBF kernel of Support Vector Machines (section 3.11) assume that all features are centered around 0 and have variance in the same order. If a feature has a variance that is orders of magnitude larger that others, it might dominate the objective function and make the estimator unable to learn from other features correctly as expected. For those reasons, feature scalers from *preprocessing* module of *sklearn* library are used.

2.4.1 Feature Values Scaling

In this paragraph, I first consider the scaling procedure of feature that are monodimensional (only one value per sample, e.g. Kurtosis). There are different approach in this context:

- **Standard Scaler:** it is the most used scaling function. It generally takes the vector of the specified feature (e.g. Kurtosis) of each sample (a column of matrix X) and scale each values according to this formula

$$z_i = \frac{x_i - \mu}{\sigma}$$

  where μ is the feature mean and σ the standard deviation.

- **Robust Scaler:** it scales features using statistics that are robust to outliers. This scaler removes the median and scales the data according to the interquartile range (IQR), which is the range between the 1^{st} quartile (25^{th} quantile) and the 3^{rd} quartile (75^{th} quantile).

- **MinMax Scaler:** if we need more control on our feature values, we might decide to scale them linearly to a given range (e.g. from 0 to 1, called respectively *bottom* and *top* in the following formula). For this purpose, the following formula is used:

$$z_i = \frac{x_i - min(x)}{max(x) - min(x)} \cdot (top - bottom) + bottom$$

- **No Scaling:** This option is still a possibility left open to the user.

- **Custom Scaler:** I left the possibility to the user to define his own scaler and still be able to apply it without hard-coding anything.

In the Excel file it is possible to indicate a list of scalers, which are applied sequentially to the data. For the features not needed, but just used to compute further statistics (e.g. FFT or DWT), there is the possibility for the user to drop it, in order to reduce the dimensionality of the data in a preventive way.

2.4.2 Feature Vector Scaling

The procedure described before works only for monodimensional features. We can also scale the FFT as it was a series of single feature, but we would then lose the relationship among values, and a higher low-frequency values would not be captured. In order to maintain the horizontal relationship among the data, and still scale it in a defined range (e.g. 0 to 1) I implemented a normalization function with the aim of first reducing the amplitudes of the vector elements maintaining the horizontal relations, by keeping tracks of the maximum value of the function and scale only than value with respect to the others. Let's make an example: we extract the Fourier Transform, which is a vector of 500 elements. We scale that vector horizontally between 0 and 1 with the *MinMax Scaler* described above. The relationship between high and low frequencies are kept, but we would lose track of its peak amount. For this reason, I followed a four steps approach:

1. Extract the maximum value within the FFT vector
2. Scale the FFT vector between 0 and 1
3. Add that max value computed in step 1 in front of the vector
4. Scale only that new first element with one of the scalers described in previous Section (as it was a single value) while we keep the rest of the vector as is, due to the fact that it is already scaled in a range (0,1) in step 1 through normalization.

In this way, both goals are reached: scaling and keeping track of the original peak of the FFT. Another idea would be in step 1 to keep track of the norm of the FFT vector instead of its maximum, but it generally leads to similar results.

2.4.3 Resulting Feature Distributions

To put it all together, features distribution sorted by label are shown in the following figures:

- In Figure 2.36 distribution is shown sorted by different fuel type.

- In Figure 2.38 distribution is shown sorted by the number of cylinders.
- In Figure 2.37 distribution is shown sorted by the presence of the turbocharger.

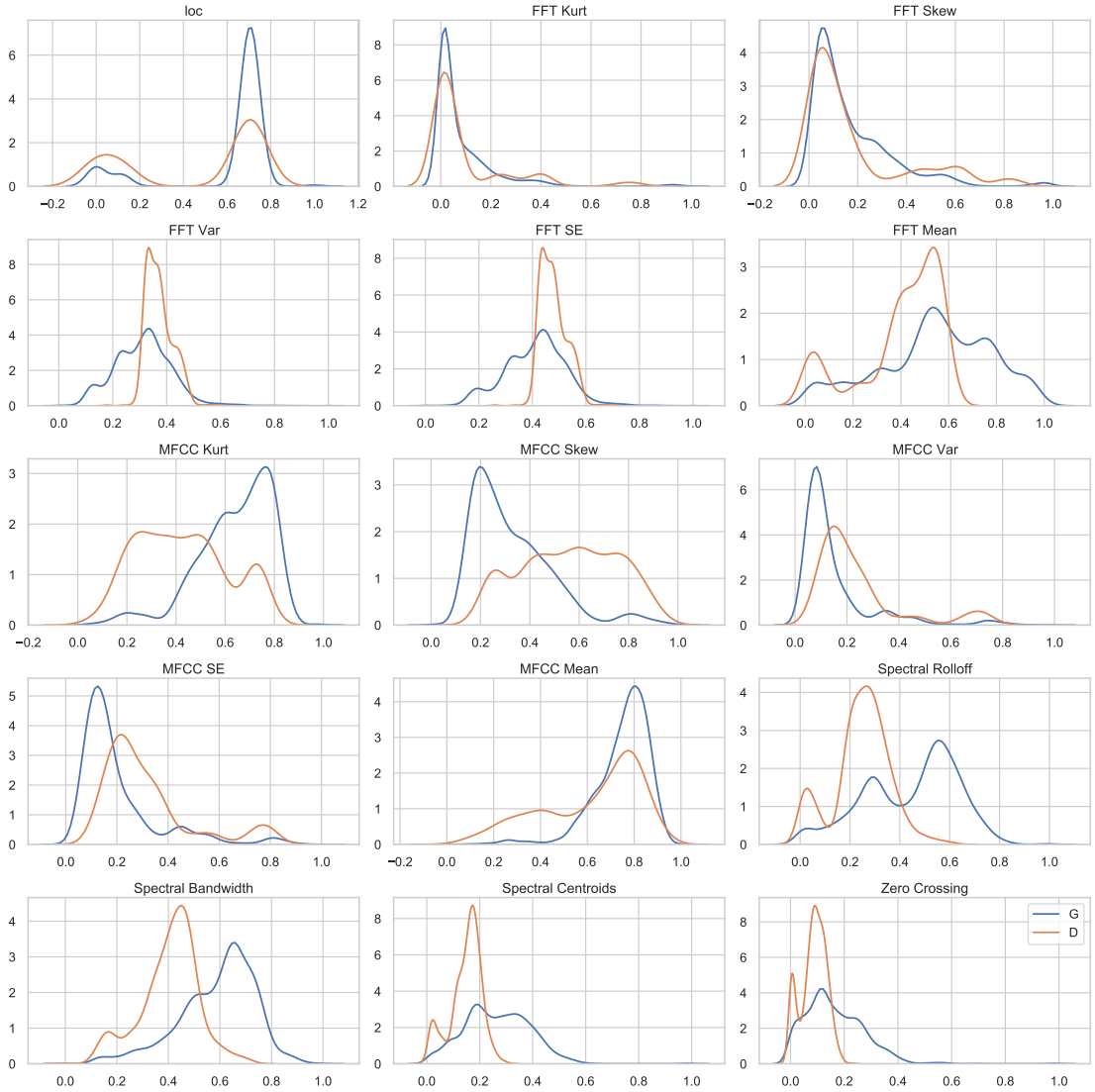


Figure 2.36. Features Distribution by fuel. We have often different distribution, hinting that a good classification is possible. Location distribution is very similar, meaning that this feature is unlikely to bias the results. FFT Mean is one of the most significant features that differs from the two classes. In fact, Diesel engines of this dataset have higher values of the mean of FFT. This intuition is tested and verified in chapter 5

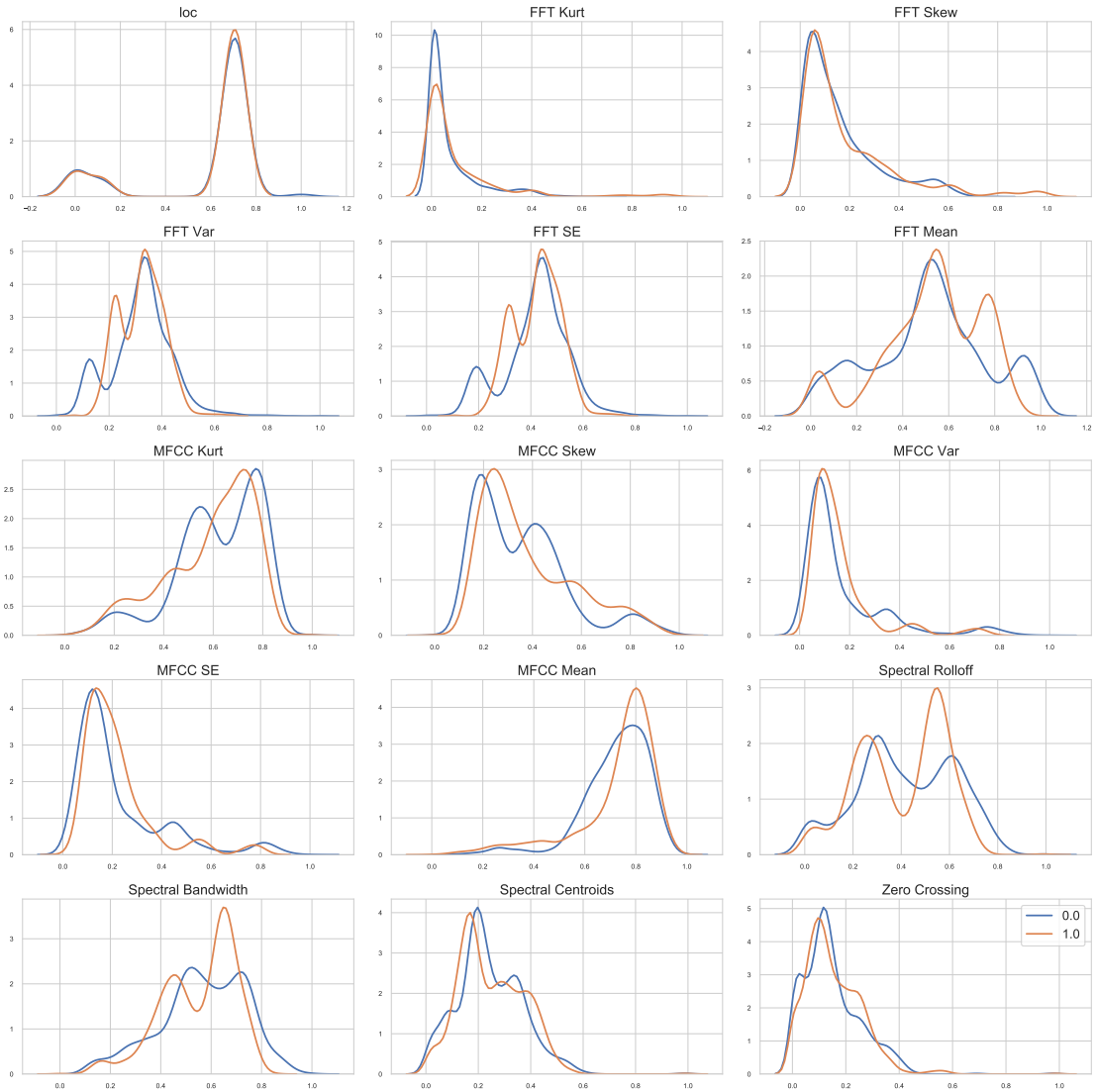


Figure 2.37. Features Distribution by turbo. Distributions are similar and often overlapping, making it difficult separate in a visual manner

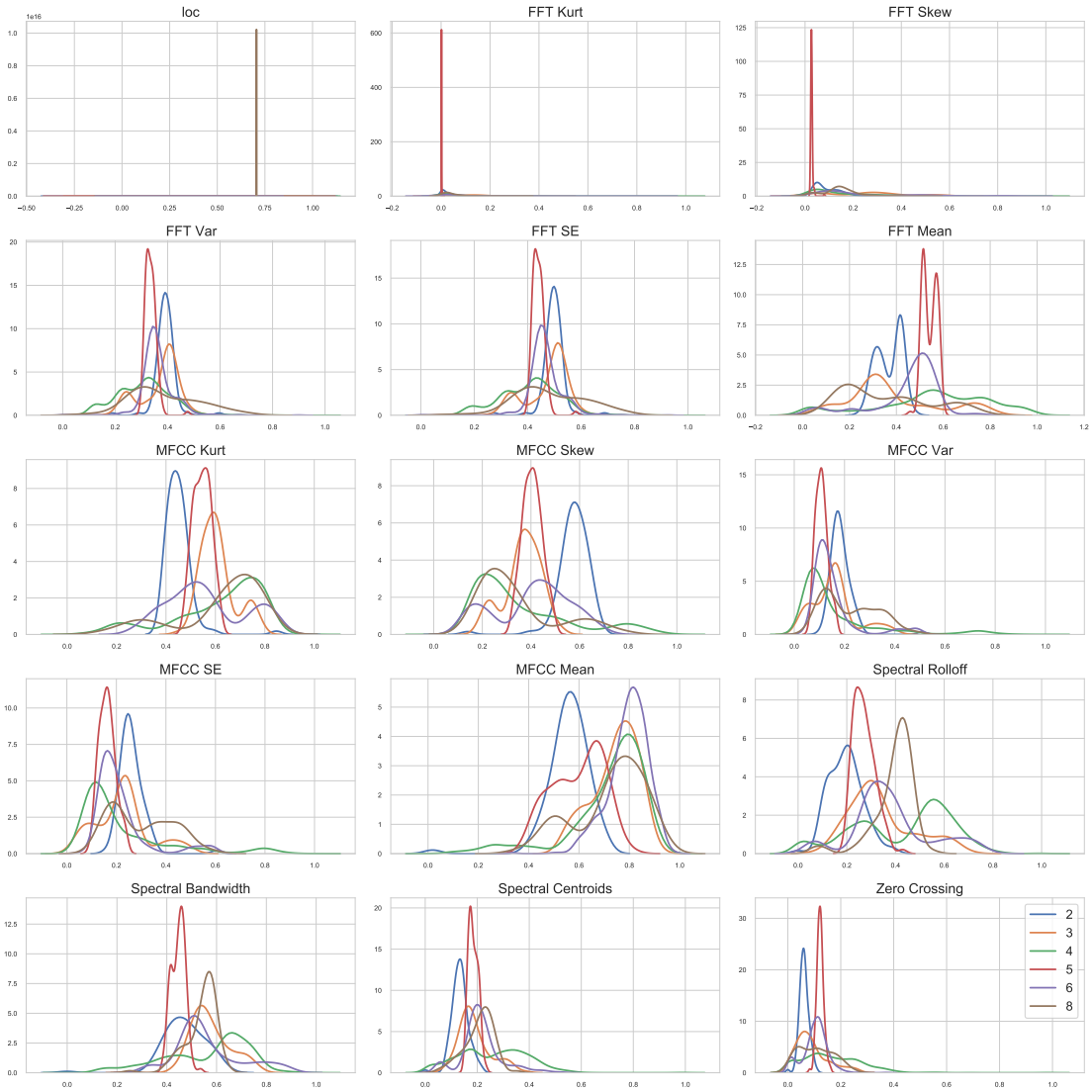


Figure 2.38. Features Distribution by number of cylinders: 5 cylinders class is different from the others, but with few data its distribution is not robust enough. Multiclass classification increases the complexity, and also from a visual perspective is not immediate to separate the classes

2.5 Exploratory Data Analysis (EDA)

The obtained database is too large to be able to look into manually, i.e. exploring each sample independently. This work is left to AI. However, there are some insights that can be gained by combining the samples, their feature and by doing some statistics on it before writing an algorithm. We seek to understand both process physics and the data itself from exploratory data analysis process by plotting several charts:

- Mean feature value or vector per class
- Standard deviation of the feature (or feature vector) within the class. In fact, the mean development of the feature could be pointless if we do not look at its variability, and I addressed this need by plotting the standard deviation of each feature, broken down by class.
- Feature vector plot with deviation band
- Distribution of the features at per-class level

In this section I will provide describe analysis that helped looking into the feature characteristics before passing the "X" and "y" matrices 2.25 to the classifiers. We must pay attention to the fact that if one could not see (as humans) relationships between features and classes, it does not mean that neither an algorithm could do it. Algorithms often are able to capture non-linear relationships among classes. The main features relationship shown are:

1. **Heatmap** showing mean and standard deviation of the features. This enhance relationship visibility and provides a more general view on the features (picture 2.39 and 2.40). Different labels are observed in the different columns and the features in different rows with title. In each figure, vertical axis indicate the different classes of that label (columns) and horizontal axis the different values of that feature vector indicated in the row and in the title.
2. **Correlation** among features. A correlation matrix may provide insights about how features relate to one another, shown and explained in Figure 2.41.
3. **Pairplot** of all features divided by classes provides a more deeper observation (with respect to correlation matrix) of the relationship among the features broken down by label. This plot can be confusing, but it has the of allowing a look into the relationships among features and their distribution. It may be used as advanced technique with respect to the simple correlation matrix. Again I sorted it by Fuel (Picture 2.42), by Cylinder Amount (Picture 2.43), and by Turbo (Picture 2.44).

Unfortunately, in this dataset is too complex to enable this layout to provide good results. Some considerations are however interesting.

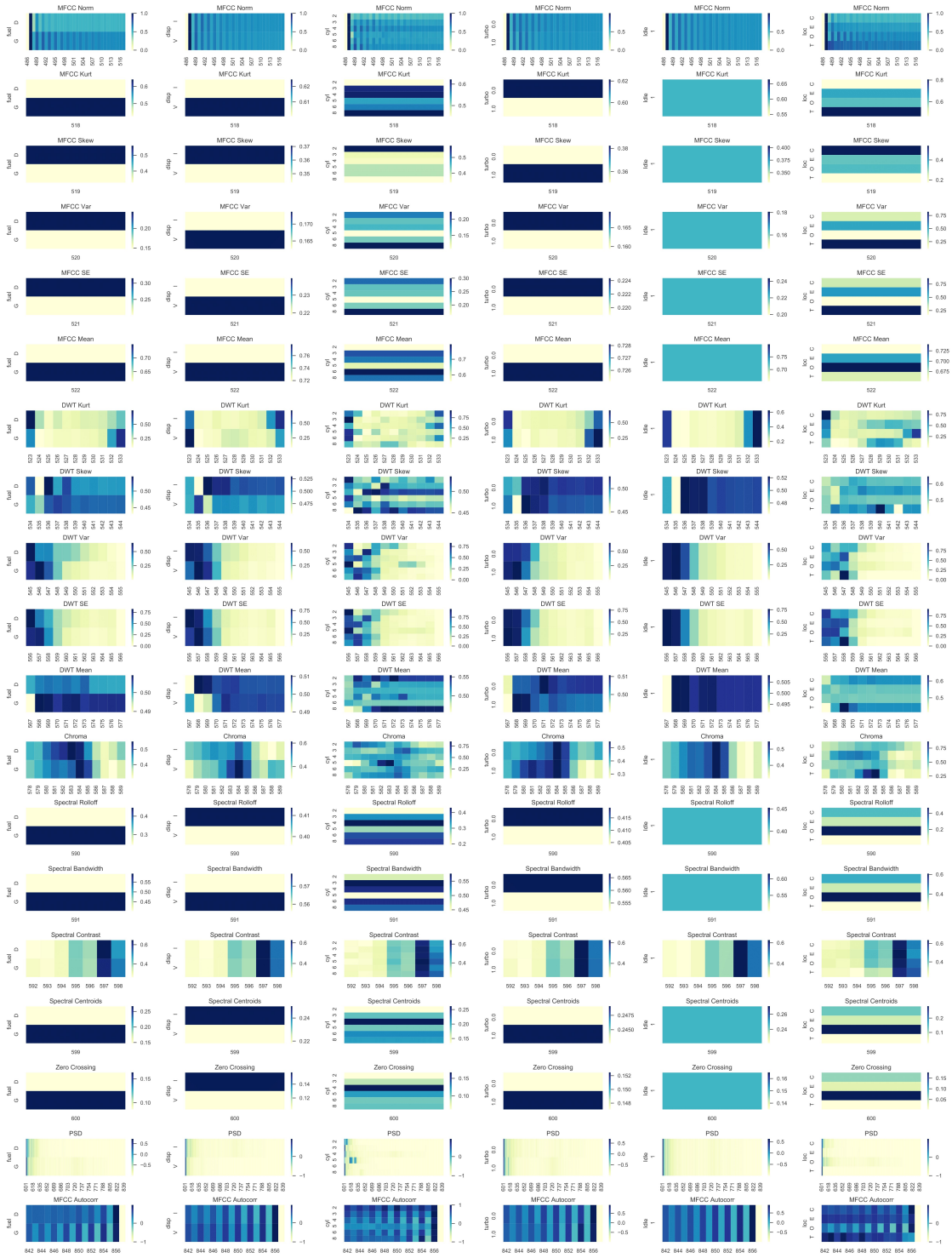


Figure 2.39. Features Mean Heat-map

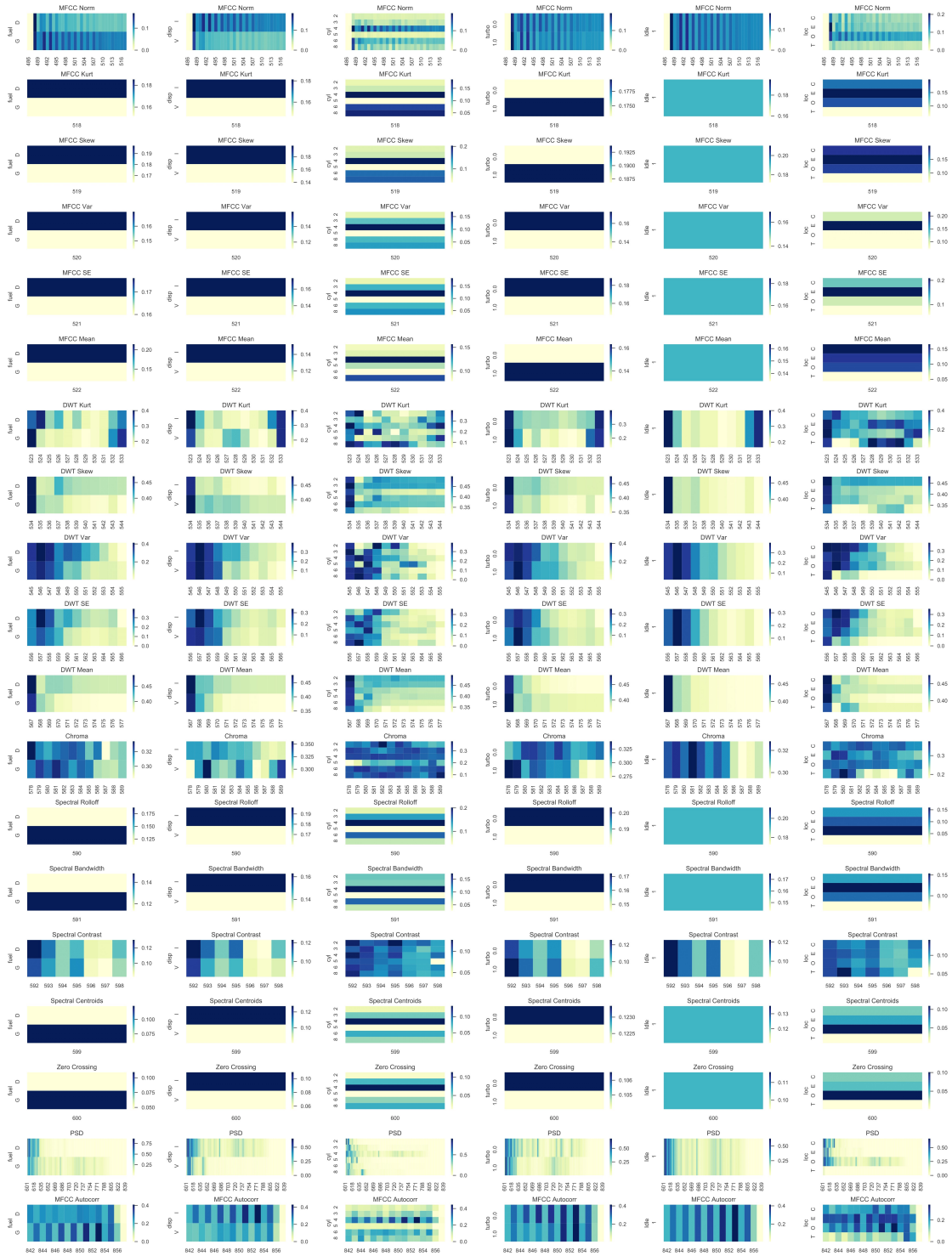


Figure 2.40. Features Standard Deviation Heat-map

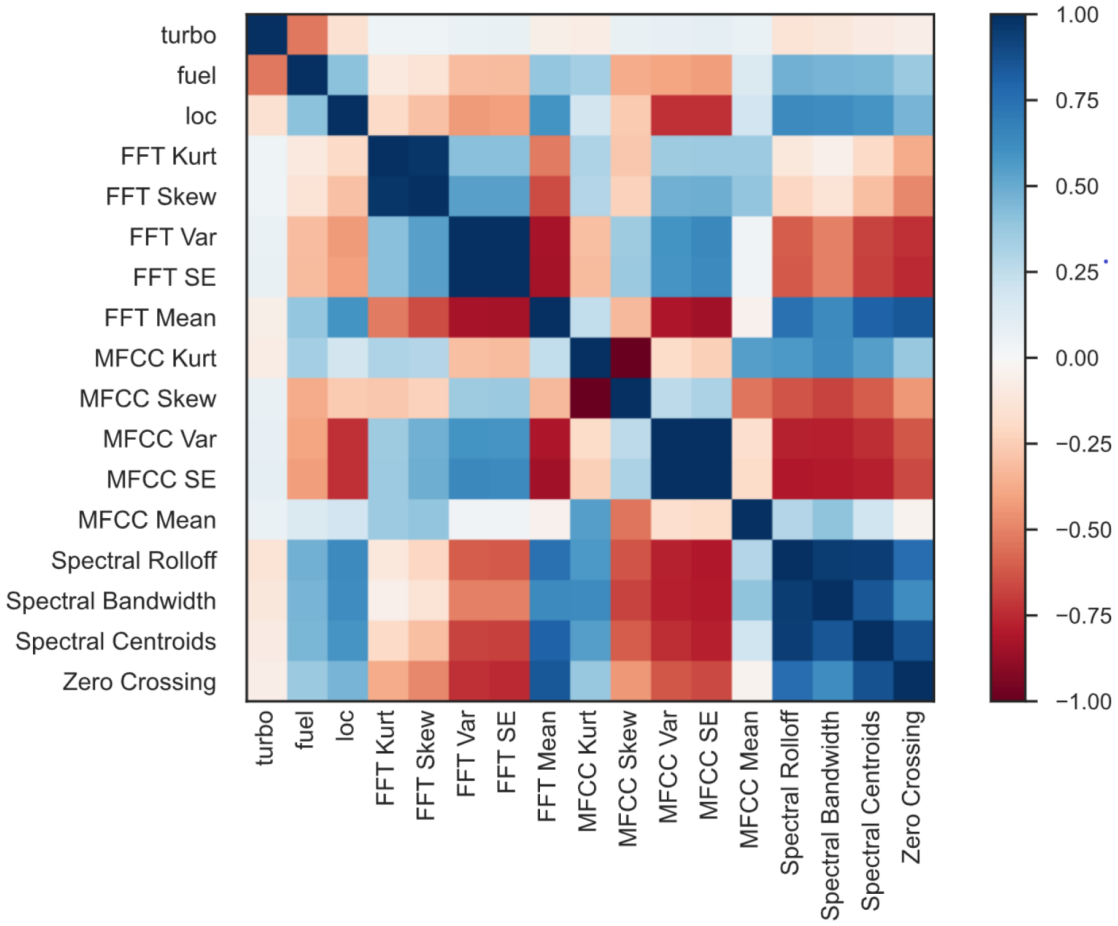


Figure 2.41. In the bottom right corner we see a strong correlation among the spectral analysis features, that are strongly negative correlated with MFCC statistics. This could mean that one of the two features is redundant. Variance and Standard Error (SE) show the same value because they are scaled the same way, so we see a perfect correlation between the two. Also Kurtosis and Skweness are strongly correlated, either positively (FFT) or negatively (MFCC)

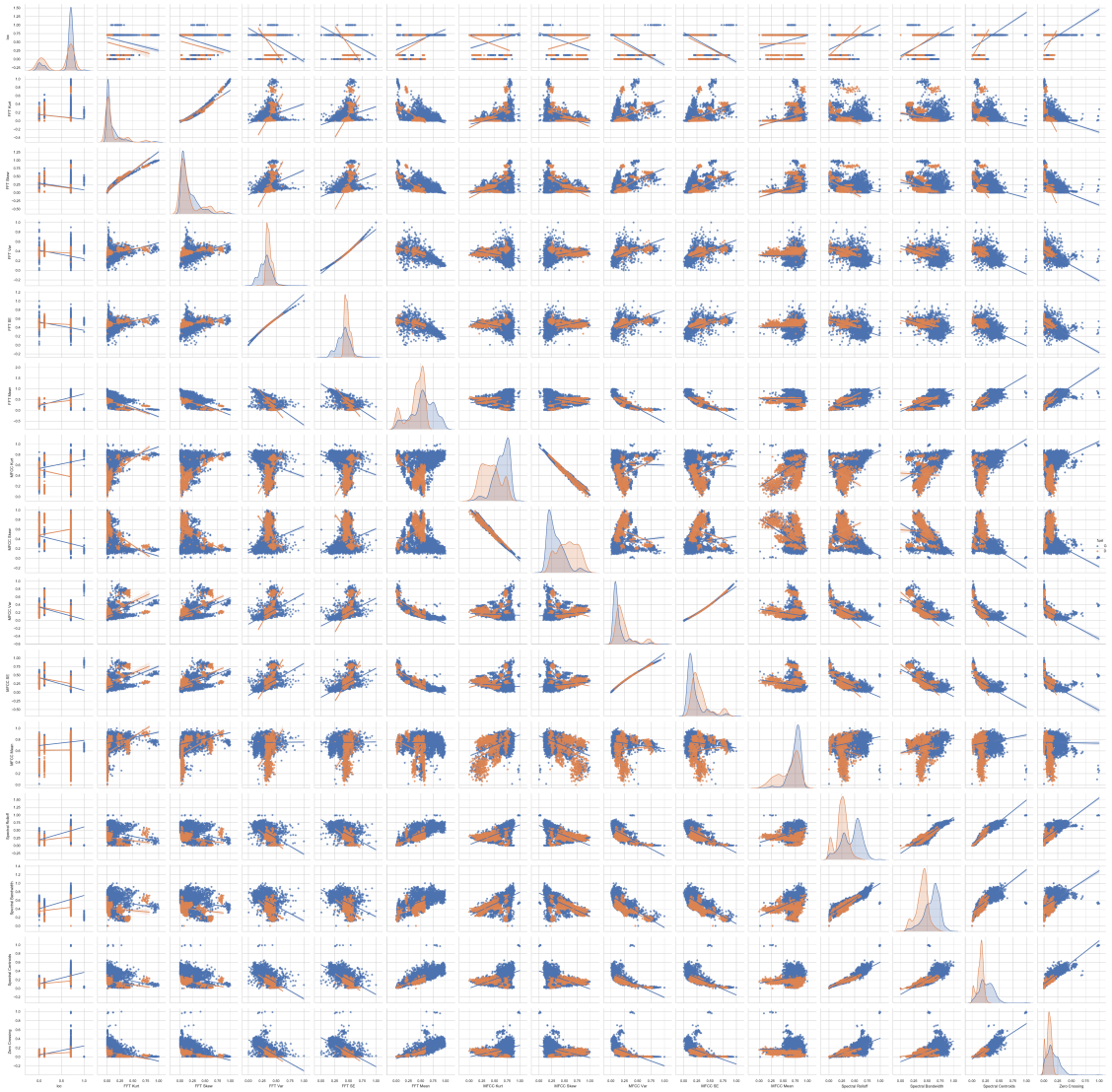


Figure 2.42. Pairplot by Fuel

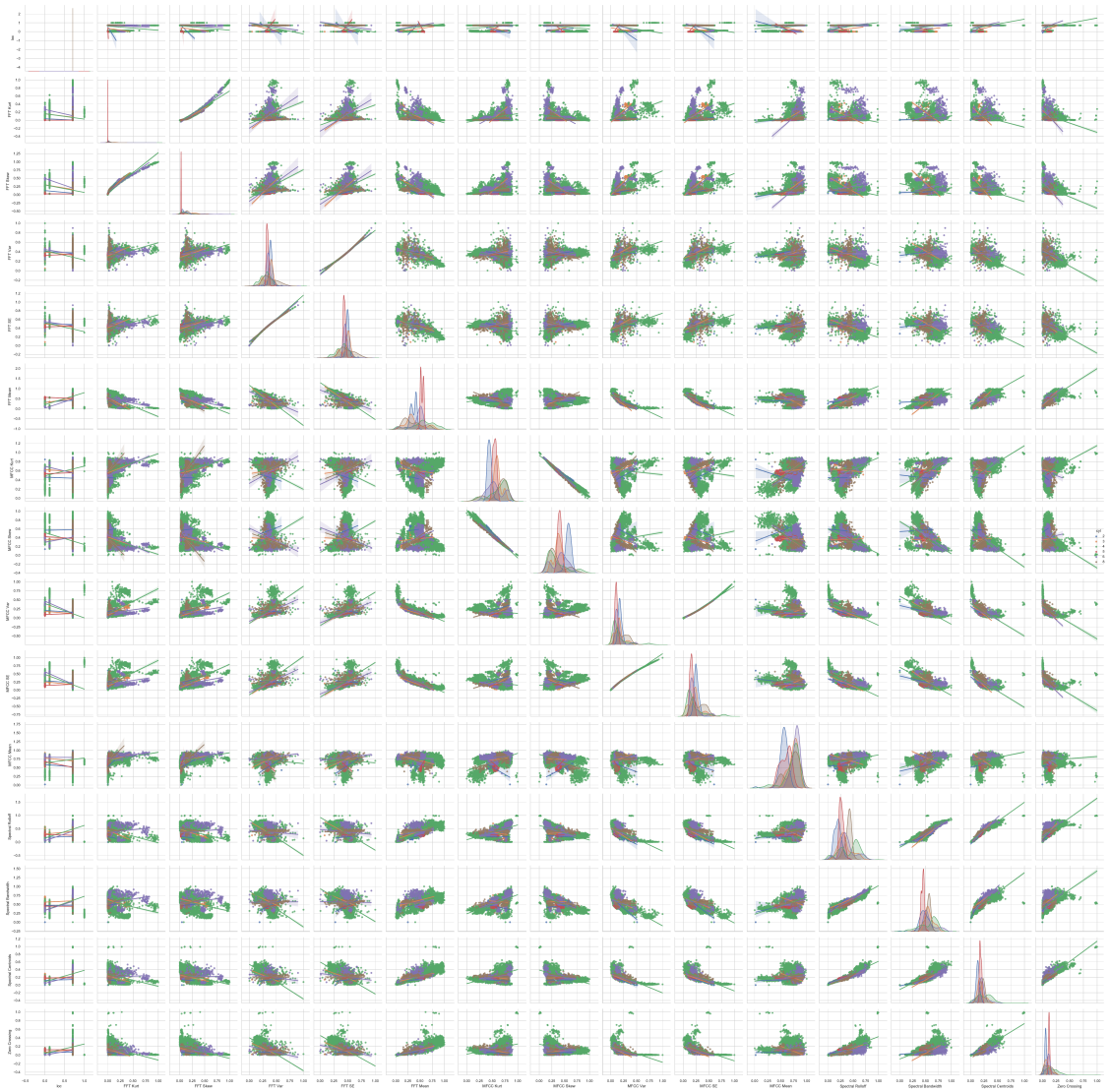


Figure 2.43. Pairplot by Cylinder Amount

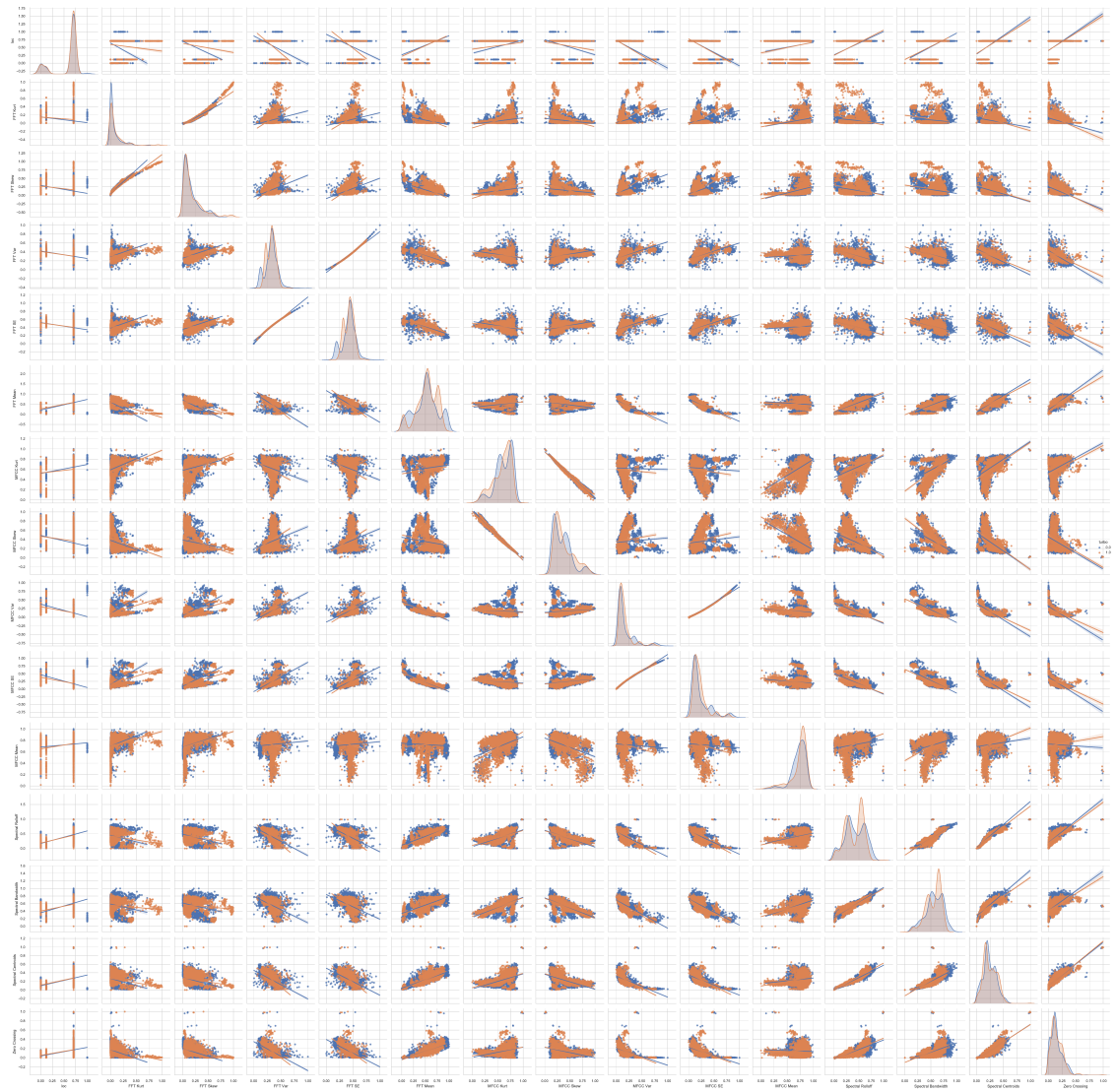


Figure 2.44. Pairplot by Turbo

Chapter 3

Machine Learning Workflow: Algorithms and Metrics

3.1 Cross Validation

As we outlined in Section 2.1.3, overfitting is a challenge when dealing with artificial intelligence algorithms. As a recap, overfitting is when the algorithm perform well on data it trains on (training set), but is not able to generalize its "knowledge" and therefore performs poorly on previously unseen data (test set). Cross validation helps to detect and reduce the effects of overfitting. To understand *cross validation* we first have to introduce the concept of the *validation set*. This is a subset of the *training* set, which is considered as a *test* set during the *training* phase. We save this portion of the training set for validation purposes, fit the algorithm to the remaining part of training set, and test the performance of the algorithm on that previously-unseen validation set. This helps to give an idea of what the test outcome would be without touching it and to change the parameters to improve the performance (Figure 3.1). For this purpose instead of directly using the test set it is wise to use the validation set. If instead we manually tweak the parameters based on the performance on the test set we risk to artificially overfit the problem.

The concept of validation may be expanded through *cross validation*. The idea of cross validation is that there is some chance that the results on the single validation set are not representative. To have a more robust evaluation of the training phase performance, we are repeat the train-validate procedure multiple times on different portions of the sample called *folds* (Figure 3.2). We then collect all performance results by averaging the score of each validation fold.

A final cross validated score is then used to benchmark the set of hyperparameters. It the allows to optimize them and achieve a higher validation score, but still guaranteeing a certain degree of generalization. We can understand the performance by comparing the scores (or errors) computed respectively on the training

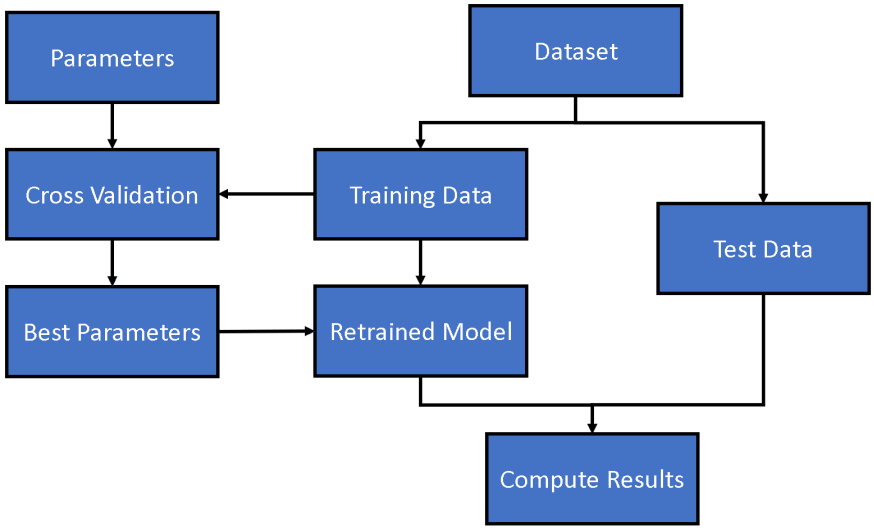


Figure 3.1. How to use cross validation in order to tweak the best algorithm's parameters without the risk overfitting the test sample

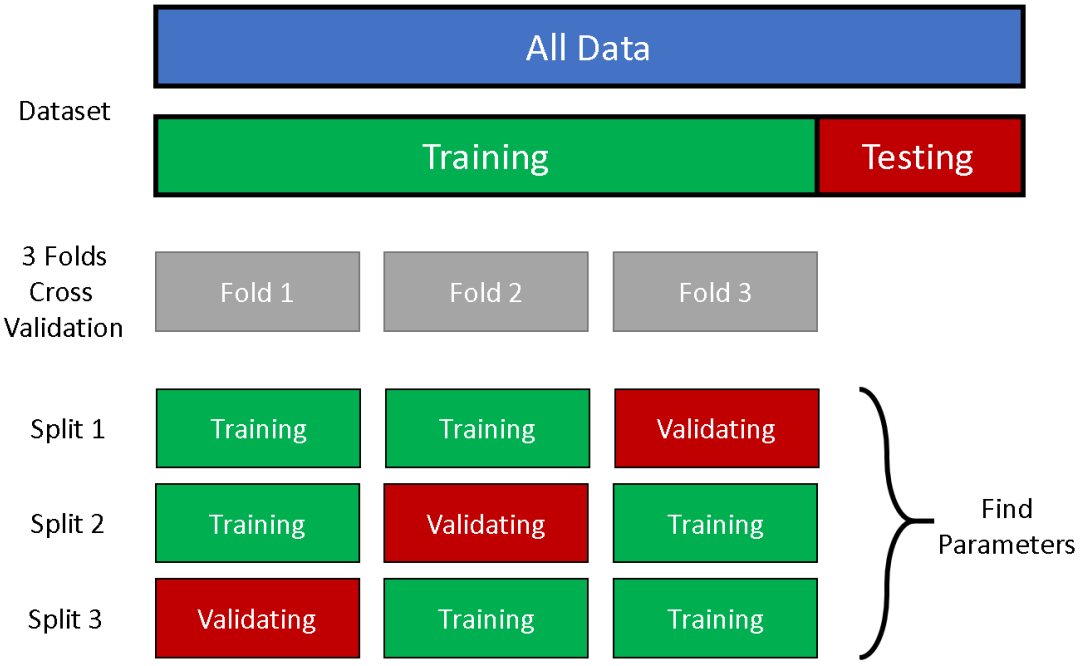


Figure 3.2. Cross Validation 3 folds

and validation set. As shown in Figure 3.3, if the error on the training and validation set are both high, we are experiencing underfitting. On the other hand, if

training error is much lower than validation error, we are experiencing overfitting. The optimal condition is when validation error is only slightly higher than training one, and they are both low. There are cross-validation strategies, and the choice

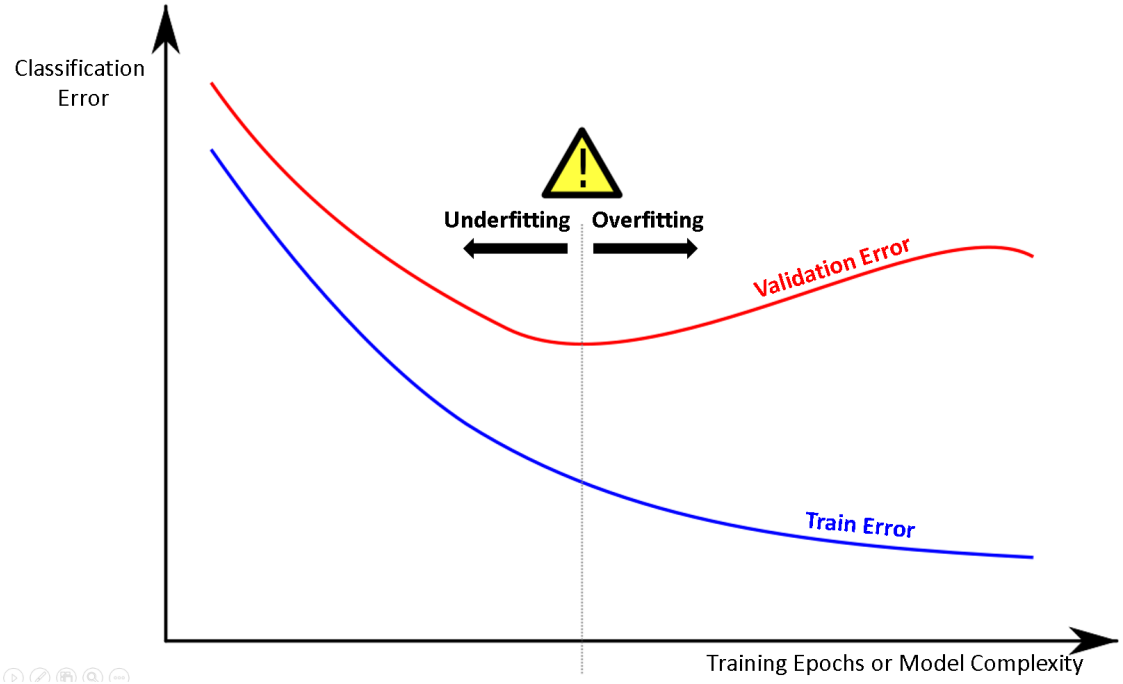


Figure 3.3. Train - Validate Curve and Overfitting. Source: [89]

among them depends on the type of problem we are dealing with. To illustrate those differences, we refer to an example provided by the Python library *sklearn*'s documentation [16], by collecting the plots in Figure 3.4. Common strategies include:

- **Shuffle:** if the data are collected sequentially, it may happen that they contain some trend. If we shuffle the data before splitting train and validate we overcome the problem of unwanted trends intrinsic in the data
- **Stratify:** strategy used to keep the same proportions of classes in train and validate
- **Group:** this possibility is crucial for this use-case, because we want to keep separated different samples belonging to the same ID. In fact, if we have the same sample ID in both splits (train and validate), we encounter the problem of overfitting the validation set (not representative of test set). With group cross validators samples with the same ID belong to only one of the two splits
- **Mix** of the above

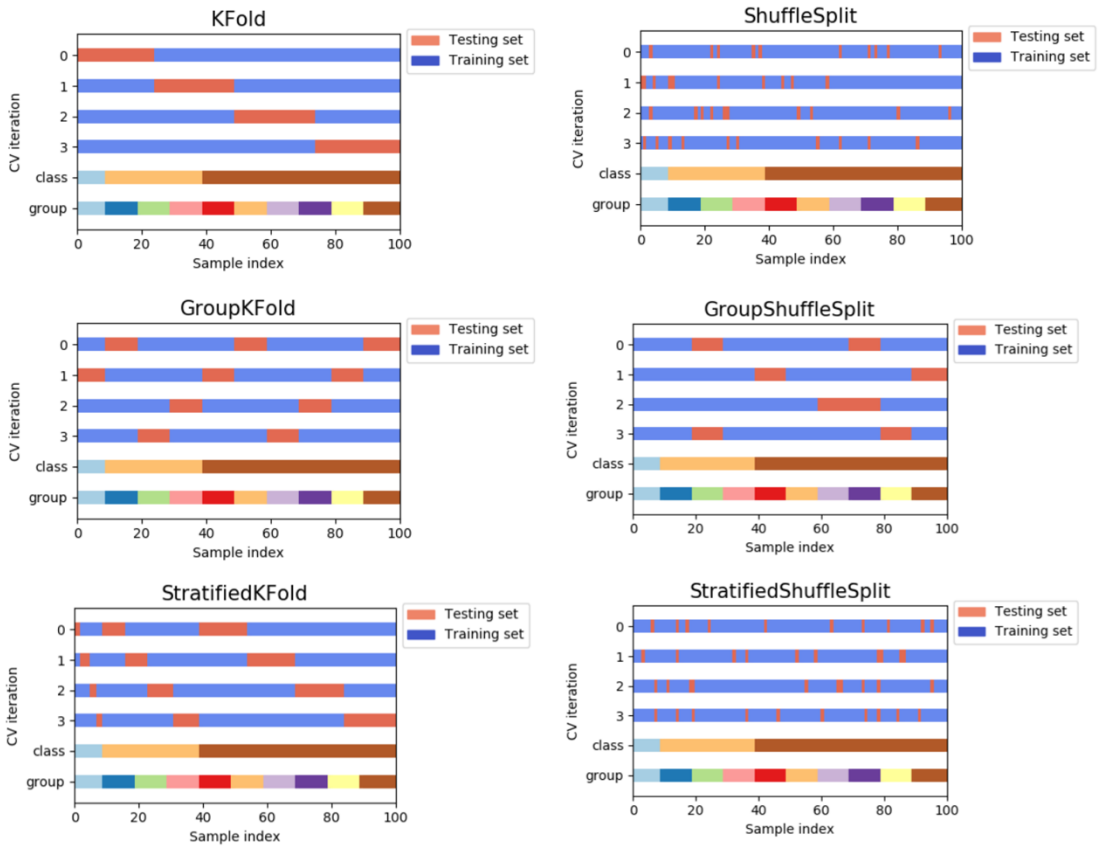


Figure 3.4. Cross Validation Strategies Compared

3.2 Feature Selection and Dimensionality Reduction

Once we extract features, there are several reasons why we may want to reduce that amount by selecting only the most informative ones:

- To avoid extracting them again in the future, when the **computing resources** are important (e.g. application embedded in a smartphone)
- **Speed** up classification algorithm, since the more features we have, the more time it will take to find relationships among those and the class to predict
- Reduce **Overfitting**, since many useless features may "distract" the algorithm, which risks to focus on wrong ones
- **Curse of Dimensionality:** Many classification algorithms are based on the concept of distance between points. If we have 3 features, the distance is

computed on a tri-dimensional plane and it is working fine. But when dimensionality increases a lot, every observation appears equidistant from all the others. If the distances are all approximately equal, then all the observations appear equally alike (as well as equally different), and no meaningful clusters can be formed

For those reasons, we need a step that prepares the matrix X for the classification algorithms by reducing the size of the feature matrix. We use *sklearn's* [16] *Pipeline* function, which concatenates a sequence of estimators and applies cross-validation on a unique object, avoiding the information leakage from training set. It would imply validation scores that may result in misleadingly-high performance results.

3.2.1 Principal Components Analysis (PCA)

We aim to select features with three characteristics [140]:

- **High Variance:** features with a lot of variance contain a lot of potentially useful information.
- **Uncorrelated from each other:** so that the systemic risk of having all the same tendency (and potentially being wrong) is mitigated.
- **Few enough:** they should be few compared to the amount of data we are using, as explained before.

In this regard, Principal Components Analysis (PCA) gives us an ideal set of features, because it creates a set of principal components that are ranked by the variance of the dataset they can explain, they are uncorrelated and reduced down to a specified number. We could specify that amount of values (new features) that we want to be left, or the percentage of variance explained from those remaining features. Those principal components are created by a linear combination of the different features and are orthogonal to each other. The features are then projected to that new coordinate system, resulting in a lower dimensional space explaining a great percentage of the variance of the original one, namely approximating it. An interesting way to show the effects of PCA is to see how the *explained variance* increases by increasing the number of components. In our case, the trend is shown in Figure 3.2.1. We show the difference of PCA behavior depending on the different output labels. As we can see, to reach approximately 80% of the variance it needs around 10 first principal components. Another interesting way to look at it is to check whether we can already spot some clustering from the first two components. We have to remember that PCA is an unsupervised method, so it does not know the classes outcome in advance. We can plot the values of the projection of the features on those two axis, and color the resulting point based on the class. As we can see, it is not easy to already cluster the two classes.

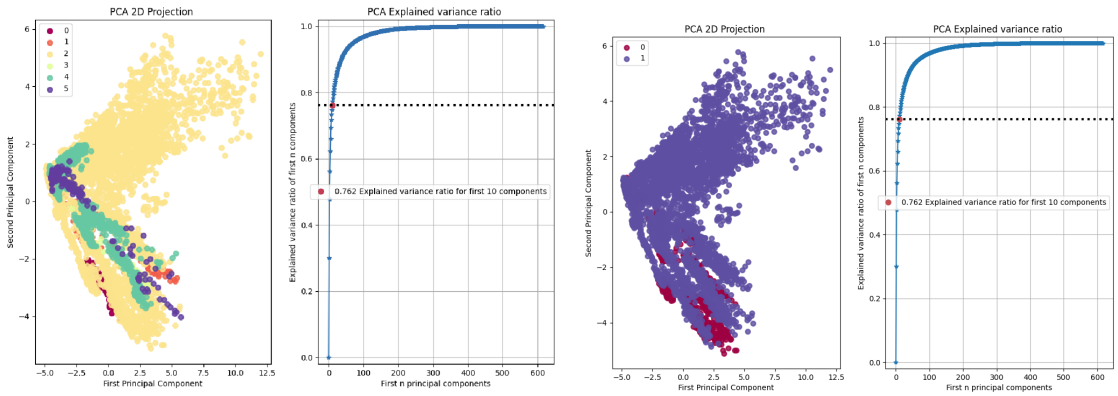


Figure 3.5. PCA explained Variance and 2D projection of first 2 principal components

3.2.2 Kernel Principal Components Analysis (KPCA)

PCA performs a linear transformation on a given data, but many real-world data are not linearly separable. Kernel PCA is a non-linear form of PCA that better exploits the complicated spatial structure of high-dimensional features. It maps features to a higher dimensional space by means of a non-linear function (exploited with Kernel trick), and subsequently applies PCA. Due to its versatility, it is generally more powerful than PCA in my tests.

3.2.3 Linear Discriminant Analysis (LDA)

Linear Discriminant Analysis (LDA) is another tool for dimensionality reduction similar to PCA, but with the key difference that it uses the information from both features and classes (supervised method) to create new axis and projects the data on those axis in order to:

1. **Minimize** the within-class variance.
2. **Maximize** the distance between the means of the two classes (between-class variance).

Since this method is supervised and does not make use of cross validation, we discovered that it did not lead to good results on testing due to overfit.

3.2.4 Univariate Feature Selection

While the previous methods combine features to obtain a lower dimensional space, the following ones score each feature and select the best ones. We can threshold this selection by the top p percent of features, or the best k features. Either ways, only a specified number of the "best" features is selected, whereas the others are

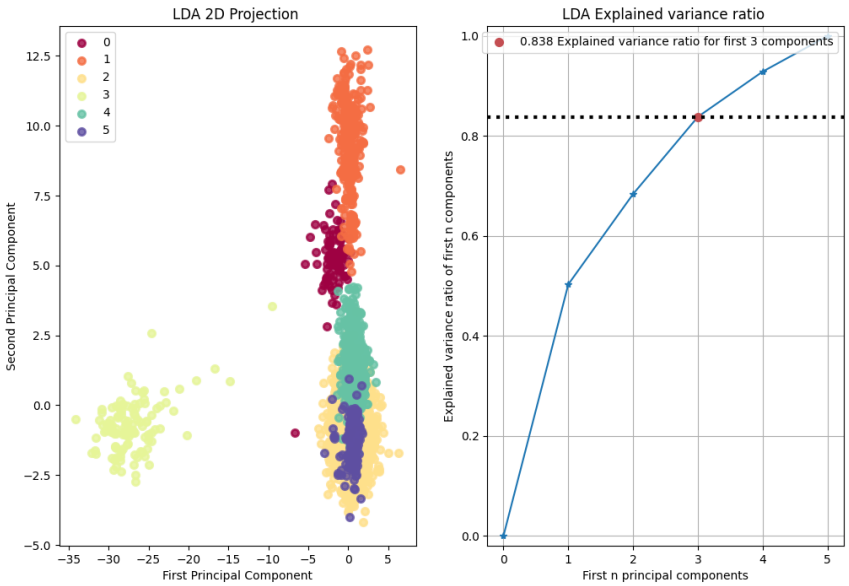


Figure 3.6. LDA explained Variance and projection

discarded because of their low estimated impact on the prediction. The way we used in this project to select the optimal number of features consists of giving a uniform distribution of percentiles to keep (e.g. from 10 to 40 %, called "MixedPercFew", and another run from 40 to 70%, called "MixedPercLot") and the grid search will select the optimal one. For example, as shown in Figure 3.7 on the left, 34% of features were selected, whereas the others were discarded. We can notice that this method of selecting features privileges the feature values belonging to the same feature vector, such as *PSD* or *MFCC Autocorr*. To score the feature importance, Univariate Feature Selection can use different functions, such as Fisher Score, Mutual Information Score or Chi Square.

3.2.5 Tree Based Selection

An interesting method for feature selection makes use on tree-based algorithms and their ability to select the features that they find more relevant for the classification, based on the purity they reach in the leaves (more details in Section 3.3.3). Some examples we used during our work include Random Forest Reducer 3.4.1 and XGBoost Reducer 3.4.4. Some limitations of these reducers include:

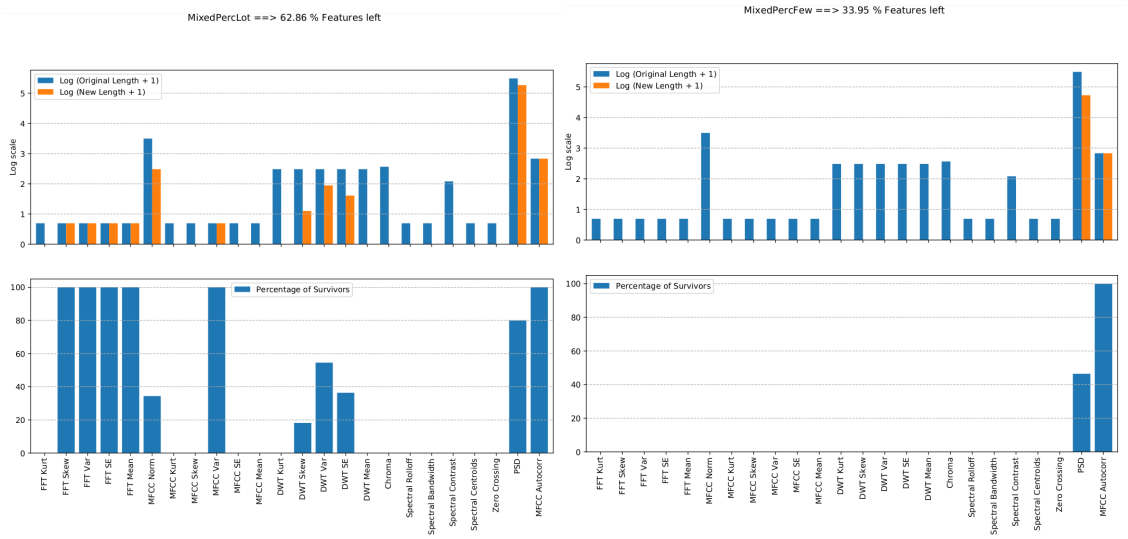


Figure 3.7. Features Left with Univariate Feature Selection

- Correlated features will be given equal or similar importance, but overall reduced importance compared to the same tree built without correlated counterparts. This is why we need to check features correlation, as done in Section 2.5.

- Random Forests and decision trees, in general, give preference to features with high cardinality, and this may serve as motivation for the complex procedure of feature scaling highlighted in Section 2.4 [42].

As we can see in Figure 3.8, tree based methods like XGBoost select different features with respect to univariate feature selection method, and we can see that they are more variable, i.e. not all related to one feature vector (e.g. Power Spectral Density PSD of Figure 3.7), even though less then 9% of them were selected.

To see which features are considered important by Random Forest in all different target labels, see Figure 3.2.5

It may be also interesting to check how and if the tree-based reduction improves the clustering implemented by PCA and LDA, applying those transformers only to the features left after another tree based method (e.g. XGBoost). We can see PCA requires 10 features to get to 80% of explained variance (figure 3.2.1), whereas if we first apply XGBoost as reducer, just the first three components are sufficient for LDA and five for PCA, as shown in Figure 3.10

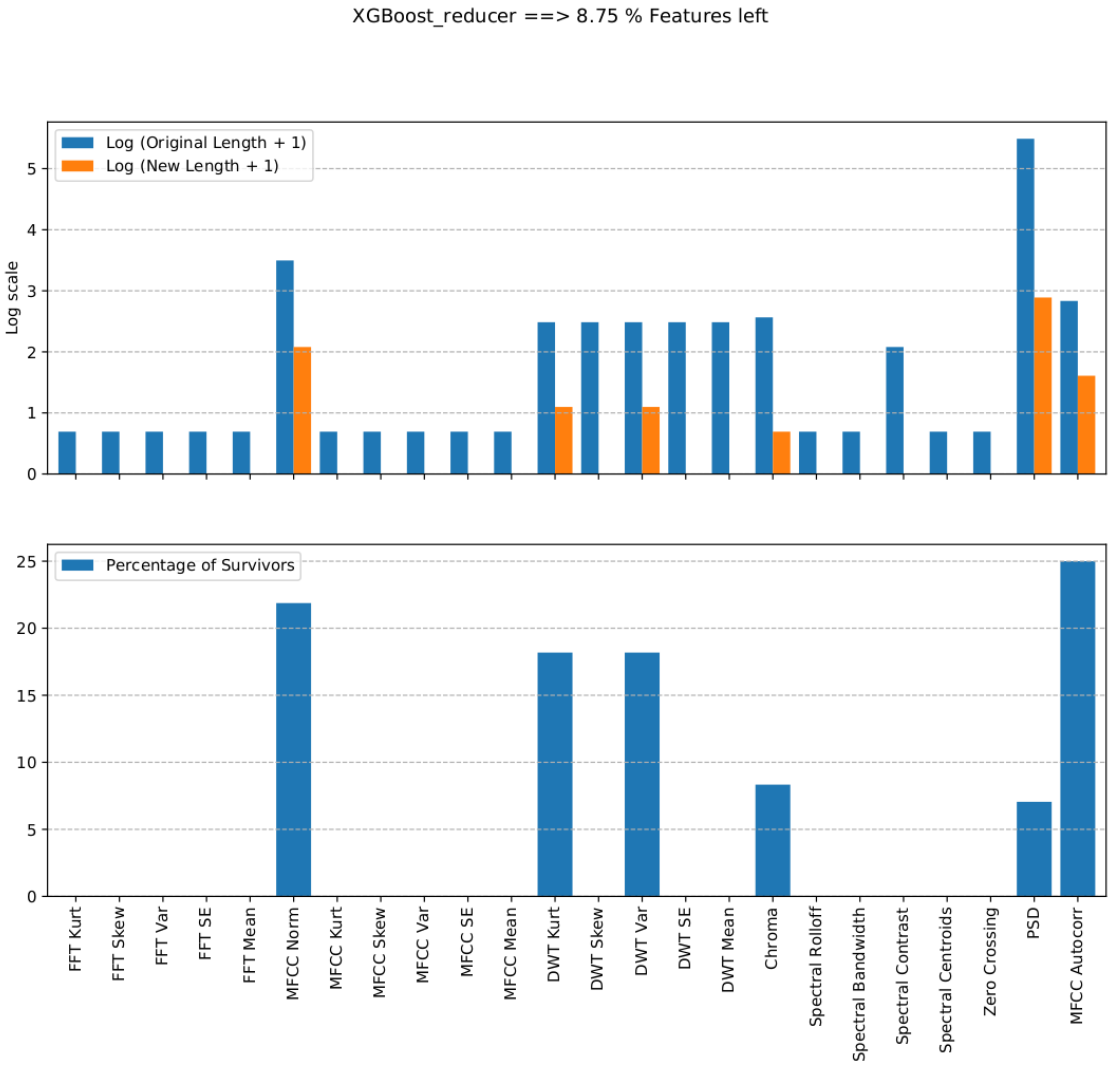


Figure 3.8. Features Left with XGBoost Reducer

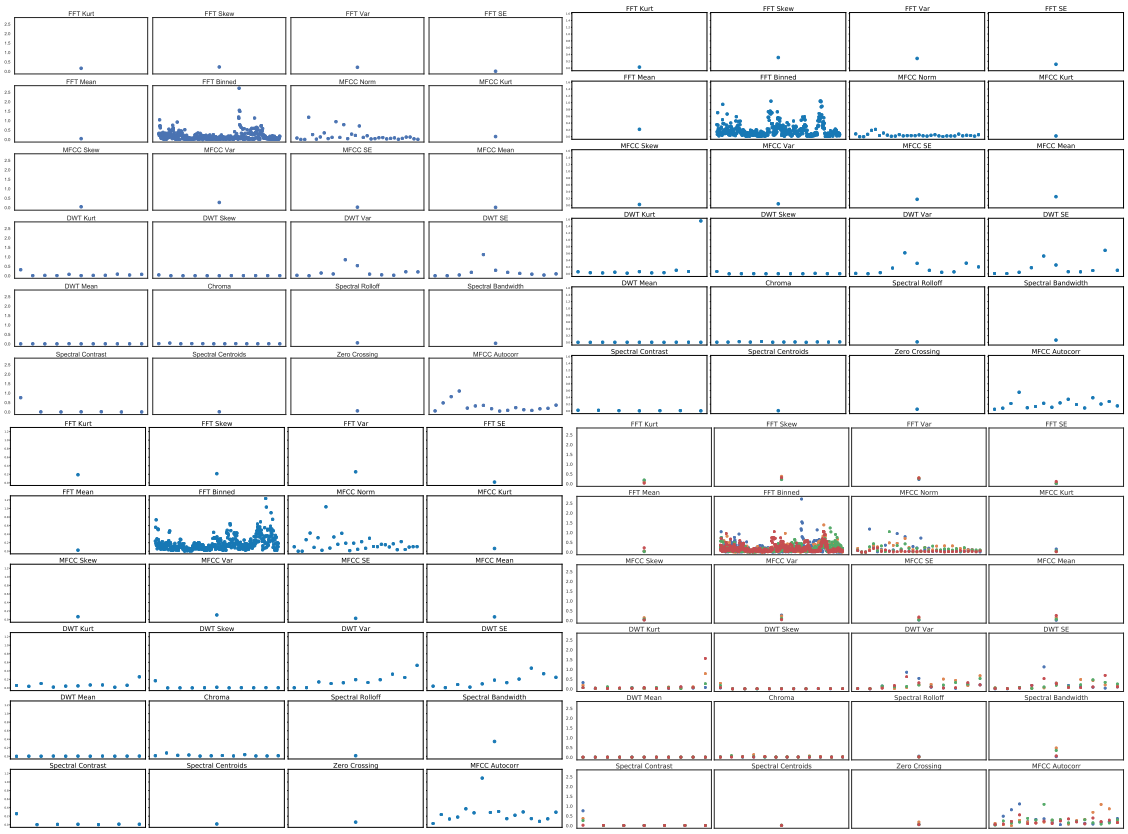


Figure 3.9. Feature Importance with Random Forest for Fuel (top left), Turbo (top right), Cylinder Amount (bottom left), All compared (bottom right). FFT, MFCC and MFCC Autocorr are considered important for all labels

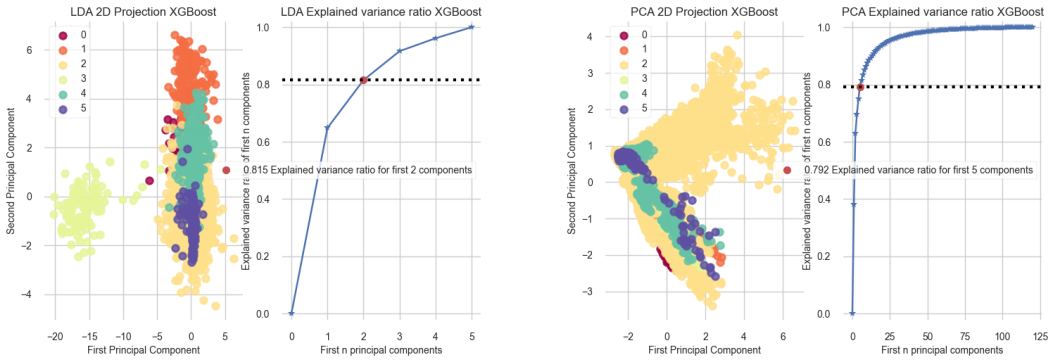


Figure 3.10. PCA and LDA after XGBoost: Clear reduction in number of components to explain 80% of the variance

3.3 Main Algorithms

In this section, we start describing the relevant classification algorithms serving a basis for the ensemble methods explained in section 3.4. We will briefly explain the intuition behind those algorithms, some details on their way of working and the main advantages and disadvantages.

3.3.1 Support Vector Machine (SVM)

A Support Vector Machine is a supervised machine learning model for classification and regression problems. SVM is based on the idea of finding a hyperplane that best-separates the features into distinct regions: it takes the data as input and provides a separation boundary as output. We can imagine the situation in 2D (two features per sample). By plotting each sample on our feature space, we may identify some regions where the different classes are generally located (same as Figure 2.2). Many possible straight lines are available, and SVM chooses the one maximizing the distance from the classes. To compute the distance, the algorithm will select some points for each class and consider them as reference. Those points are called *support vectors*. It then computes the distance between the line and the support vectors: the *margin*, as shown in Figure 3.11. The primary goal is find the optimal separation line (slope and intercept) that maximizes the margin. When the feature dimension is three, instead of a line it will build a plane [121], and since the amount of features is generally higher than that (around 200 features left in my case), a *hyperplane* is generated. In an n-dimensional Euclidean space a hyperplane is a flat, n-1 dimensional subset of that space that divides the space into two disconnected parts. Originally, SVM was only able to draw flat hyperplanes, but today there is the possibility to draw non-linear boundaries between classes thanks to the kernel approach, allowing to enlarge the feature space, similar to the strategy implemented by KPCA (section 3.2.2).

There are three parameters driving the Kernel SVM behavior:

- C: This controls the trade off between smooth decision boundary and classifying training points correctly. A large value of C means that it gets more training points correctly, with a higher risk of overfitting.

- Kernel Function: is the kernel function used for building nonlinear decision boundaries. The mot common kernels are Radial basis function kernel (RBF) also called Gaussian Kernel, characterized by γ

$$K(x_1, x_2) = e^{-\gamma||x_1 - x_2||^2}$$

  and the polynomial kernel characterized by a and b

$$K(x_1, x_2) = (a + x_1^T x_2)^b$$

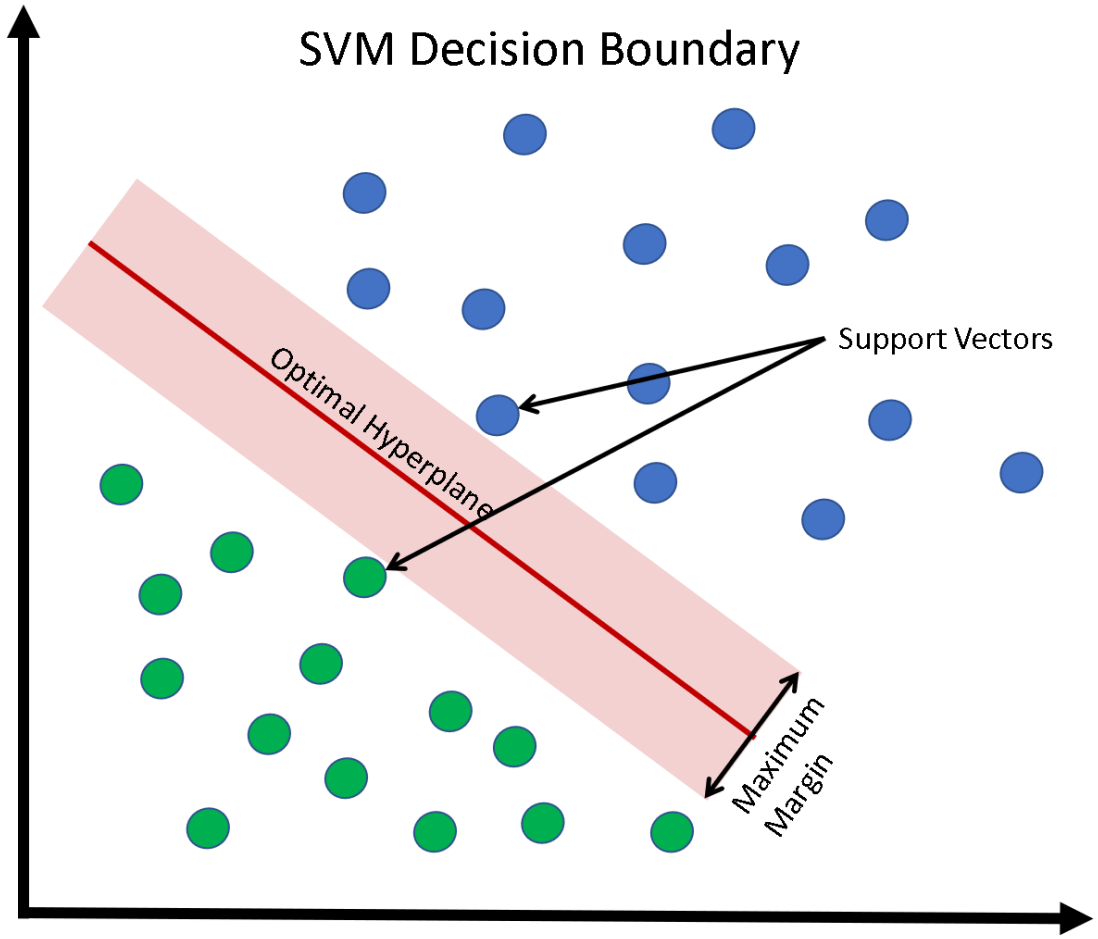


Figure 3.11. Support Vector Machine

- γ : since I primarily used RBF kernels, an important parameter to control is γ. It defines how far the influence of a single training example reaches. If it has a low value it means that every point has a far reach and conversely high value of gamma means that every point has close reach. The higher is γ, the higher is the chance that the model will overfit, because of smaller radius kernels. On the other hand, if the gamma value is low even the far away points get considerable weight and we get a more linear curve.

SVM is a great classifier because it is effective with higher-dimension input data (when there are many features) and because it is influenced only by support vectors, possible outliers in the sample have reduced impact. On the other hand, SVM requires a large amount of time to process large datasets, and performance is heavily influenced by manual parameter selection.

3.3.2 k Nearest Neighbors (kNN)

An idea we can exploit to classify samples is to look where they lie in the feature space compared to the others already present and to infer its class from those neighbors. It follows a natural principle stating that *you are the average of your k closest people*. If they belong to a class (e.g. usual golf players), it is likely that you belong to that class as well. Even though this principle might be questionable, it is intuitive and simple to use. As an example, we consider the feature space of dimension 2 shown in Figure 3.12, so that we can plot the location of each train sample on a 2D plane and mark each point's class by a specific color (green and blue). If a new unknown (red) sample is to be classified, we look at its location in its feature space. Once placed we look at its closest neighbor in terms of euclidean distance to define its class. The idea behind this intuition is that the class of a data point is determined by the data points around it. However, as we can see from Figure 3.12 it may happen that there are some outliers, like the green sample placed in the blue cluster on the right, leading to a mis-classification of the new (red = to-be-classified) one. For this reason, in order to make the model more robust, we introduce the term k, defining the number of neighbors you have to look at to classify a new sample. If in this case we set $k = 5$, we check the class of five closest training points, each one will vote for its class, and the majority defines the class to be assigned to this new point. The parameter k has to be carefully chosen, because it impacts both performance and generalizability. This classification algorithm

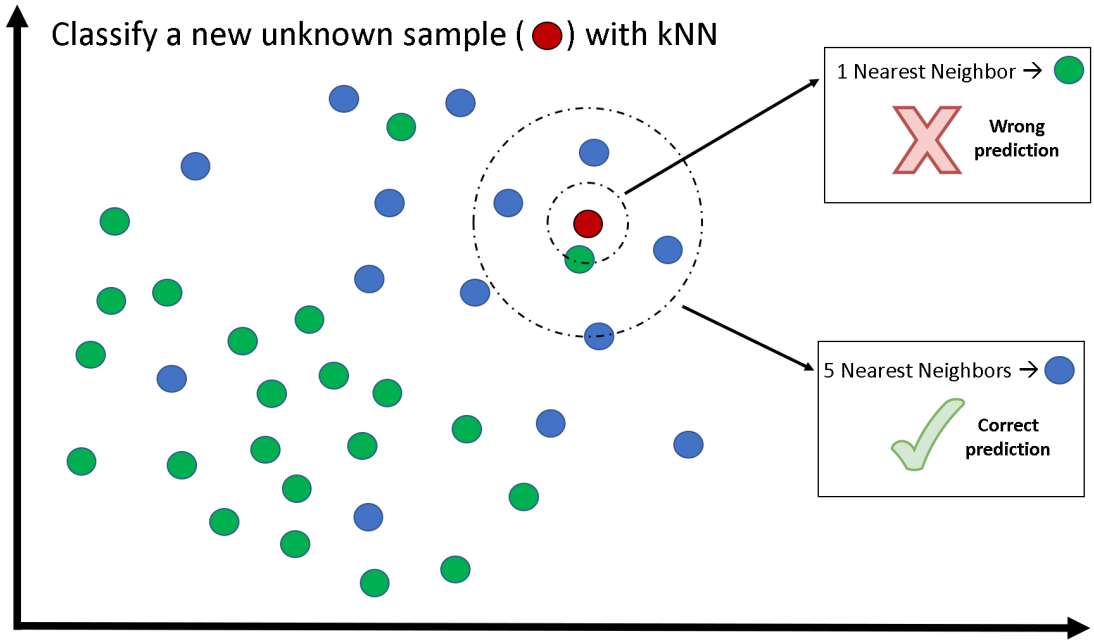


Figure 3.12. k Nearest Neighbors

is easy to implement, it requires short training time, it is suitable for parallel implementation and for classes that are well separated in feature space. On the other hand, it consumes a lot of memory because it has to store each point of the training dataset, and compare each test sample to the whole dataset during the test phase, which results in long computing time, Furthermore it is susceptible to the "curse of dimensionality", (that is, small increases in dimensionality result in rapidly-increasing resource requirements for computation). For this project I needed fast predicting time during testing even with limited device resources, so kNN is unsuitable.

3.3.3 Decision Tree

Decision Tree is a classification and regression algorithm. The name "tree" comes from the fact that it involves splitting the prediction space into a number of simple regions and the set of splitting rules can be summarized in a tree, drawn upside down 3.13. In tree structures, leaves represent class labels and branches represent conjunctions of features that lead to those class labels. The idea behind decision trees is to create a list of subsequent questions to narrow down the options. More formally, the decision tree is the algorithm that partitions the observations into similar data points based on their features. It is built through an iterative process of splitting the data in each node, and then splitting it further on each nodes of the created branches [20, 33]. While single-tree performance is often worse than

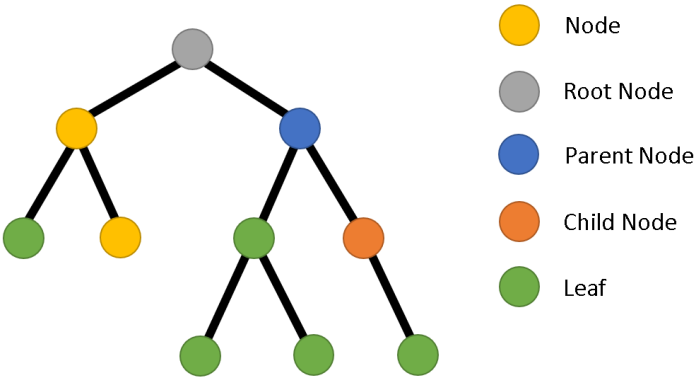


Figure 3.13. Decision Tree Structure

other classification algorithms, this approach is suitable to be used in ensemble methods such as bagging, random forest and boosting thanks to its simplicity (see section 3.4).

Way of splitting

Algorithms for constructing decision trees usually work top-down, by choosing a variable at each step that best splits the set of features. Different algorithms use different metrics for measuring "best". These typically measure the homogeneity of the target variable within the subsets. These metrics are applied to each candidate subset, and the resulting values are combined (e.g., averaged) to provide a measure of the quality of the split.
The most common cost functions for this evaluation are Gini Impurity and Index, Entropy and Information Gain [15]:

- Gini Index is computed in order minimize the feature noise after the split. It is based on the concept of Gini Impurity, which measures how much noise a feature category has. If a feature is a continuous variable, the reasoning is the same as the algorithm split the features in two blocks through a threshold, exactly like categories. Gini Index is computed with the following formula:

$$Gini(K) = \sum_{i \in N} P_{i,K}(1 - P_{i,K})$$

  where N is the list of classes and P is the probability of category K of having class i Gini Index is then computed by weighting the sum of Gini Impurities based on the corresponding fraction of the category in the feature. It combines the category noises together to get the feature noise.

- Information gain maximizes the reduction in uncertainty towards the final decision by minimizing the entropy of the child split. It is based on the concept of Entropy, representing the unpredictability of a random variable, thet is computed as

$$H = \begin{cases} -\sum_{i \in N} P_{i,K} \cdot log_2 \cdot P_{i,E} & \text{if} \quad P_{i,K} \neq 0 \quad \forall i \\ 0 & \text{otherwise} \end{cases}$$

  After obtaining the entropy for each category, we can combine them to get the information gain values for the features. The more information we gain after each split, the better.

$$IG = H(T) - H(T|a) = H(T) - \sum_{i \in K} P_{i,a} \cdot H(i)$$

  where T is the sample space, a is the feature and $H(T|a)$ can be understood as weighted sum of all entropies.

Using the decision algorithm, we start at the tree root and split the data on the feature that results best. Due to this procedure, this algorithm is also known as

a greedy algorithm, as it has the only goal (excessive desire) of maximizing that parameter. In an iterative process, we can then repeat this splitting procedure at each child node until the leaves are pure (only one class). As the data become more complex, the decision tree also expands. If we keep the tree growing until all the training data are classified, our model will be overfit. Setting when to stop is an important parameter and is based on some criteria:

- The number of observations in the node: if we reach that (lower) limit, the algorithm stops.
- The node's purity: The Gini index shows how much noise each feature has for the current dataset and then chooses the minimum noise feature to apply recursion.
- The tree's depth: we can pre-specify the limit of the depth so that the tree will not expand excessively when splitting complex datasets.
- Max features: Since the tree is split by feature, reducing the number of features will result in reducing the size of the tree.

Some advantages Decision Trees classifiers is related to the fact that they are [20]:

- Inexpensive to construct,
- Extremely fast at classifying new testing samples,
- Easy to interpret, understand and visualize for small sized trees,
- Comparable in accuracy to other classification techniques for simple datasets,
- Non-Parametric: they make no assumptions about the space distribution,
- Able to exclude unimportant features and give importance to the remaining ones,
- Able to handle both numerical and categorical data,
- Able to capture nonlinear relationships.

Some disadvantages include:

- Tendency to overfit,
- Decision boundary restricted to being parallel to features axes,
- Often biased toward splits on features having a large number of levels,
- Small changes in the training data can result in large changes to decision logic,
- Large trees can be difficult to interpret.

3.3.4 Passive Aggressive Classifier

Passive-Aggressive is one of the few "online learning" algorithms and may be very efficient for large datasets. In online machine learning algorithms, the input data comes to the learner in sequential order and the model is updated step-by-step. Normal algorithms are referred as "batch learning", where the entire training dataset is used at once. Online learning is very useful in situations where there is a such large amount of data that it is computationally impossible to train on the entire dataset at once. We can simply say that an online learning algorithm will get a training example, update the classifier, and then forget that sample [83].

Even though the working principles of this algorithm are rather complex and sophisticated, we can have an intuition of it by the words composing its name:

- Passive: If the prediction is correct, keep the model and do not make any changes.
- Aggressive: If the prediction is incorrect, make changes to the model.

Some important parameters are:

- **C** is the regularization parameter, and regulates the penalization of an incorrect prediction
- **Tolerance** is the criterion by which the model will stop, i.e. when $loss > previous_loss – tolerance$.

3.4 Ensemble Learning

Ensemble methods use multiple combined individual learning algorithms to obtain better predictive performance than could be obtained from any of the constituent learning algorithms alone. The fundamental concept behind Ensemble Algorithms is a simple but powerful one: the wisdom of the crowd.

Wisdom of the Crowd

Wisdom of crowds is the idea that large groups of people are collectively smarter than individual experts when it comes to problem-solving, decision making, innovating and predicting. For crowds to be wise, they must be characterized by a diversity of opinion and each person's opinion should be independent of those around him or her. For example, by averaging together the individual guesses of a large group about the weight of an object, the answer may be more accurate than the guesses of experts most familiar with that object, and probably more accurate than the single-best guess, as described by the statistician Francis Galton in 1906, when he observed the guesses of 800 people participated in a contest to estimate

the weight of a slaughtered and dressed ox [44] The concept can be expanded to the Machine Learning world by combining individual weak models, each of which performs poorly by themselves, either because they have a high bias (low degree of freedom models, for example) or because they have too much variance to be robust (high degree of freedom models, for example).

Bias and Variance

The ensemble model tends to be more flexible (less bias) but are mostly used to reduce variance of the model, to be less data-sensitive and less overfitting. Bias and variance are in fact strictly correlated with overfitting and underfitting, and a trade-off is often needed (figure 3.14).

There are two main tools to apply Ensemble Learning: Bagging and Boosting [89]:

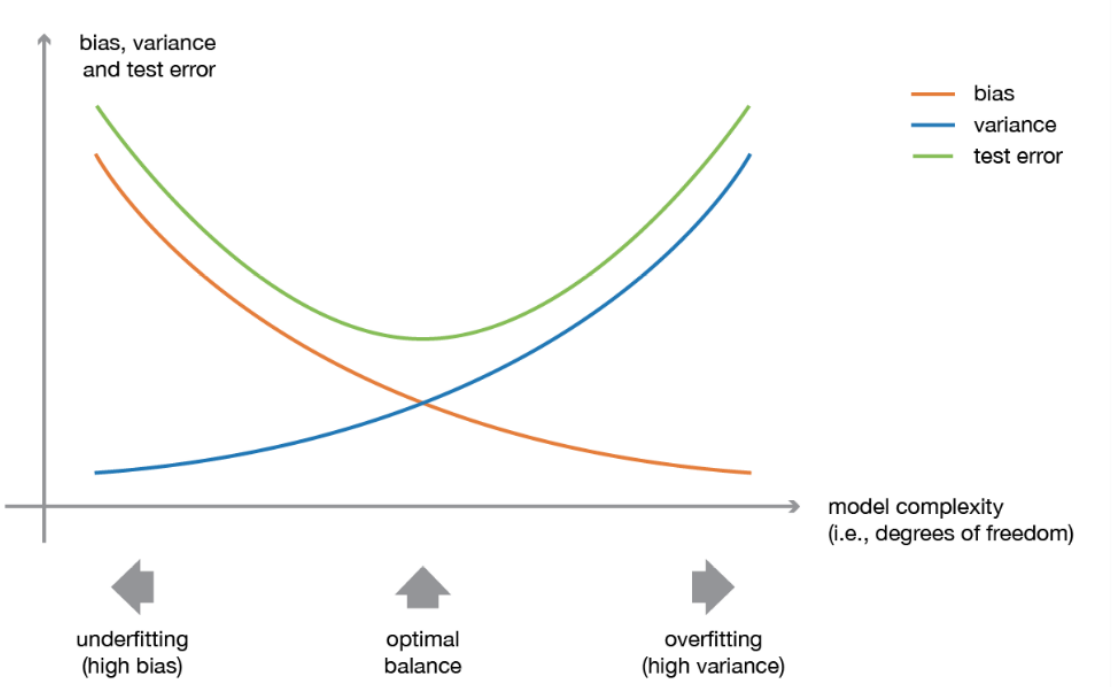


Figure 3.14. Bias vs Variance Tradeoff

- **Bagging (or Bootstrap Aggregating):** It consists of training in a parallel way a bunch of individual models on a random subset of the data (bags) to get a fair idea of the distribution (complete set). It uses a sampling technique called *Bootstraping* in which we create multiple subsets of observations by extracting samples from the original dataset with replacement. The size of the subsets is the same as the size of the original set, but they may be some

repetitions, so that they are not equal to the original one. Models are then run in parallel, as they are independent from each other. The algorithms trained on those different datasets are aggregated to get the ensemble, and the final predictions are determined by combining the predictions from all the models.

- **Boosting:** It is a technique that consists in fitting sequentially multiple weak learners so that each model is fitted giving more importance to observations in the dataset that were badly handled by the previous ones. Each individual model learns from mistakes made by its predecessors. In sequential methods the different combined weak models are no longer fitted independently from each others. The idea is to fit learners iteratively such that the training of a model at a given step depends on the models fitted at the previous steps. The *boosted* ensemble model produces an algorithm that is in general less biased than the weak learners that compose it. Being mainly focused at reducing bias, the base models considered for boosting are often characterized by low variance and high bias, such as shallow decision trees. The final model (strong learner) is the weighted mean of all the models (weak learners).

Very roughly, we can say that bagging will mainly focus at getting an ensemble model with less variance than its components whereas boosting and stacking will mainly try to produce strong models less biased than their components.

3.4.1 Random Forest

Random forest is a classification algorithm consisting of many decisions trees. It uses bagging and feature randomness when building each individual tree to try to create an uncorrelated forest of trees whose prediction by committee is more accurate than that of any individual tree. The low correlation between models is the key, and is ensured thanks to two important methods [128]:

- **Bagging:** Decisions trees are very sensitive to the data they are trained on, and small changes to the training set can result in significantly different tree structures. Random forest takes advantage of this by allowing each individual tree to randomly sample from the dataset with replacement, resulting in different trees.

- **Feature Randomness:** In a normal decision trees' node, we consider every possible feature and pick the one that produces the greatest separation (according to Gini or Entropy metrics) between the observations in the left node and those in the right node. In contrast, each tree in a random forest can pick only from a random subset of features. Sampling over features has indeed the effect that all trees do not look at the exact same information to

make their decisions and, so, it reduces the correlation between the different returned outputs.

To summarize, the random forest approach is a bagging method where deep trees, fitted on bootstrap samples and random subspace of features, are combined to produce an more robust output with lower variance.

Another version of the Random Forest is the Extremely Randomized Trees (or ExtraTrees) model introduced in 2006 by Pierre Geurts, Damien Ernst and Louis Wehenkel [45], which introduces more variation in the ensemble and is faster to compute and shows lower variance than Random Forest.

3.4.2 AdaBoost (Adaptive Boosting)

Adaboost is the first boosting algorithm, that iteratively increases the weight of the samples misclassified by the previous model before the fitting phase. As all others Boosting Algorithms, Adaboost algorithm works in several subsequent steps:

1. Initialize the sample weights
2. Train a shallow decision tree
3. Calculate the weighted error rate of the decision tree (err)
4. Calculate the decision tree's weight in the ensemble with the following formula

$$Weight_{Tree} = Learning\ rate \cdot \log\left(\frac{1 - err}{err}\right)$$

5. Update weight of misclassified samples

$$New\_Weight = Old\_Weight \cdot e^{Weight_{Tree}}$$

   Note that the higher the weight of the tree (the more accurate this tree performs), the more boost (importance) the misclassified data point by this tree will get. The weights of the data points are normalized after all misclassified points are updated.

6. Repeat from step 1 until enough models are built
7. Make the final prediction by adding up the weight of each tree multiplied by the prediction of that tree. In this way, the trees with higher weight will have more influence on the final decision.

A key difference of Adaboost with respect to Random Forest is the effect of the number of trees in the ensemble. In facts, the more tree we add in our Boosting mode, the less bias we get, hence the more likely we are to overfit the data. In Random Forest algorithm, on the other hand, the higher number of trees only impacts the computing time and will not decrease the performance of the model.

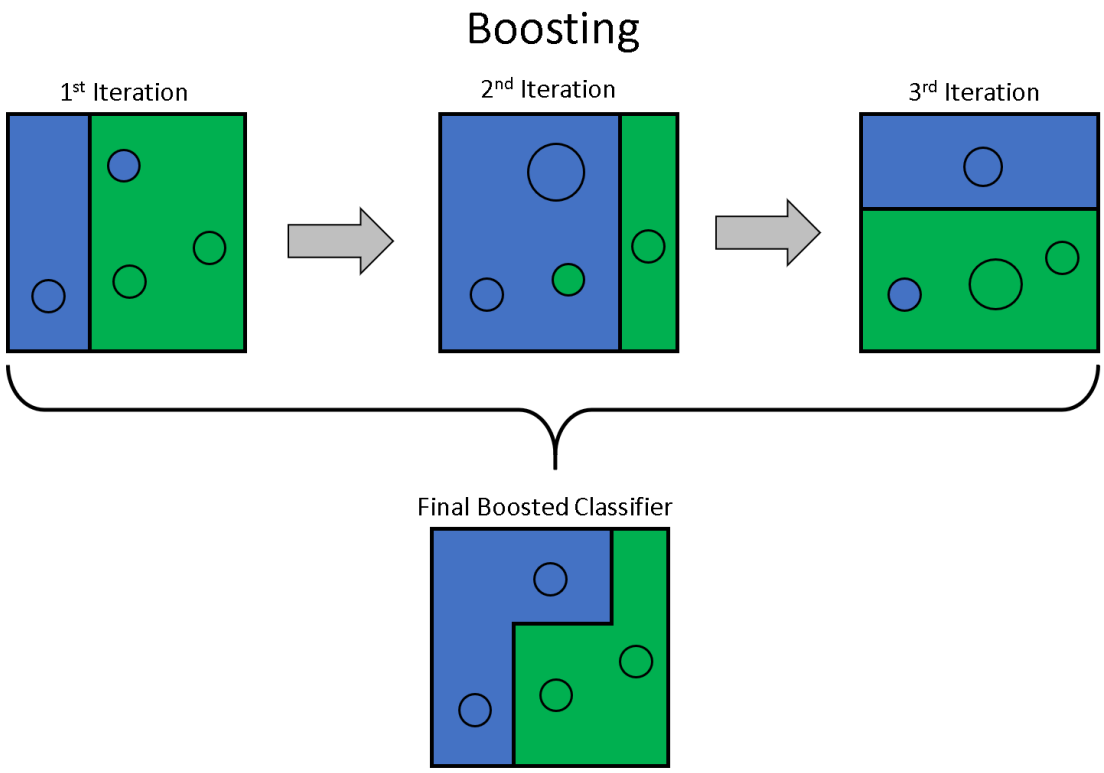


Figure 3.15. Adaboost working principle - picture inspired by [41]

3.4.3 Gradient Boosting Machine (GBM)

Gradient Boosting Machines are also correcting errors made by previous models, but instead of adjusting weights like AdaBoost, each model directly train on the errors of the previous one. This method is named gradient boosting as it uses a gradient descent algorithm to minimise loss when adding subsequent models to the ensemble. Its intuition is built upon 2 key insights:

1. *If we can account for our model's errors, we will be able to improve our model's performance.* For example, if we have a regression model which predicts 3 for a sample whose actual outcome is 1, and if we know the error ($Actual - Predicted = 3 - 1 = 2$), we can fine-tune the prediction. We simply subtract the error (2) from the original prediction (3) and obtain a more accurate prediction (1).

2. *We can train a new model to predict the errors made by the original model.* We can improve the accuracy of a model by training another model to predict its current errors. Then we form a third new improved model that is accounting for the predicted error of the original one. The improved model, which

requires the outputs of both the original predictor and the error predictor, is now considered an ensemble of the two predictors. In gradient boosting, this process is repeated to continually improve the accuracy.

This repeated process is the fundamental of gradient boosting, literally learns from the mistake in a direct way.

1. Train a decision tree

2. Calculate the error of this decision tree, and save it as the new label to predict, so that the new trees will literally train on the errors of the previous ones

3. Repeat from step 1 until the number of trees we set to train is reached

4. Make the final prediction

The Gradient Boosting makes a new prediction by simply adding up the predictions of all trees. Here the learning rate parameter is to be coupled with the number of estimators. If we give less importance to each estimator (lower learning rate) we have to increase the number of trees, but like AdaBoost we have to pay attention not to overfit the model with too many trees.

3.4.4 Extreme Gradient Boosting (XGBoost)

XGBoost is a gradient boosting machine which improves the results obtained by standard GBM models using a combination of software and hardware optimization techniques. It generally provides superior results using when using less computing resources and time. Without going too much into details, here are the advantages of XGBoost with respect to previous generations of GBMs:

- Parallelized Tree Building.

- Tree Pruning using depth-first approach, that improves computational performance significantly.

- Cache and out-of-core computing, optimizing available disk space while handling big data-frames that do not fit into memory.

- Regularization to reduce overfitting.

- Automatic handling of missing data.

- Built-in Cross Validation capability.

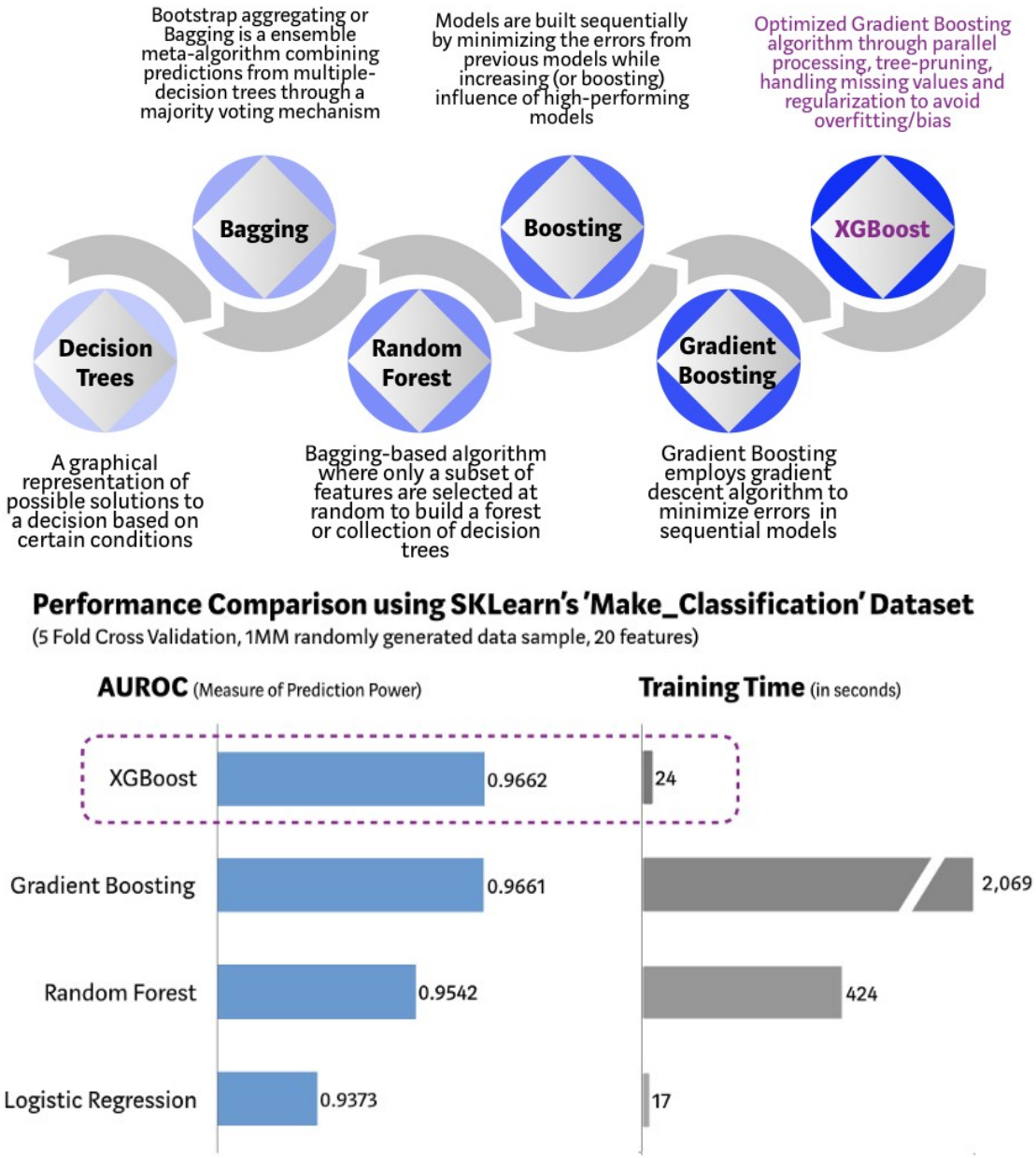


Figure 3.16. Hystory of XGBoost. Source: [78]

3.4.5 Light Gradient Boosting (LightGBM)

LightGBM is a further evolution of XGBoost, optimized for big data by Microsoft developers. Therefore, it has all the innovations of XGBoost with some additional ones. One of the main changes is the way tree is constructed. LightGBM adopts a leaf-wise tree growth strategy, whereas all other GBM implementations follow a

level-wise tree growth (figure 3.17 and 3.18), where you find the best possible node to split and you split that one level down. This strategy will result in symmetric trees, where every node in a level will have child nodes resulting in an additional layer of depth. LightGBM, instead, finds the leaves which will maximize the reduc-

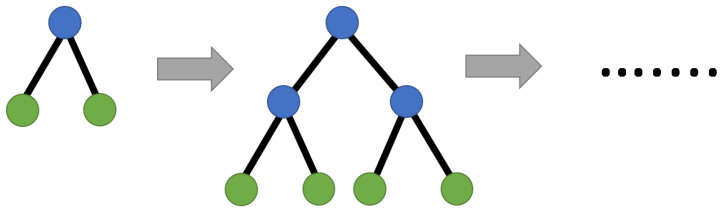


Figure 3.17. Level-wise tree growth

tion of loss. It then splits only that leaf and not the rest of the leaves in the same level. This results in an asymmetrical tree, where subsequent splittings can happen only on one side. Another improvement lies in the way features are considered:

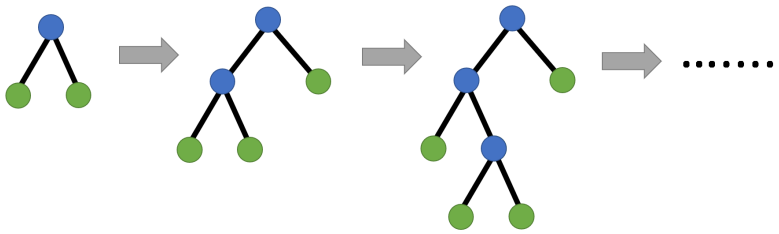


Figure 3.18. Leaf-wise tree growth

they are grouped into set of bins and splits are performed based on those bins. This reduces the algorithm time complexity from $O(n_{data})$ to $O(n_{bins})$ [55]. Leaf-wise tree growth strategy tend to achieve lower loss as compared to the level-wise growth strategy, but it also tends to overfit on small datasets. Our dataset is not considered big enough to need LightGBM, and XGBoost is enough. However, we included it in this Thesis because the future vision is to expand the dataset. The parameters space of LightGBM is rather big, so it is not easy to optimize. However, there are some suggestions taken from the official documentation for what the parameters is concerned, depending on what we want to pursue. For better accuracy

- Use large *max_bin*, but may be slower
- Use small *learning_rate* with large *num_iterations*

- Use large *num_leave*, but may cause over-fitting

For dealing with over-fitting you do the opposite of the suggestions before.

3.4.6 CatBoost

CatBoost is a new generation of Gradient Boosting Machine developed by Yandex and released Open-Source. CatBoost addresses another need, in some way opposite to the one of LightGBM, especially for what the parameters tuning is concerned. In fact, it is supposed to provide good results with no parameters tuning. It automatically supports categorical features coming from different sources, making it more flexible than other GBM algorithms, and uses a novel gradient boosting scheme that helps improving accuracy. Some of the advantages of CatBoost is also on the performance side, with support to GPU computation and reduction of internal latency to speed up the prediction.
Taking the description from the official website, it states:

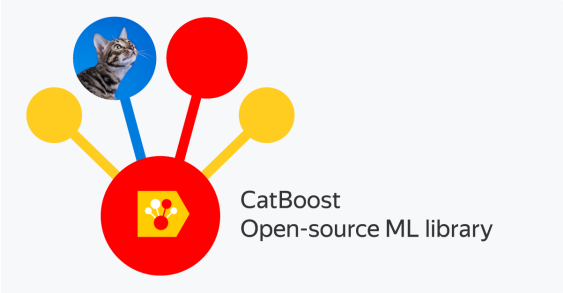


Figure 3.19. CatBoost [18]

CatBoost is an algorithm for gradient boosting on decision trees.
It is developed by Yandex researchers and engineers, and is used for search, recommendation systems, personal assistant, self-driving cars, weather prediction and many other tasks at Yandex and in other companies, including CERN, Cloudflare, Careem taxi.
It is in open-source and can be used by anyone [18]

CatBoost also differs from the rest in another key aspect: the kind of trees that is built in its ensemble. CatBoost, by default, builds Symmetric Trees or Oblivious Trees: This has a two-fold effect [56, 54]:

1. **Regularization:** Since we are restricting the tree building process to have only one feature split per level, we are reducing the complexity of the algorithm, resulting in a regularization strategy.

2. **Computational Performance:** One of the most time consuming part of any tree-based algorithm is the search for the optimal split at each nodes. Since we are restricting the features split per level to one, we only have to search for a single feature split resulting in a much faste inference phase.

Another important detail of CatBoost is that it considers combinations of categorical variables in the tree building process. This helps to consider joint information of multiple categorical features. Furthermore, it has an inbuilt Overfitting Detector. CatBoost can stop training earlier if it detects overfitting.

3.4.7 Voting

Voting is maybe one of the most intuitive ensemble algorithm, because it is exactly what humans do when they have to make a decision: voting. The basic idea is to combine several learners (e.g. classifiers) of any kind (not necessarily decision trees) and make them vote for the best class. Each model makes a class prediction for a defined sample, and vote for it to influence the final decision. There are two ways of voting:

- **Hard Voting:** Each classifier makes its prediction and the final outcome is the class predicted by the highest number of classifier. For example, if classifier 1 (e.g. Decision Tree) predicts class C1 and classifers 2 (e.g. SVM) and 3 (e.g. kNN) also predict class C2, then with hard voting strategy the final prediction will be class C2, as the majority is voting for class C2.

$$Class = \text{mode}(C1, C2, C2) = C2$$

- **Soft Voting:** With this method, we also want to take into account how sure each classifier is about its prediction, by looking at the probabilities for each class, average them across the different classifiers, and select the class resulting with the highest probability. For example, if classifier 1 predicts class C1 with 90% (class C2 with 10%) and classifiers 2 and 3 predicts class C2 with 60%, we get an average probability of:

$$P(C2) = \frac{0.1 + 0.6 + 0.6}{3} = 0.43 = 43\%$$

$$P(C1) = 1 - P(C2) = \frac{0.9 + 0.4 + 0.4}{3} = 0.57 = 57\%$$

$$Class = \text{argmax}\big(P(C1), P(C2)\big)$$

  The final prediction resulting from Soft Voting procedure is hence class C1.

Voting can be used also downstream to other ensemble methods, such as Random Forest or Gradient Boosting. In my software, I used it to combine the outcome of the n best scoring classifiers.

3.4.8 Stacking

Stacking brings the idea of voting one step further. In facts, it combines different parallel classifiers like in *Soft Voting* algorithm, but instead of averaging the probabilities it stack those results, building a new much smaller dataset. It then fits a meta-classifier (generally a simple Logistic Regression) on this new dataset made of probability predictions of previous models rather than on the original features [38].

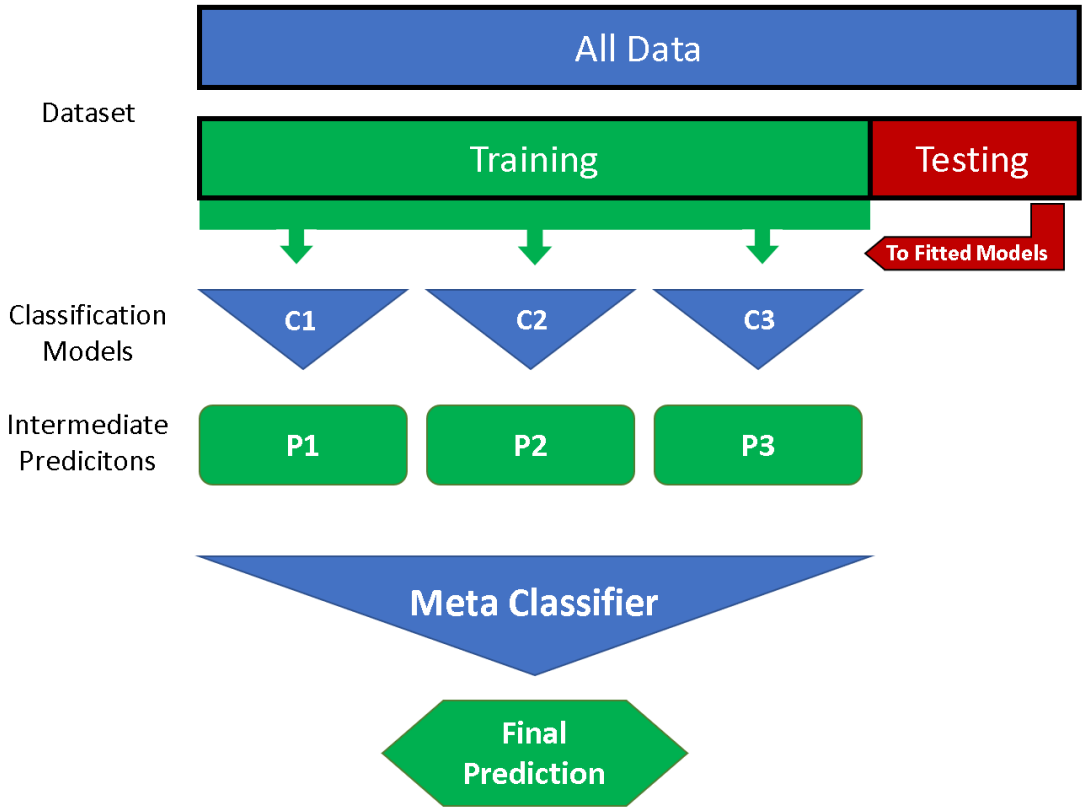


Figure 3.20. Stacking Algorithm Flow

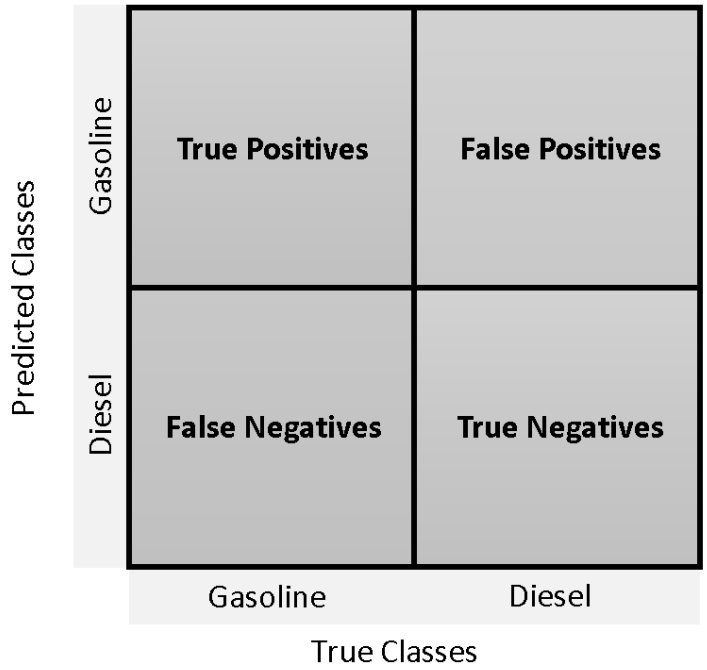
I implemented Stacking in my code with some more tweaks and cross validation strategy in order to avoid overfitting. The same way I did with Voting, the best n classifiers are selected and stacking with logistics regression as meta-classifier is applied. It allowed to improve the results even further, as shown in chapter 5.] sectionPerformance Metrics In previous chapters we briefly explained the main algorithms tested. The ultimate goal is to select one of them in order to give the final result. To achieve that, we need a metrics allowing to score the performance of each estimator. In this chapter I briefly describe the main metrics used, on top

of which the best model is selected [94].

3.4.9 Confusion Matrix (CM)

When dealing with classification tasks, the main tool to use to look deeper into the results is the confusion matrix. It is basically a matrix counting the number of predicted label with respect to the actual label. Each row is indexed by the true label and each columns by the predicted one. The cell of the intersection of the is how many samples labelled as i were predicted as j, as shown in figure 3.4.9. Of course the diagonal shows the correctly classified samples. For simplicity we refer to a binary classification task (like Turbo-Naturally Aspirated, Gasoline-Diesel, Vee - Inline, ...) but the concepts can be expanded to multi-class problems as well From

Figure 3.21. Confusion Matrix



the CM we can compute some useful measures:

- True Positive (TP): Number of labels of positive class (Inline for engine shape, Turbocharged for turbo classification, Gasoline for fuel, but in our case they may be interchangeable) that are labelled as positive (correctly classified).
- True Negative (TN): Same as before, all negative samples correctly classified as negative.

- False Positive (FP): Amount of samples belonging to the negative class but incorrectly classified as positive.

- False Negative (FN): Amount of samples belonging to the positive class but incorrectly classified as negative.

- Sensitivity, also called Recall or True Positive Rate (TPR): it is the amount of true positive over the total amount of positive samples $P = TP + FN$

$$TPR = \frac{TP}{P} = \frac{TP}{TP + FN}$$

  Its value goes from 0 to 1: a recall of 1 means that every item from class C1 was labeled as belonging to class C1 (but says nothing about how many items from other classes were incorrectly also labeled as belonging to class C1).

- Specificity, also called Selectivity or True Negative Rate (TNR): exactly as before, it measures the proportion of actual negatives that are correctly identified as such.

$$TNR = \frac{TN}{N} = \frac{TN}{TN + FP}$$

- Precision, or Positive Predicted Value (PPV): it is computed as

$$PPV = \frac{TP}{TP + FP}$$

  Being the ratio of positive classes correctly classified as positive, precision measures the value of confidence we have that a samples predicted as being positive is effectively positive.

- Accuracy: measures the ratio of positively classified samples, (both positive and negative) over the total amount of samples.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

  Accuracy is a great performance indicator for balanced classes, but as explained in section 2.2.4 it can be misleading for unbalanced classes.

- F_1 Score combines precision and recall into one metric, giving equal importance to both

$$F_1 = \frac{precision \cdot recall}{precision + recall} = \frac{PPV \cdot TPR}{PPV + TPR}$$

- F_β is the generalization of F_1, as it allows to give more importance to precision ($\beta < 1$) or to recall ($\beta > 1$)

$$F_\beta = (1 + \beta^2) \cdot \frac{precision \cdot recall}{\beta^2 \cdot precision + recall}$$

- Cohen Kappa: tells us how much better is our model over the random classifier that predicts based on class frequencies.

$$k = \frac{Accuracy - Accuracy_{RandomClassifier}}{1 - Accuracy_{RandomClassifier}}$$

  It is a great alternative to accuracy when the classes are not balanced

- Matthews Correlation Coefficient MCC is a correlation between predicted classes and ground truth, and can just be computed from the Classification Matrix as follows

$$MCC = \frac{TP \cdot TN - FP \cdot FN}{(TP + FP) \cdot (TP + FN) \cdot (TN + FP) \cdot (TN + FN)}$$

3.4.10 Curves

Classification algorithms often make use of probability to predict the various classes, hence they compute for each sample the probability of belonging to each of the possible classes, and after setting a threshold they define the class. This threshold is a tradeoff between two different performance metrics, and varying this we can plot the trend of these metrics, obtaining insightful curves. These are the Receiver Operating Characteristic Curve (ROC) and the Precision Recall Curve (PR)

- **ROC Curve:** this curve shows the relationship between Sensitivity and Specificity by varying the threshold level. Generally in the x axis the parameter $1 - Specificity$ is used. Besides the shape of the curve, an interesting parameter could be computed starting from it: the Area under the Curve (AUC). This chart is bounded horizontally and vertically between 0 and 1, hence the maximum that AUC can achieve is 1. Any trivial classifier has the points (0,0) and (1,1) belonging to the curve, by putting all samples in one of the two classes, reaching sensitivity = 0 for specificity = 1 ($1 - specificity = 0$), and sensitivity = 0. By increasing sensitivity, a random classifier would linearly decrease specificity, reaching an AUC of 0.5 (Figure 3.22). This is the reference value for judging our classifier: if $AUC < 0.5$, a random classifier would have performed better. For good results, we want sensitivity and specificity to be both higher than 0.5.

- **PR Curve** is the trend of Precision and Recall, again by changing the threshold. Similar reasoning as before could be made.

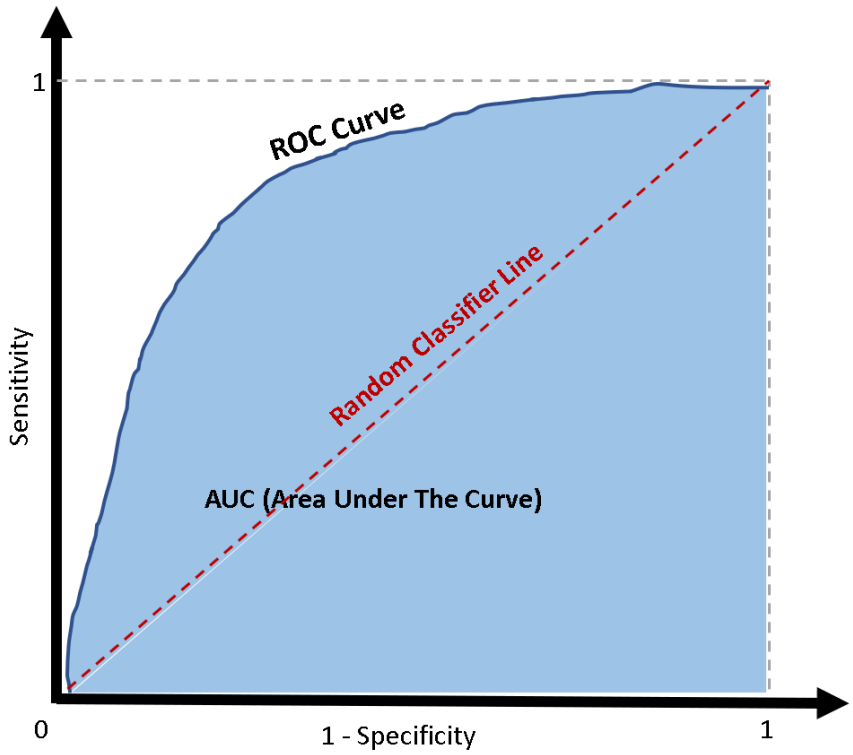


Figure 3.22. Example of ROC Curve

Both results are computed only for binary classification problems, but we are able to extrapolate this concept to multiclass problems computing the score at per-class level. We can score the results of each class against all the others combined (One Versus Rest, OVR) or against each of the other (One Versus One, OVO), resulting in a binary classification problem, and then merge the results obtained by each class. This merge may be done at the *macro* level, simply averaging the scores, at *weighted* level, i.e. weighting the score with the number of sample of that class, or at *micro* level, which also takes into account the impact of class size. Another interesting curve to observe is the **learning curve**, which shows the score (e.g. Accuracy) on train and validation set over the training process. It tells us how the algorithm is improving over the training "epochs" and whether it is overfitting. If we see that our training error is decreasing significantly, whereas the validation is constant or even increasing, we can get some insights that the model is overfitting and the training phase should be stopped. We can deduce that the parameters for regularization are too permissive, or that some features are not informative enough and mislead the judgement of the classifier during validation.

3.4.11 Reconstruct Audio

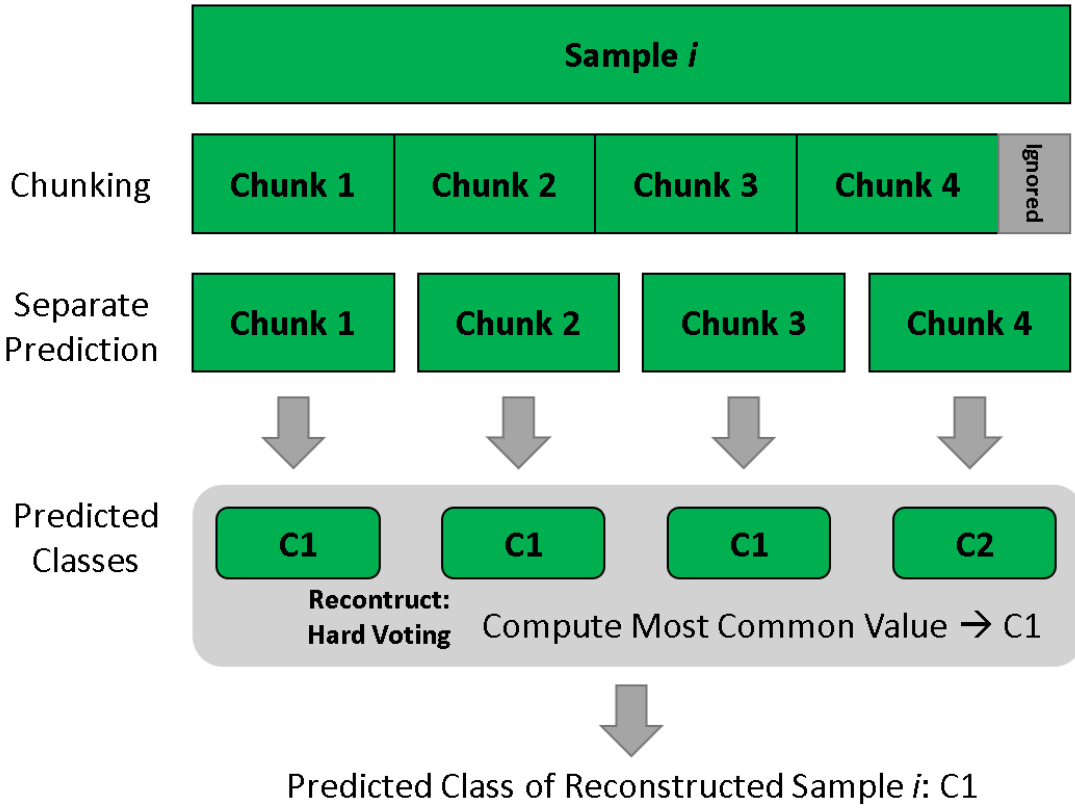


Figure 3.23. Sample reconstruct procedure: each sub-chunk votes for the class of its "parent" chunk. Even though they are treated separately, they belong to the same sample, on which actual classification is made

As outlined before, I decided to chunk the audio signal in pieces of the same length, to reduce sensitivity to noise and have features consistent among the dataset. This allows us get more robust predictions on the original sample. In fact, we can consider to track back the predictions of each sub-chunk by its ID, so that each of them will vote for the class of its parent sample. For example: sample with $ID = 15$ is 32 seconds long. we chunk it with 3 seconds pieces, resulting in 10 sub-samples with $ID = 15$ (and 2 seconds lost). If 6 of them are predicted to belong to class C1 and the other 4 to class C2, when we reconstruct the sample the final prediction would be C1, because of the majority of sub-samples "voting" for class C1 (Figure 3.23). We could really see an improvement on the results with this technique, as detailed in chapter 5.

Chapter 4

Framework

Following the structure of this Thesis, it is clear that the work code framework is divided into three main parts: From Sound to Features (Chapter 2), From Features to Results (Chapter 3), and Results Evaluation (Chapter 5)

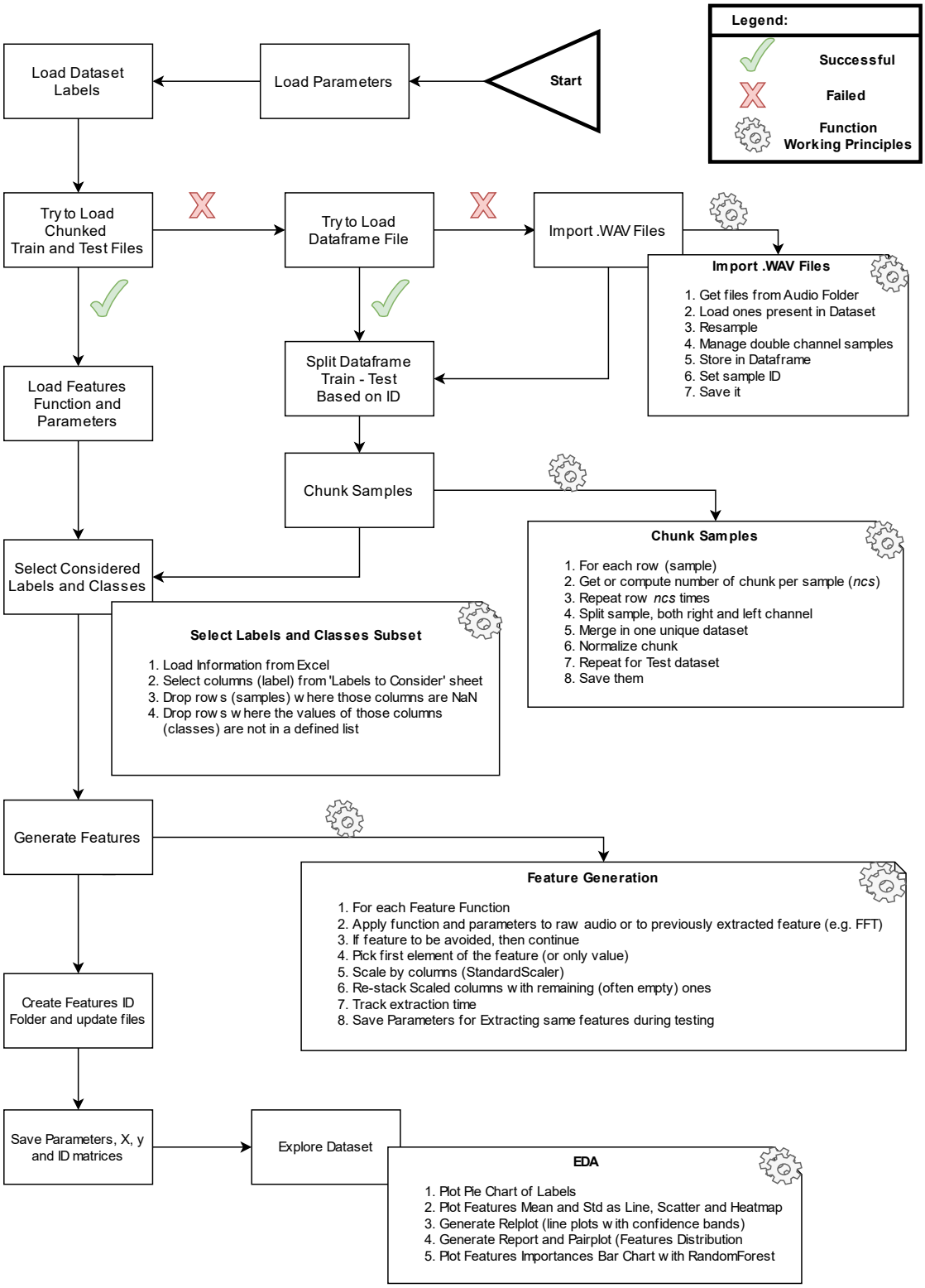
In this chapter I will provide a major overview on how the framework works, including input and processing pathways. It is a collection of the pieces explained in the previous chapters and wraps them all together to form the big picture. This explanation is carried on without the burden of technical details. For those specific aspects, such as dealing with particular data structure limitations and computing performance, I refer to the actual documentation of the framework, provided as exhaustive comments inline within the code.

4.1 From Sound to Features Flowchart

To provide an in depth look into some of the sub-functions for feature extraction and their general usage, a flowchart comes very useful. (figure 4.1) Here I will only focus the attention on the following sub-functions:

1. **Import WAV files**, responsible for the conversion between standard uncompressed audio format to *numpy* array and *pandas* dataframe.
2. **Chunk samples**, responsible for the chunking of the samples, as explained in section 2.2.3, and the management of the dimension of the dataset in case we want to use a smaller version for developing purposes.
3. **Select Labels and Classes Subset**, used to filter the database and remove the empty lines.
4. **Feature Generation**, the core function of this part, responsible for feature extraction and feature scaling.
5. **EDA** for database exploration, as outlined in section 2.5.

Figure 4.1. From Sound to Features Framework Flowchart



4.2 From Features to Results Flowchart

This part is the actual machine learning where estimators are fitted to the data and a grid search over defined parameters is run. Here in flowchart of Figure 4.2, the functions I want to focus the attention on are:

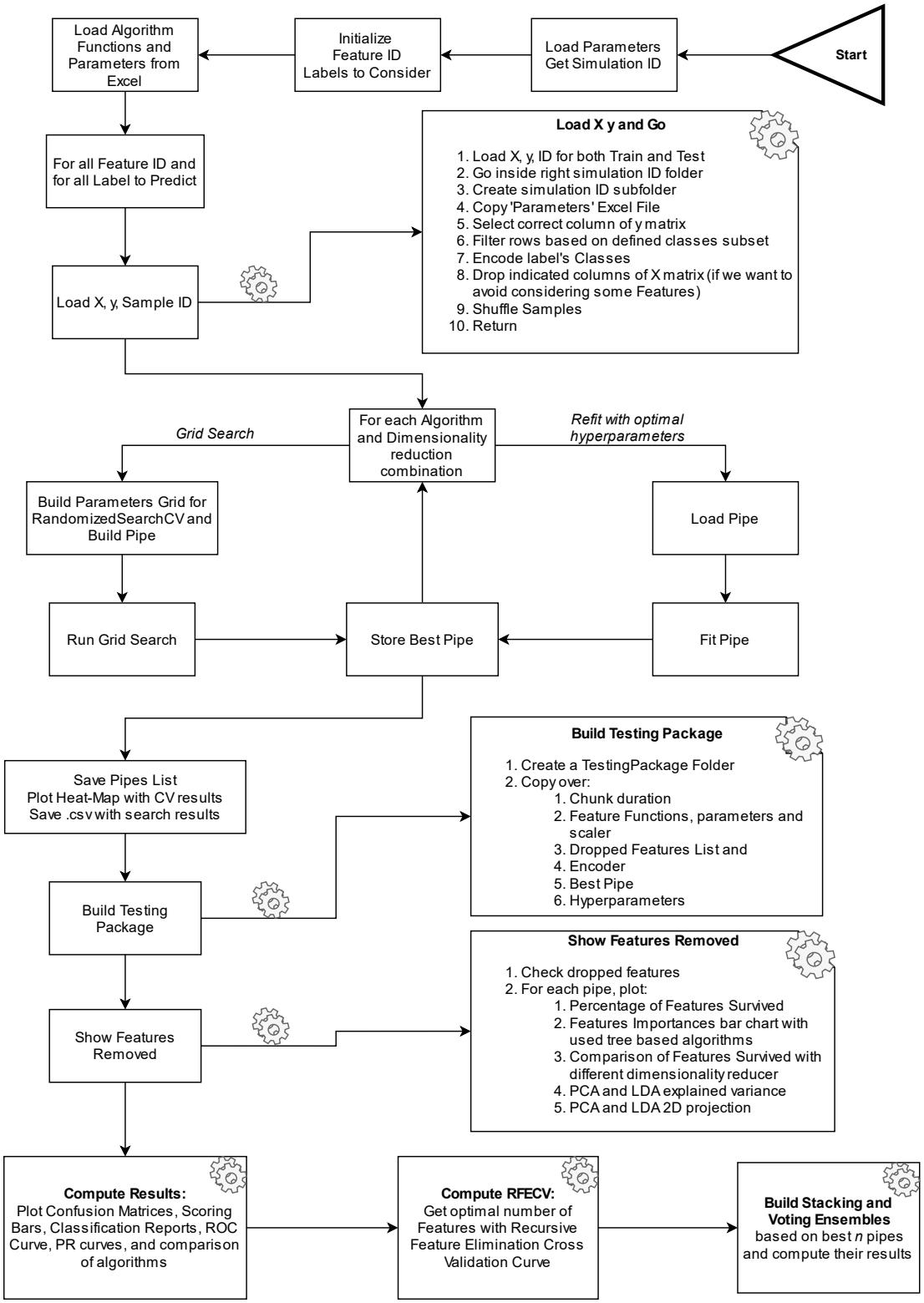
1. **Load X, y and go** which takes care of all initialization steps of the process
2. **Build Testing Package** which packages an environment stored in a folder, allowing to do real testing on new samples. It will provide the whole process (from loading the raw audio to the prediction of the class)
3. **Stacking and Voting Ensemble** is a way to combine the best n estimators in a stronger one, either with stacking or voting strategies

4.3 Results Evaluation Flowchart

The functions for evaluating the results are used straight after all estimators have been fitted, so they belong to the same flowchart shown in Figure 4.2. For the purpose of results evaluation several plots are saved, in order to have insights on how the algorithms are performing and select the one(s) that are giving the highest Accuracy, Precision, F1 or ROC-AUC. There are three main functions dealing with this purposes:

1. **Compute Results** plotting the results in a lot of different ways, as shown in next section
2. **RFECV**: Recursive Feature Elimination with Cross Validation uses a Random Forest Classifier and computes the cross validated accuracy several times, removing one feature at time until only one is left. In this way one can see how many features are truly necessary for a good classification task.
3. **Show Features Removed** is a function that tries to give some insights about the features in relations with the output. Some tree-based classifiers are able to attribute *importance* to the features they split. Furthermore one can look at the first components of PCA and LDA and their explained variance.

Figure 4.2. Machine Learning Fitting Algorithms Framework Flowchart



Chapter 5

Results

In this chapter I am going to present the results obtained after fine-tuning of procedures steps. The class prediction of which I obtained the most successful results are:

- Turbo
- Fuel
- Cylinders amount

Those results are proposed in this specific order, because the correlation among labels enables a sequential investigation of the characteristics of the engine. I went through several iterations before deciding which engine characteristics is the easiest to infer without previous knowledge about other ones, resulting at the end in the list shown before. I first predicted the engine fuel type, then used this information to infer the presence of turbocharging, and finally used both information to predict the number of cylinder. The production of the best results is a long process, searching for the best of the potentially infinite combinations:

- Which features to extract and
- Parameters of the extracted features
- Dimensionality reduction method
- Dimensionality reduction hyper-parameters
- Classification algorithm
- Classification algorithm hyper-parameters
- Metric to optimize

To do this I ran a grid search several times over the parameter space, but since I realized that the most important feature are FFT and PSD, minor tweaks in features parameters did not change the performance of a considerable amount. As explained in Section 3.4.8, there are a lot of ways to judge the performance of a classifier, and since I explored several classifiers with several preceding dimensionality reducers, the possible pictures are a lot. For this reason, it was needed to build a function that automatically create several images for each fitted pipe (sequence of one dimensionality reducer and one classifier). The main pictures used to evaluate the performance are:

- A comparison between confusion matrix on train and on test dataset, one for each pipe.
- A comparison between Classification report of train and test. Classification report (CR) is a tool allowing to compute different metrics at the same time, and compare them in a heat-map matrix.
- A chart containing all confusion matrices computed on the test dataset, because quite all the ones computed from the train dataset were almost perfect diagonal. This was exploited to see at glance a fast comparison among all different algorithms.
- A bar chart showing some selected metrics in a bar chart, again used to compare the different algorithms.
- ROC curve and Precision Recall Curve, as well as feature importances and confusion matrix, allowing to look more in depth the performance of the best selected classifiers.

In the following sections, I will present some of the best algorithms for each target label with different pictures, and finally I will show a table with the very best performance figures, in order to set a benchmark for future research in the field.

5.1 Target Label: Turbo

In the sequential procedure aiming at classifying the engine, the first step is to predict the type of aspiration, that is either turbocharged ($turbo = 1$) of naturally aspirated($turbo = 0$). In the following Sections, I will provide several charts showing the results obtained in all different combination of dimensionality reduction methods and learning algorithms. I will then focus my attention on the ones that performed best, by looking at some of metrics and curves described in Section 3.4.8. At the end, the final Confusion Matrix for turbo is shown.

Comparison among algorithms

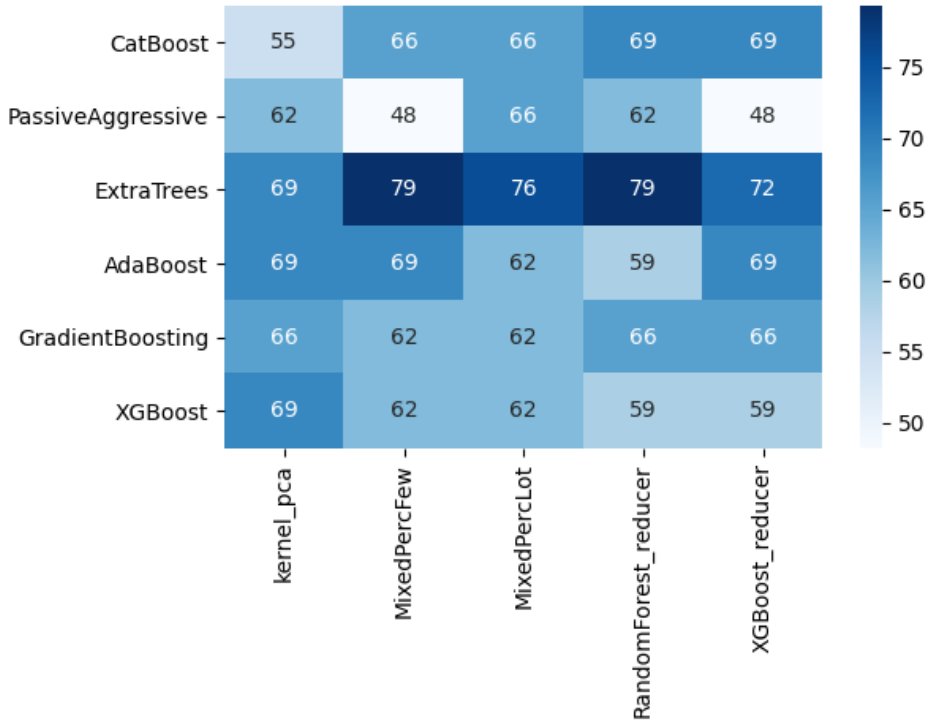


Figure 5.1. Comparison of the F1 Score for Turbo prediction

As shown in Figure 5.1, Extremely Randomized Trees algorithm outperformed the others independently from used dimensionality reduction method. In Figure 5.2 we can see the reasons why the other algorithms perform worse: It is not because they completely misclassify some classes, but simply they are less "sure", and tend to distribute the samples less specifically. Furthermore, if we look at the confusion matrices of on the training set they are perfect, meaning that the algorithms tend to overfit. We can state that they have high variance. Random Forest and ExtraTrees

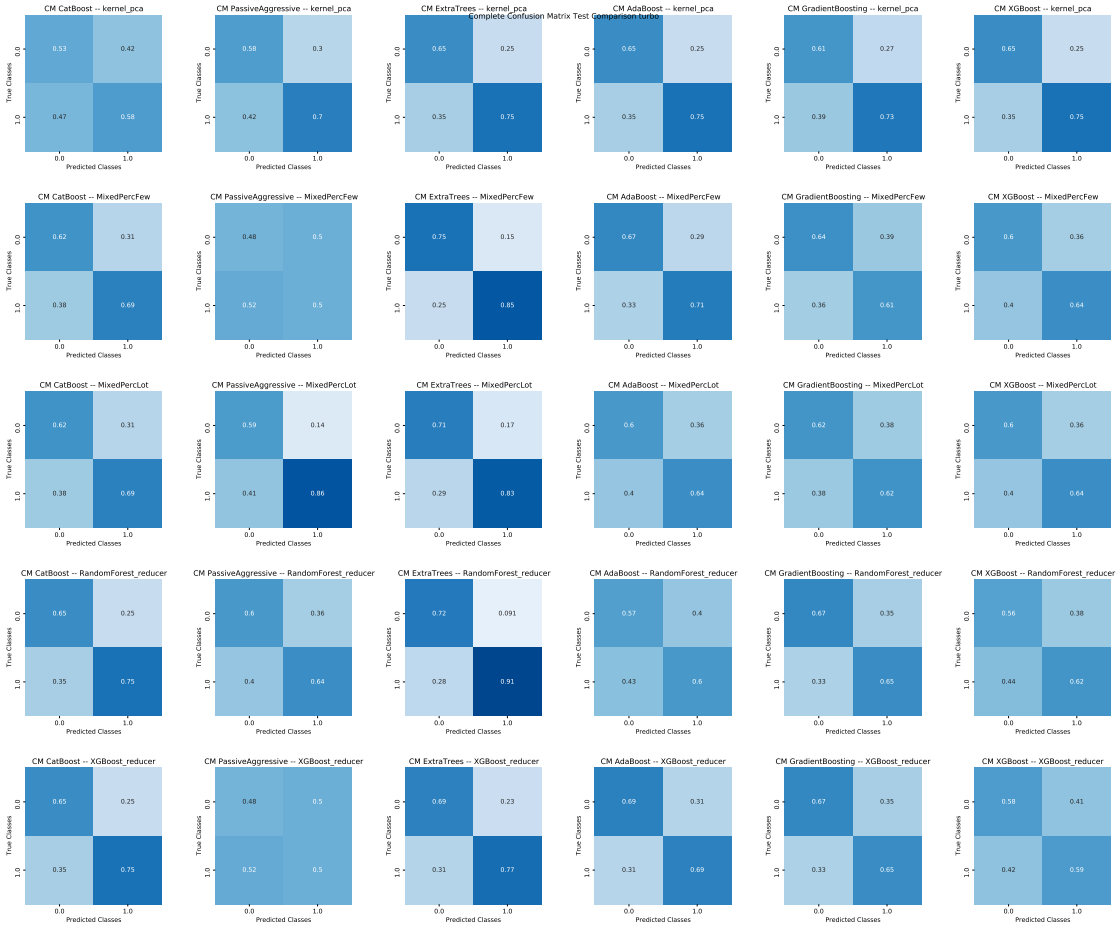


Figure 5.2. Comparison Confusion Matrices for Turbo prediction

are specifically designed to reduce the variance of the problem, and therefore the results obtained by the combination of Random Forest as dimensionality reducer and ExtraTrees as learning algorithm yields the best performance.

Confusion matrices and Curves of the best performers

It seems that univariate features selection is a better reduction method in this case, but if we look at the other metrics, such as Precision, Recall and F1. For this reason, I decided to select Random Forest as a ultimate reducer for ExtraTree when prediction the presence of turbocharger. These difference can be observed in Figures 5.4, in two views than facilitate the comparison.

As a last step, we look at the Confusion Matrix in Figure 5.5 to have a more detailed view of the above-mentioned metrics.

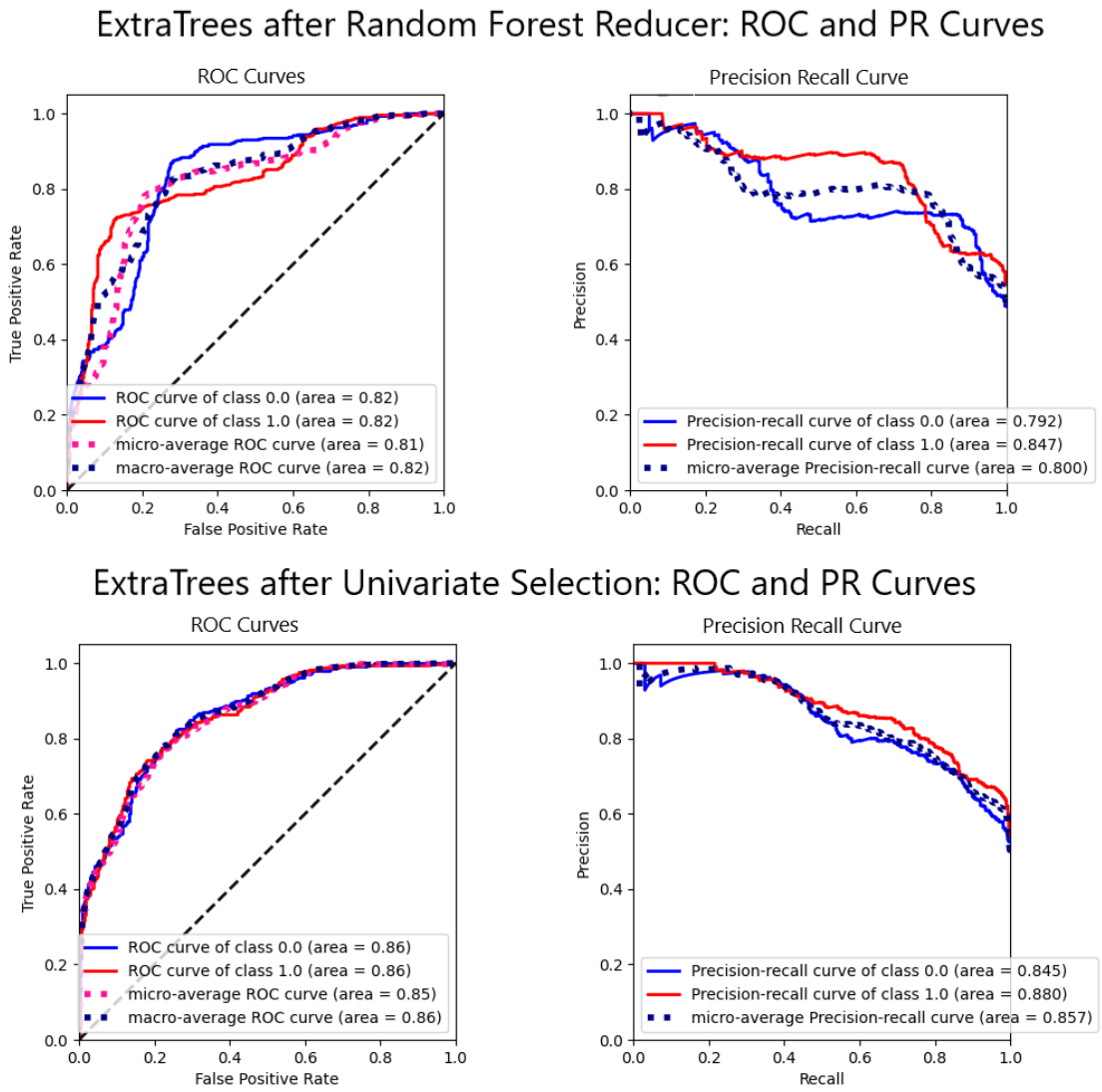


Figure 5.3. ExtraTree for Turbo: ROC and PR Curves. Comparison between Random Forest Reducer (ROC-AUC= 0.82 and PR-AUC= 0.8) and Univariate Features Selection (ROC-AUC= 0.86 and PR-AUC= 0.86)

Informative Features

To understand our process and optimize feature extraction, let's have a look at the features selected by the dimensionality reduction methods in Figure 5.6. This chart has to be read in the following manner: The dots at the very bottom of each square represents the features that are discarded (e.g. *MFCC Var* feature was discarded by all algorithms), and each algorithms represents with a colored dot the features kept (e.g. *FFT Kurt* was kept by everyone, and only some element of the

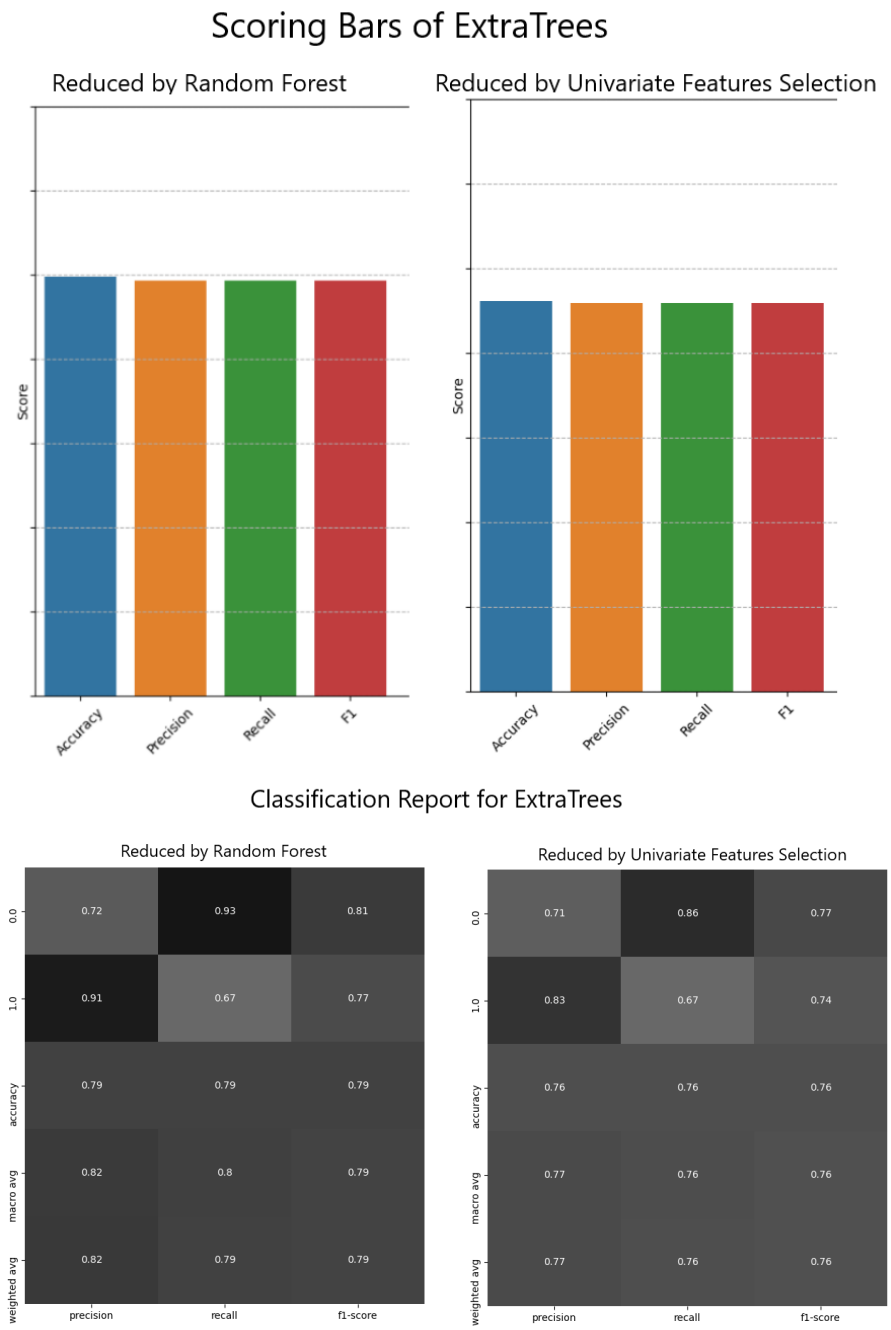


Figure 5.4. Scoring Bars and Classification Report: the performances of Random Forest as reducer are higher in every metric with respect to Univariate Selection

DWT Kurt feature vector were selected by some algorithms). They are shown in different y-levels only for appearance purposes, so that they can be: y-axis does

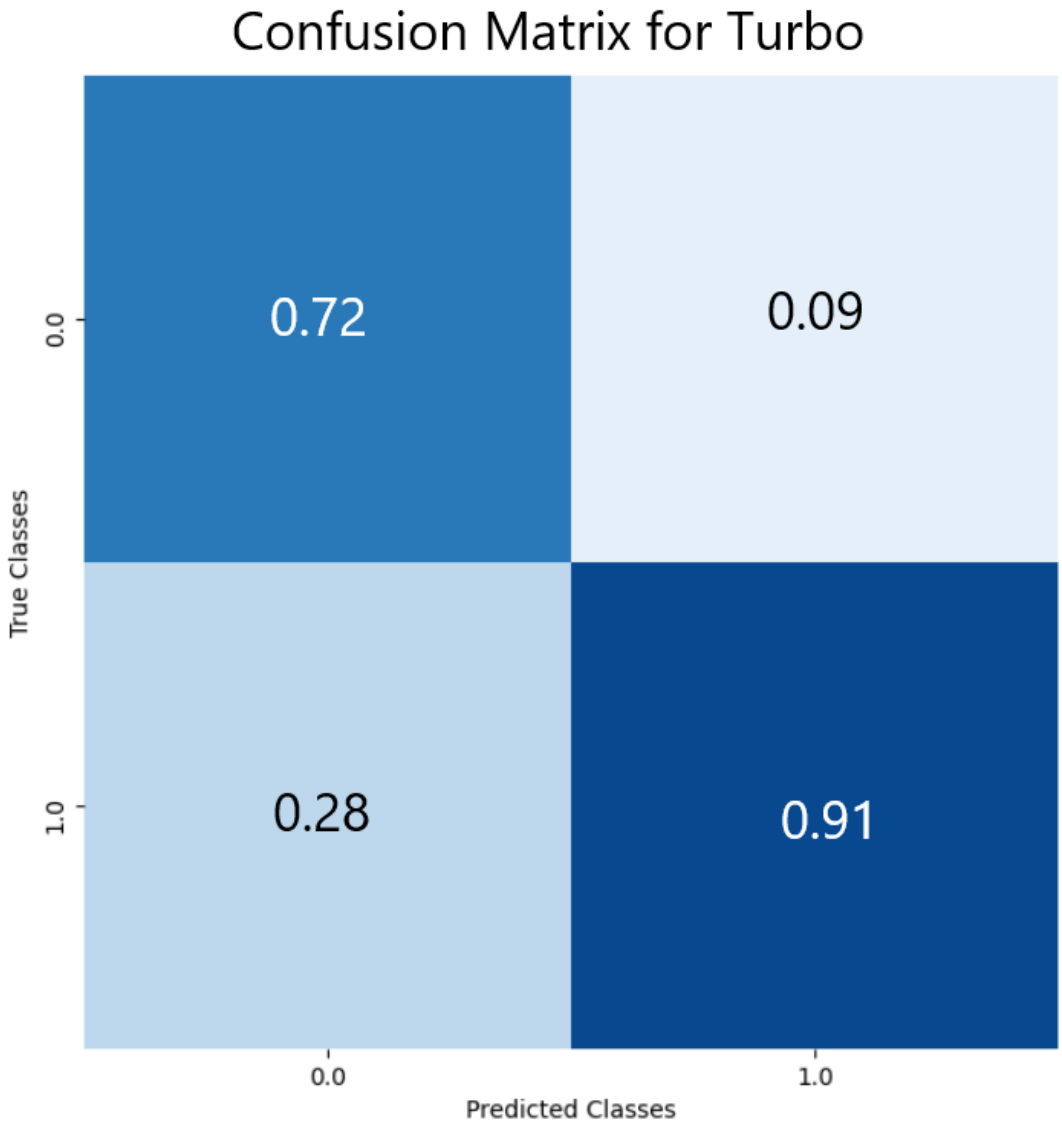


Figure 5.5. Confusion Matrix of ExtraTrees reduced by Random Forest: we noticed a strong diagonal, with one of the values above 0.9. This result is satisfactory

not represent quantitative values.

For a more aggregate look about the quantities of features kept, we refer to Figure 5.8 After the feature selection process, ExtraTrees also ranks the features and gives them an "importance". It says how relevant they are for the learning process. In Figure 5.7, the values of the importance given to each feature value is displayed. In this chart we can see that some of the square are empty. This is

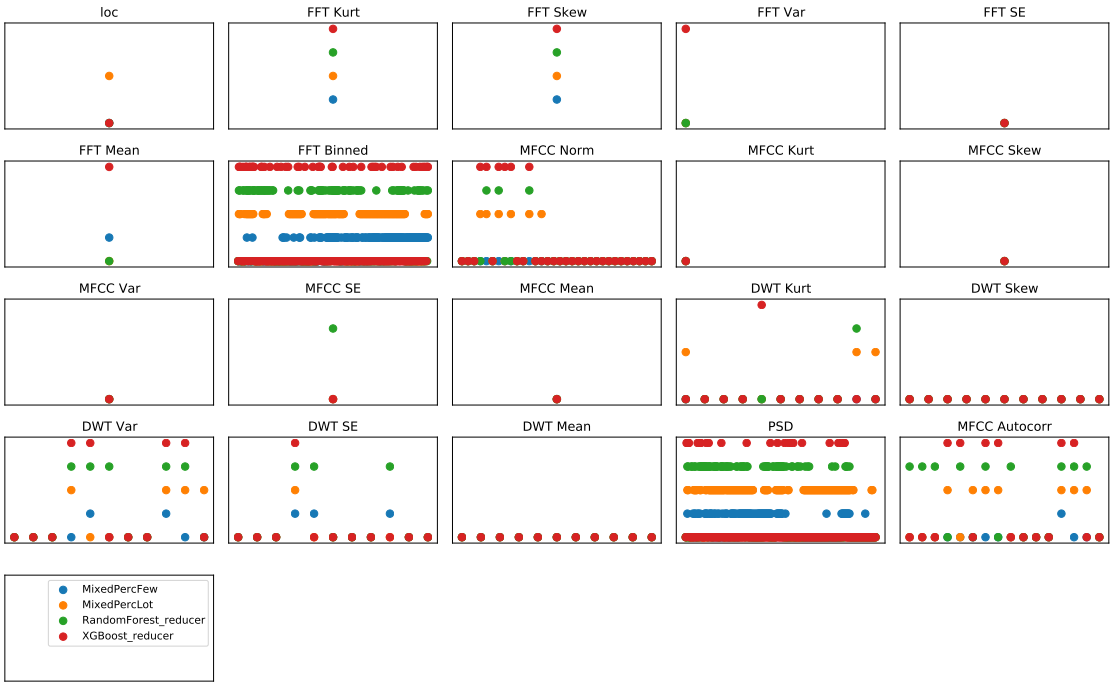


Figure 5.6. Dot Chart for comparison of features selection for Turbo

simply because those features did not pass the first filtering level of Random Forest Reducer.

Running Time and Hyperparameters

In order to provide useful information for future work and for reference, I provide here the list of hyperparameters and the time needed to run the fitting procedure.

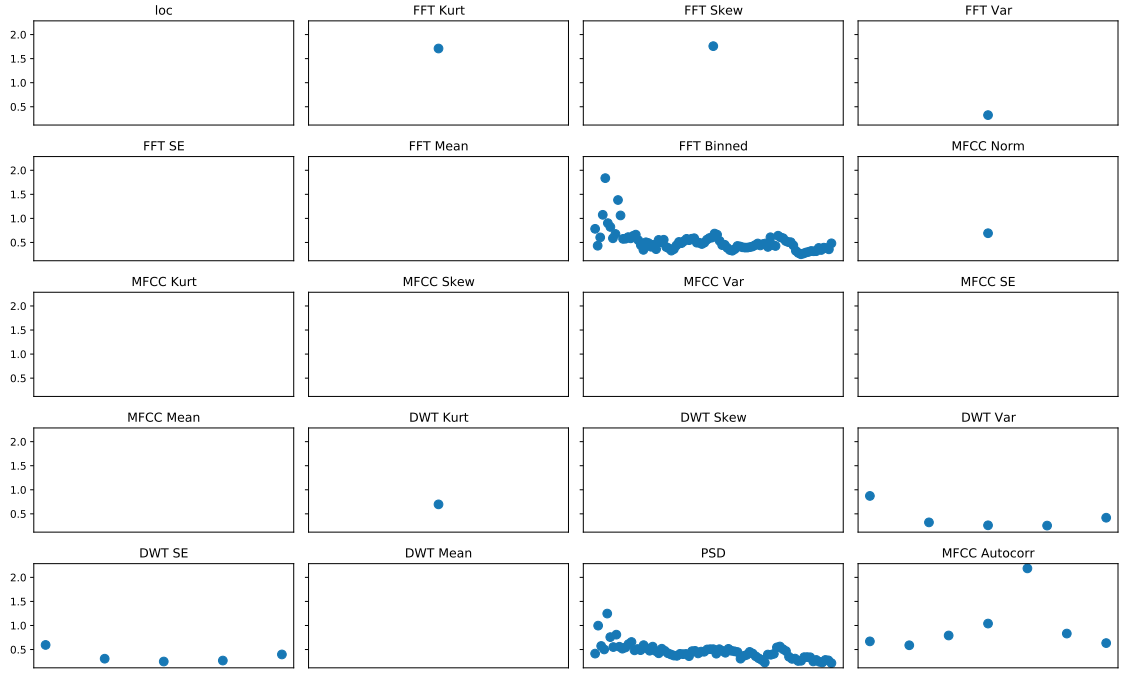


Figure 5.7. Detailed features importance by ExtraTrees in Turbo classification

Hyperparameter	**Value**
max_features	log2
criterion	gini
n_estimators	2000
max_depth	443
min_samples_leaf	5
min_samples_split	2
ccp_alpha	0
min_impurity_decrease	0
min_weight_fraction_leaf	0
fitting time	42sec

Table 5.1. Hyper-parameters of ExtraTrees

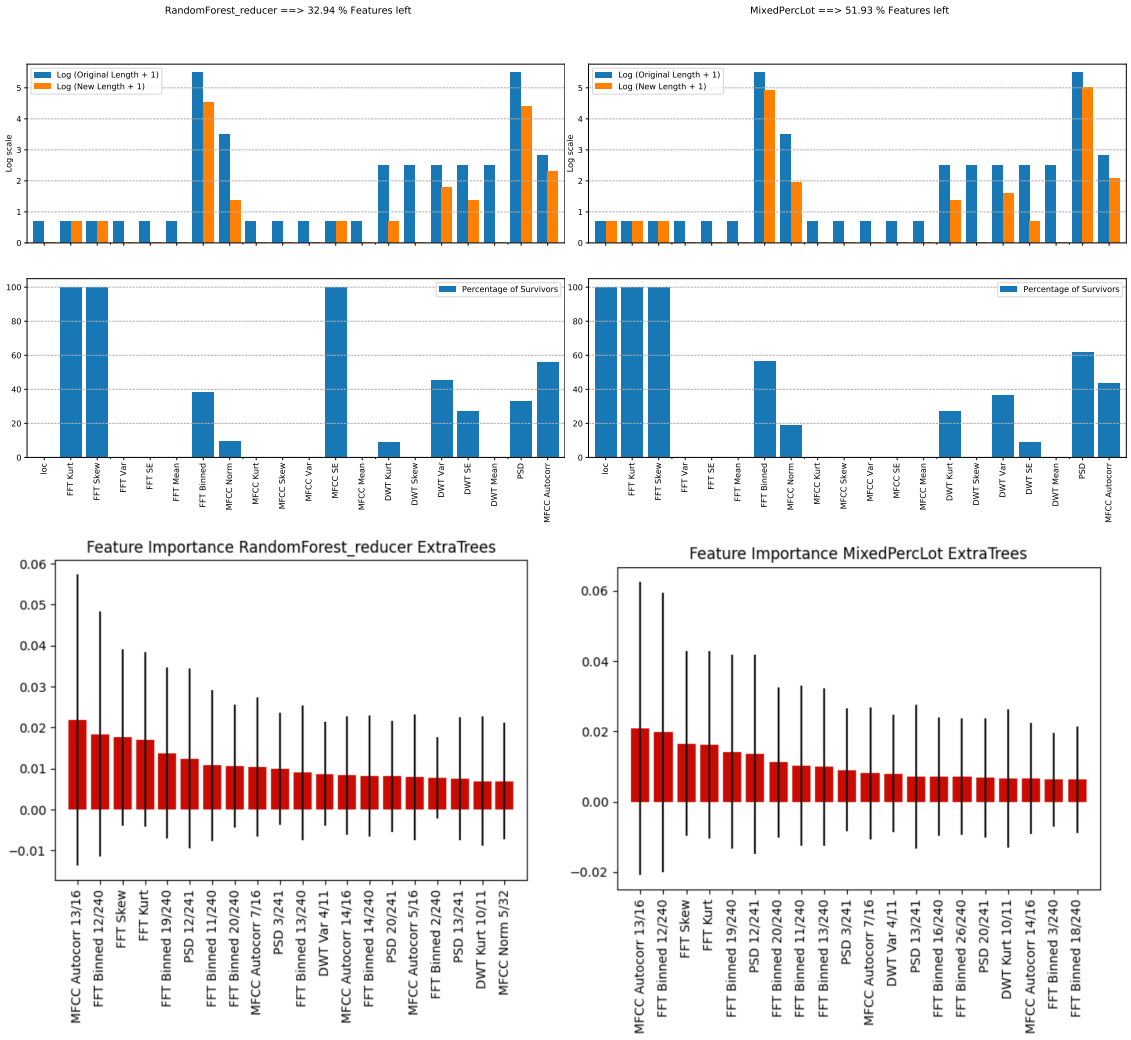


Figure 5.8. Bar Chart for features selection and importance for Turbo

5.2 Target Label: Fuel

Fuel type prediction is crucial when dealing with diagnostics, because most of the engine-related issues are tightly related to the kind of fuel. For example, when running a vibroacoustic diagnostics tool, the software will never predict "ignition coil failure" if it knows that the engine it is inspecting is diesel powered. Now the algorithms can use the information about the presence of Turbo as feature, because this was computed in the step before. In the following Sections, I will provide several charts showing the results obtained in all different combination of dimensionality reduction methods and learning algorithms. I will then focus my attention on the one that performed best, by looking at some of the values and curves described in Section 3.4.8, as well as the Confusion Matrix. After that, I will show the improvement that comes from the reconstruction of the audio explained in Section 3.4.11 and the benefit coming from using a stacking classifier.

After several experiments, I want to present two particular results, obtained with completely different features sets:

1. One more **essential**, where only few features are computed, and in particular long features vector (FFT, PSD and MFCC Autocorr) are discarded. It is useful to consider a shorter version of the features in order to save computational time in case the results are not sufficiently weaker than the complete option.

2. One more **complete**, where all features discussed in Section 2.3.1 are extracted. The results are more powerful, but at the expense of running time.

5.2.1 Comparison among algorithms

As shown in Figure 5.9, the two matrices showing the *F1 score* for each combination of classifier and feature selector, the scores are higher than the ones obtained for turbo classification. More specifically, we can notice that boosting algorithms are performing better than the others and that PCA reduction methods are not effective in capturing the informative features. As opposed to Turbo classification, where models tend to overfit the training data in Fuel they are not perfect when we try to classify the training set again. Several algorithms are reaching high performances, so we can say that this problem have low variance. Boosting algorithms are specifically designed to reduce bias and keeping variance low. For this reason, we can see that the performance of Adaboost, Catboost is considerably higher than ExtraTrees, especially in the set with a lot of features. Furthermore, in the feature set with FFT, PSD and MFCC Correlated (*Complete* set, matrix on the right of Figure 5.9) Gradient Boosting struggles, as opposed to XGBoost that achieves good performance. On the other hand, Gradient Boosting is very good when the number of features is lower, as in the Figure on the left.

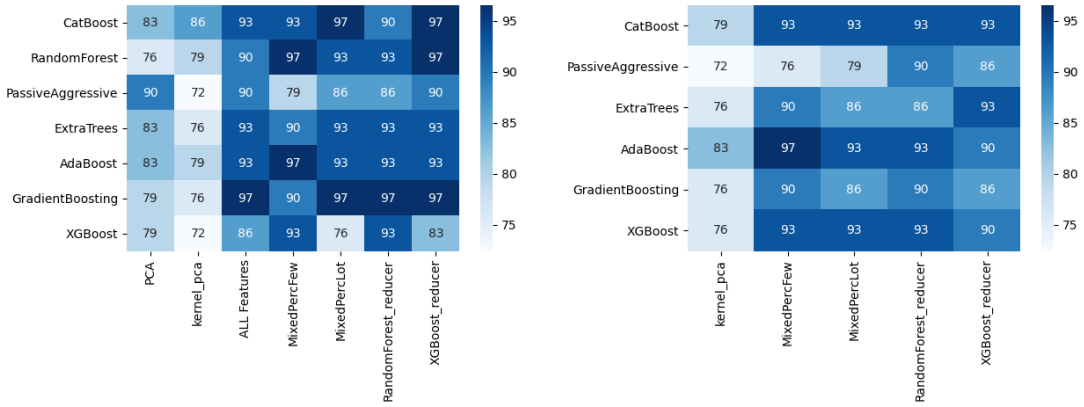


Figure 5.9. Comparison of F1 Scores for Fuel prediction: *Essential* Feature Set on the left and *Complete* Feature Set on the right. We can see that boosting algorithms generally performed better, whereas PCA reducers are not effective enough to capture the important features

5.2.2 Informative Features

In order to choose which algorithm to use in the definitive solution, it is interesting to look at the features and their importance first, and then compare ROC and other metrics. In the following I take into account CatBoost and Adaboost, as well as Gradient Boosting and XGBoost for feature set 1 and 2 respectively. There are two different aspects to consider here:

1. **Concentration:** how the "importance" is spread across different features. We refer as *concentrated* feature importance if few features reach more than 60% of the total importance, and *distributed* if the importance is spread across different features.

2. Absolute **importance of Turbo** feature. Surprisingly, some algorithms consider turbo as less important than other feature values. This could mean more robustness. In any case, every dimensionality reducer maintains it, so the responsibility of discarding is handed to the classifier.

CatBoost

Catboost is particular because it is one of the most concentrated in terms of importance given to the features. This is probably due to the peculiarity of this algorithm to work with categorical features. Another pattern is to be observed: the more are the features the more importance CatBoost gives to Turbo. In Figure 5.10, we see that turbo reaches around 50% of importance in the feature set 1 (*Complete* one)

even if their number is reduced to 50% by univariate feature selection. Considering features set 2, on the other hand, CatBoost gives 30% importance to turbo, if reduced their amount is reduced to 30%.

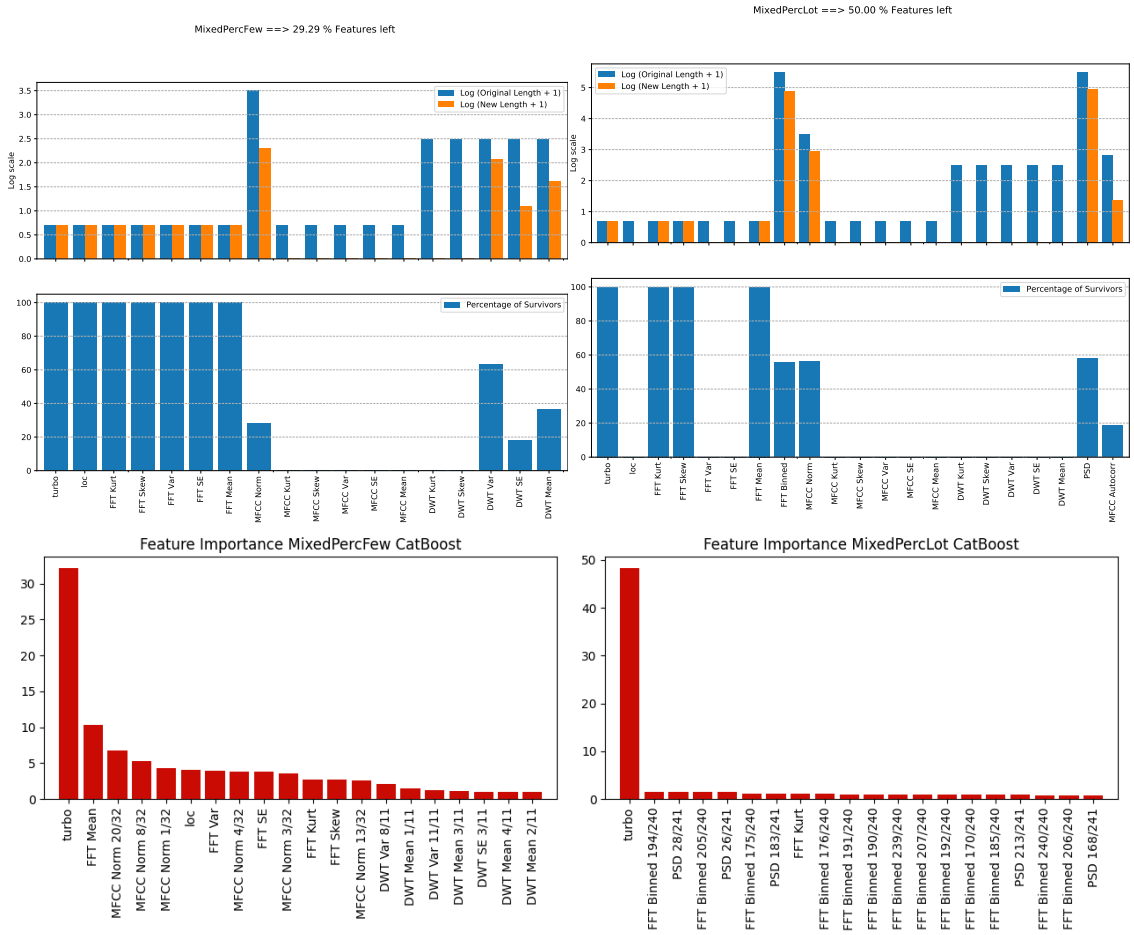


Figure 5.10. Feature Selected by CatBoost in two different features set: *Essential* on the left and *Complete* on the right, where turbo receives more importance than the other features combined

AdaBoost

AdaBoost is surprising because even though it reached very high F1 score (0.97) its certainty about the features importance is very low. As opposed to CatBoost, it distributes the importance among the features, and the most important one reaches only around 20%. This is even more evident in the *Complete* feature set, where the importance is so distribuited that the first one has less than 10% (Figure 5.11 on the right). Furthermore, the uncertainty is even wider. Another thing to notice is

that the feature "turbo" is not judged as having privileged relevance among other features, even though we observed a good correlation with the fuel in Section 2.2.4.

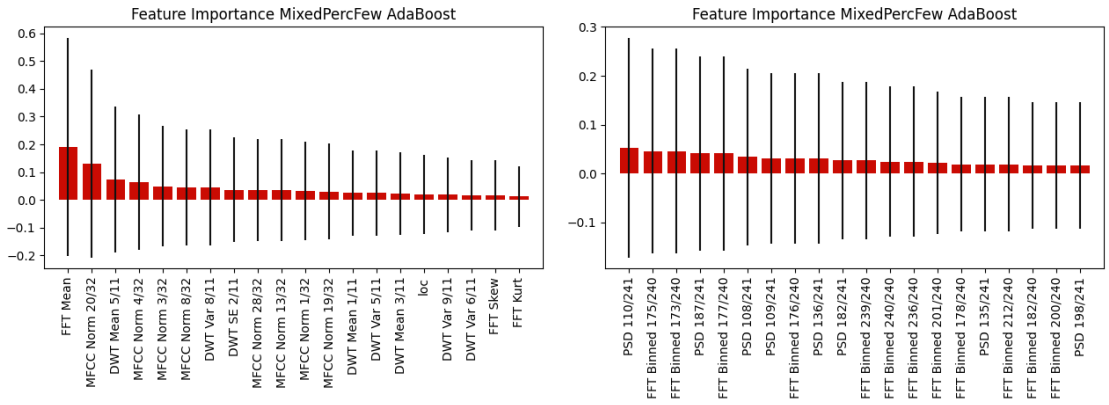


Figure 5.11. Feature Selected by AdaBoost in two different features set: *Essential* on the left and *Complete* on the right, the importance of the features is more distributed and turbo is not considered. It is relevant to note that "turbo" has little relevance in the feature set

Gradient Boosting and XGBoost

Because both AdaBoost and CatBoost present some difficulties in selecting a reasonable importance to the features, more standard Gradient Boosting Machine and its evolution XGBoost can come as a definitive solution. Let's start looking at the performance of Gradient Boosting applied (after Univariate Selection) on the *Essential* feature set with the help of Figure 5.12. By observing the features importance, we can clearly notice a good balance and a good mix. It considers the mean of the FFT arounf 35%, then turbo with 25% and subsequently the values of normalized vector of MFCC. After that, some FFT higher moments (FFT Skewness and Kurtosis) are considered alongside Variance (SE and Var) of the Wavelet transforms. Results are higher than expectations and outperformed the ones obtained for turbo, resulting in a robust process. Area under the curve of ROC curve is obtaining 0.99 in both micro and macro averages. Area under Precision Recall Curve goes even further, reaching 0.994 when micro averaged. As explained in Section 3.4.11, this performance was obtained at a per-chunk level. But when we have to classify an engine, all chunks must be merged together to obtain a final prediction. Each chunk votes for the class it was attributed to, so that the parent sample gets the class of the majority of its sub-chunks. This is what we call *Reconstruction of the Samples*. We can see the performance resulting from this procedure in Figure 5.13. The performance improved even further, reaching a ROC-AUC of 1.00.

this number is rounded, but the meaning is that it goes beyond 0.995. Precision Recall Curve has an area under it of 0.997. What may be even more interesting is the Confusion Matrix, where we can see that it reached perfection in classifying the Diesel samples. It means that if AI predicts "Diesel" we are sure that it is a Diesel indeed. If insted it predicts "Gasoline", this trust reduces to 96%. Note that this and the other values are higher than the one obtained before reconstructing the sample, proving the effectiveness of this procedure for our goal. XGBoost,

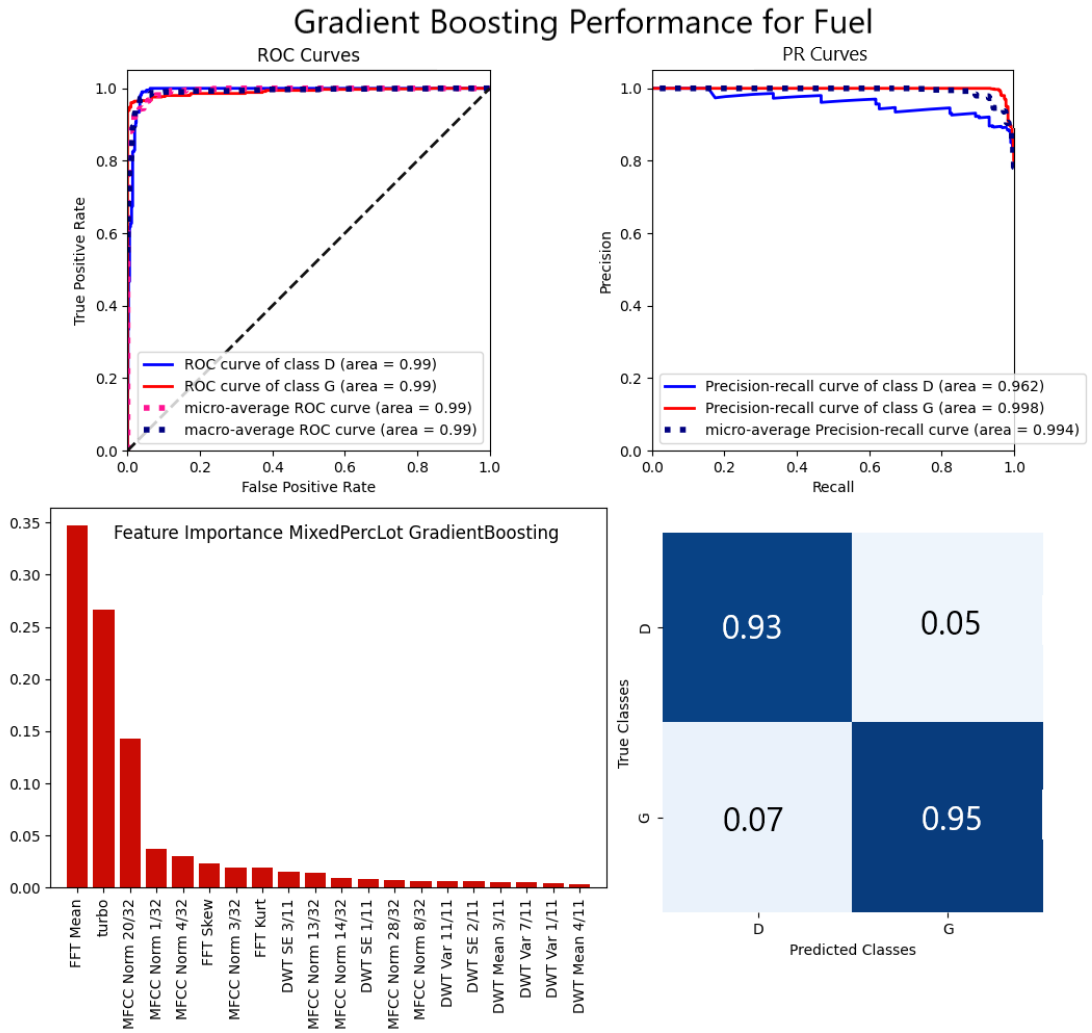


Figure 5.12. Report on Gradient Boosting for *Essential*: Performance for Fuel at chunk level. We can see a good mix of features importance, as well as high values for ROC-AUC (0.99), PR-AUC (0.994) and the values inside the diagonal of the confusion matrix

on the the other hand, was suitable for processing more features, obtaining good results in the *Complete* Feature set. Even though the performance were not as

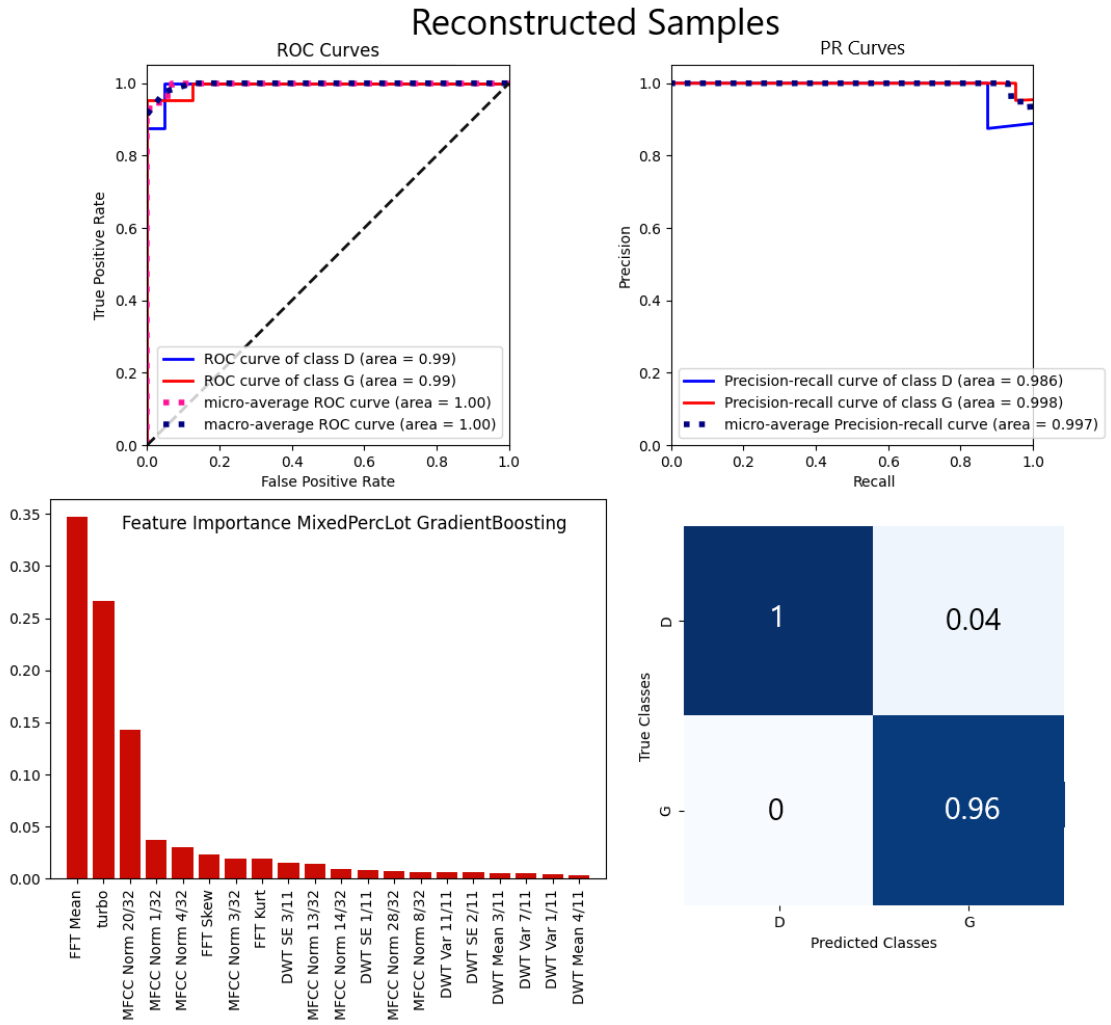


Figure 5.13. Report on Gradient Boosting Reconstructed for *Essential*. The reconstruction of the sample from the different chunks voting for their class improves the results. ROC-AUC reaches (rounded) 1.00 and PR-AUC 0.997. Confusion Matrix values also improved, reaching perfect classification for Diesel engines and 96% for Gasoline

stellar as the ones obtained by Gradient Boosting on the *Essential* feature set, it is interesting to observe to which features it gives the maximum amount of importance. As we can see in Figure 5.14, most of the importance was given to high values in the FFT Binned vector (high frequencies). One low frequency was also judged as relevant, ranked fourth. Turbo feature is also in the top five, and plays its role in classifying the fuel type. Afterwards MFCC vector is considered. This is important as a proof, because two different algorithms trained on different feature sets (XGBoost on *Complete* and Gradient Boosting on *Essential*) consider as

"important" the same aspects of the problem. It means that the features extracted have intrinsic value and are meaningful.

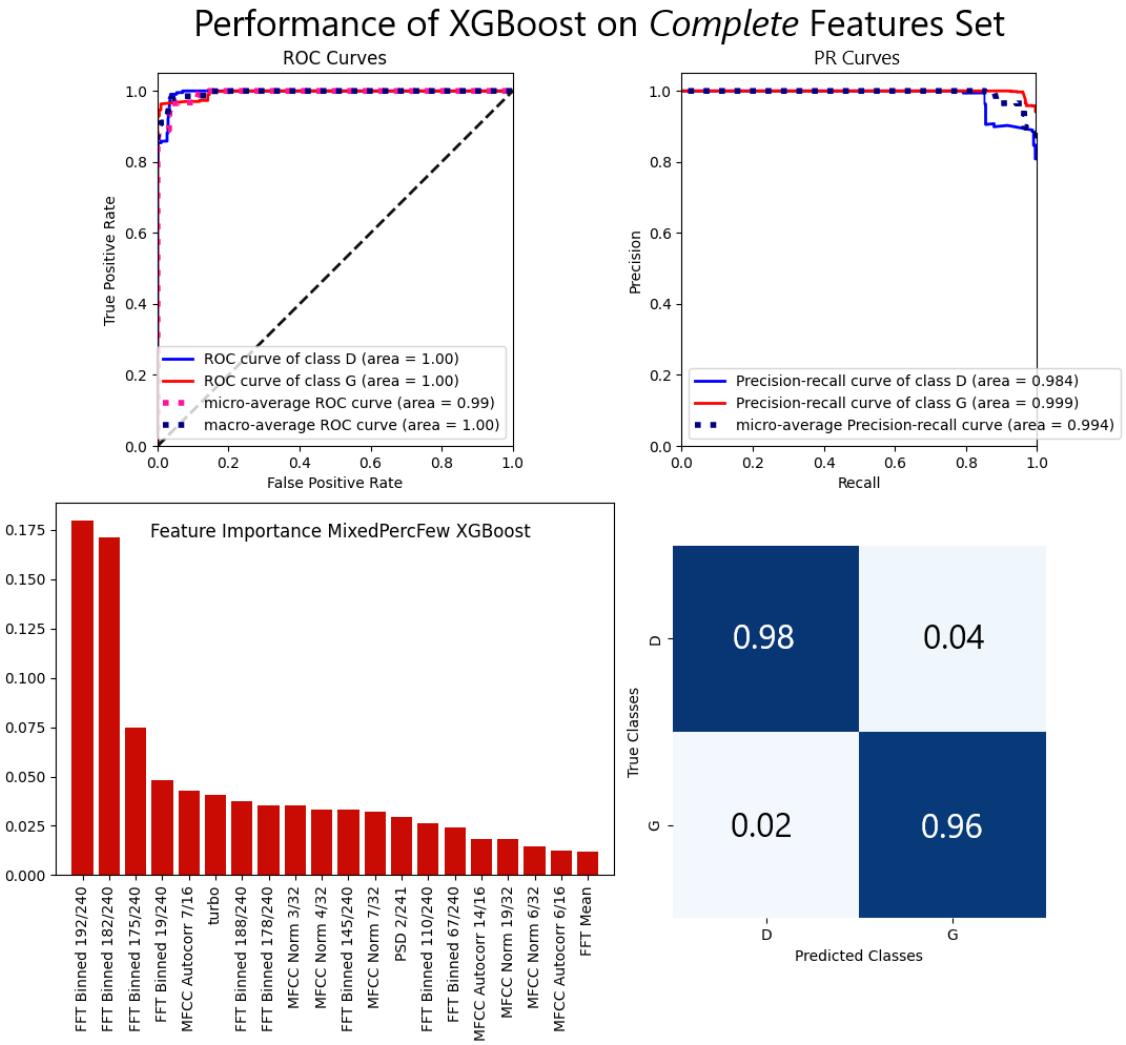


Figure 5.14. Report on XGBoost for *Complete Set*. Performance comparable to Gradient Boosting on *Essential Set*

Final decision and Hyper-parameters

Performance are similar, but the process to generate the feature set and the running time is different. *Essential* Set is faster to extract and have a smaller impact on the memory usage. Furthermore, as shown in Table 5.2.2 the time needed to fit the classifier is significantly higher for the *Complete* Feature Set, even if XGBoost is used, that is supposed to run faster than standard Grandient Boosting Machines.

Classifier	**Dimensionality Reducer**	**Feature Set**	**Fit Time**
XGBoost	Univariate Selection to 13%	*Complete*	3 min 11 sec
Gradient Boosting	Univariate Selection to 46%	*Essential*	0 min 11 sec

Table 5.2. Fitting Time comparison between XGBoost and Gradient Boosting

Following the previous consideration, I decided to select Gradient Boosting as final Classifier for Fuel, and the feature set that does not hold *FTT*, *PSD* and *MFCC Autocorr* vectors inside the feature space. Only further statistics on *FFT* are kept. In the following table, I provide the list of hyper-parameter set to GBM Classifier that resulted best from the random grid search.

Hyperparameter	**Value**
max_depth	6
min_impurity_decrease	0.43
n_estimators	20
Univariate Feature Selection	**Parameters' Value**
score_func	mutual_info_classif
mode	percentile
param	46
Fitting Time	**11 sec**

Table 5.3. Hyper-parameters of Gradient Boosting

5.3 Target Label: Cylinder Amount

The number of cylinders is an important characteristic that strongly influence the way an engine sounds. As already anticipated, this classification problem have several issues:

- **Multi-Class classification.** The more classes are describing the problem, the more difficult it is for the algorithm to perform well. During training, the procedure for the algorithm is more complex, and during testing there are simply more classes to choose from.
- **Strongly unbalanced classes.** Some classes (4 cylinders) are much more represented than others (3, 5 and 8 cylinders). 5 cylinders class is poorly represented, and there are not enough samples for both training and testing phases, so I decided to discard it. 3 and 8 classes are strongly influencing the results, as algorithms struggle with those classes. For this reason, I show in the following section different types of results obtained when filtering the database for a subset of classes.

5.3.1 Confusion Matrices and Performance Scores

If only classes 4 and 6 are considered (the ones where enough data are available), several algorithms performed with a perfect confusion matrix. Boosting algorithms are here better on average. The different confusion matrices are shown in Figure 5.15, without the intention of highlighting details, but just to show the dominance of the diagonals, that in some cases are perfect.

This is already a satisfactory result, because those two classes (4 and 6 cylinders) represent 91% of the cars of the dataset. However, it may be interesting to explore a more exhaustive solution. To counteract the difficulties of a complete model with very small classes, I tried to infer some useful information on the samples by exploring the dataset further. I observed that the size and number of cylinders of a sample engine strongly vary depending on the country where the recording was made. In my case, only samples from Europe (mostly Italy) and US are collected. By observing more carefully, I noticed that no 3 cylinders engine was recorded in the US samples, and no 8 cylinders in the Italian sample. Furthermore, we can suppose that an Italian user with a V8 car knows that its engine has 8 cylinders. Those are in fact niche product for enthusiast drivers. For this reason, I developed three different models, presented in the following sections:

1. European model
2. US model
3. General (weaker) model

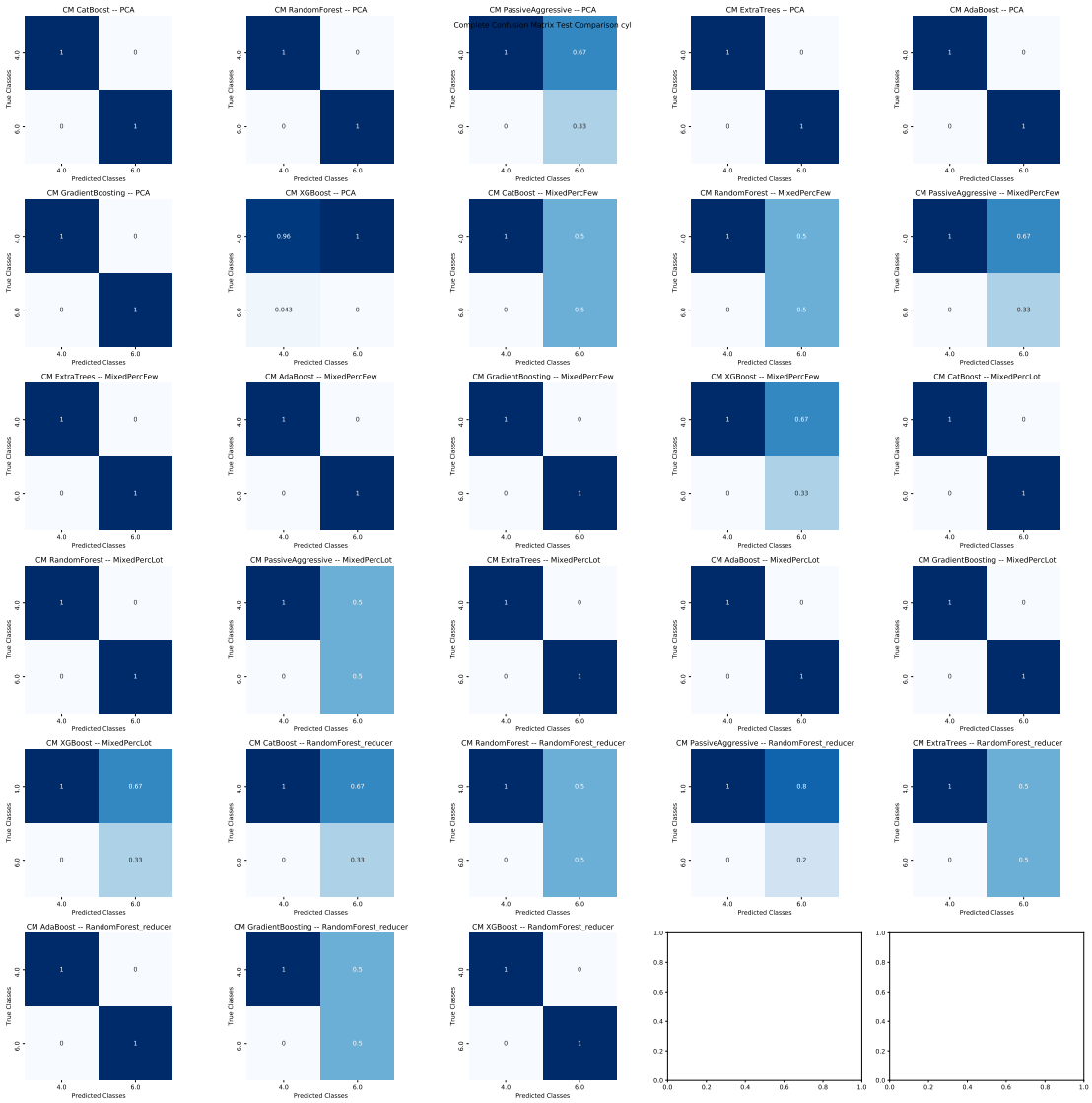


Figure 5.15. Comparison Confusion Matrices with 4 and 6 cylinders (91% of the cars, recalling Figure 2.12). This image is not intended to highlight details, but have a view on the dominance of the diagonals, that in some cases are perfect

5.3.2 European Model

European Model is a solution to the problem that considers only samples belonging to classes 3, 4 and 6 cylinders, accounting for 97% of the cars in the database, and 100% of the engines recorded in Europe. The best model, in this case, is Random Forest Classifier used together with Univariate Features Selection. As we can see in Figure 5.16, Confusion Matrix features a dominant diagonal, and the misclassified samples are considered to be 4 cylinders. It means that if the Random Forest

outputs 4 cylinders, in around 12% of the cases it is actually a 6 cylinder. On the other hand, if a 4 cylinder is tested, in 100% cases it wil be recognized. This precision on the 4 cylinders class is highlighted by the value of the are under the Precision Recall Curve of 0.939. Average Precision Recall Curve features an area underneath of 0.903, showing some weakness in the 3 cylinders class. Depending on the type of averaging, ROC-AUC goes from 0.83 and 0.95, with a constant value across classes. Features considered important are mainly in the normalized vector of *MFCC*, both in low and high values. Kurtosis of the Wavelet transform is very present in the top 20 feature values.

5.3.3 US Model

8 cylinders are more difficult to recognize, and this is especially evident because of the general lower performance of the models. After several fine-tuning activities, the best results are obtained by ExtraTrees and shown in Figure 5.17. In this case, we cannot see the importance that the classifier gave to the single features, as the reducer employed in this case (PCA) linearly combines features. For this reason, ExtraTrees gave importance to some of the linear combination of the original features, which do not have a physical meaning in most cases. Performance were perfect in 8 and 6 cylinders classes for what the normalized confusion matrix shows. It means that if AI predicts, 6 or 8, this result is relatively sure. On the other hand, we can see that all misclassified samples are predicted as 4 cylinders, lowering the area under the curve of those classes. This impact is more evident on 8 cylinders class. This means that if we provide AI with a 4 cylinders sample, it will recognize it with good confidence.

Even though improvements would be possible, more data would be needed for this purpose, as only 3% of the dataset belongs to class 8 cylinders.

5.3.4 General Model

Figure 5.16 shows the report of the Random Forest Classifier applied on all classes available in the dataset. As the performance of this general model is considerably worse than the ones obtained in the previous sections (EU model and US model, I suggest to use those when possible. One could imagine an iterative process here as well or consider to change strategy and apply a regression model. MFCC plays a crucial role in the features, as it is ranked several times in the first 20 important features. FFT Mean and Kurtosis are also considered to be relevant, ranking 5^{th} and 6^{th} respectively. The confusion matrix is not diagonal dominated anymore, whereas the average ROC-AUC scores between 0.82 and 0.93, depending on the average strategy. Precision on class 6 is influencing the results, leading to an average area under the precision recall curve of 0.86

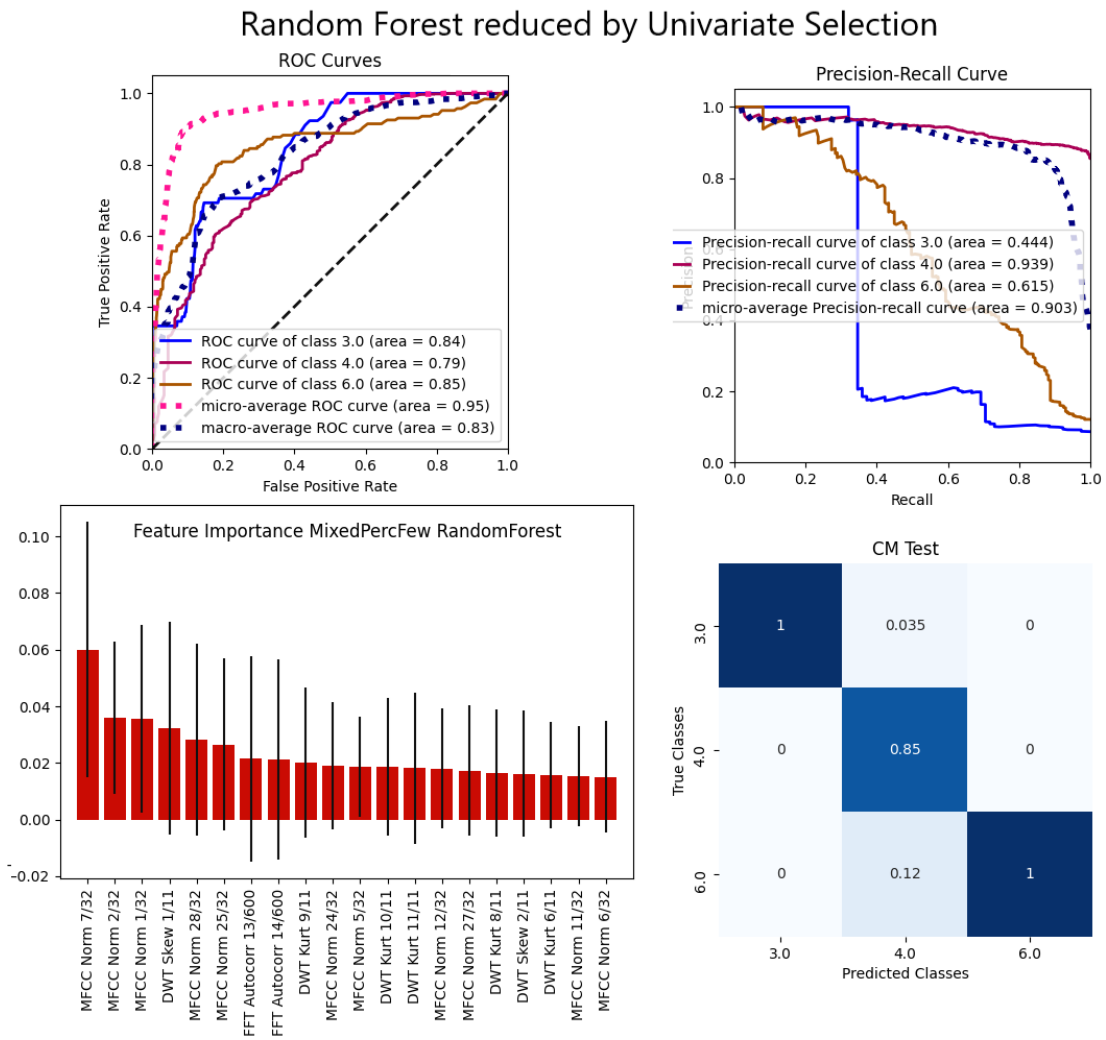


Figure 5.16. Report on Random Forest for **European Model**. Confusion Matrix features a dominant diagonal, and the misclassified samples are considered to be 4 cylinders. Depending on the type of averaging, ROC-AUC goes from 0.83 and 0.95. Precision Recall Curve features an area underneath of 0.903, showing some weackness in the 3 cylinders class

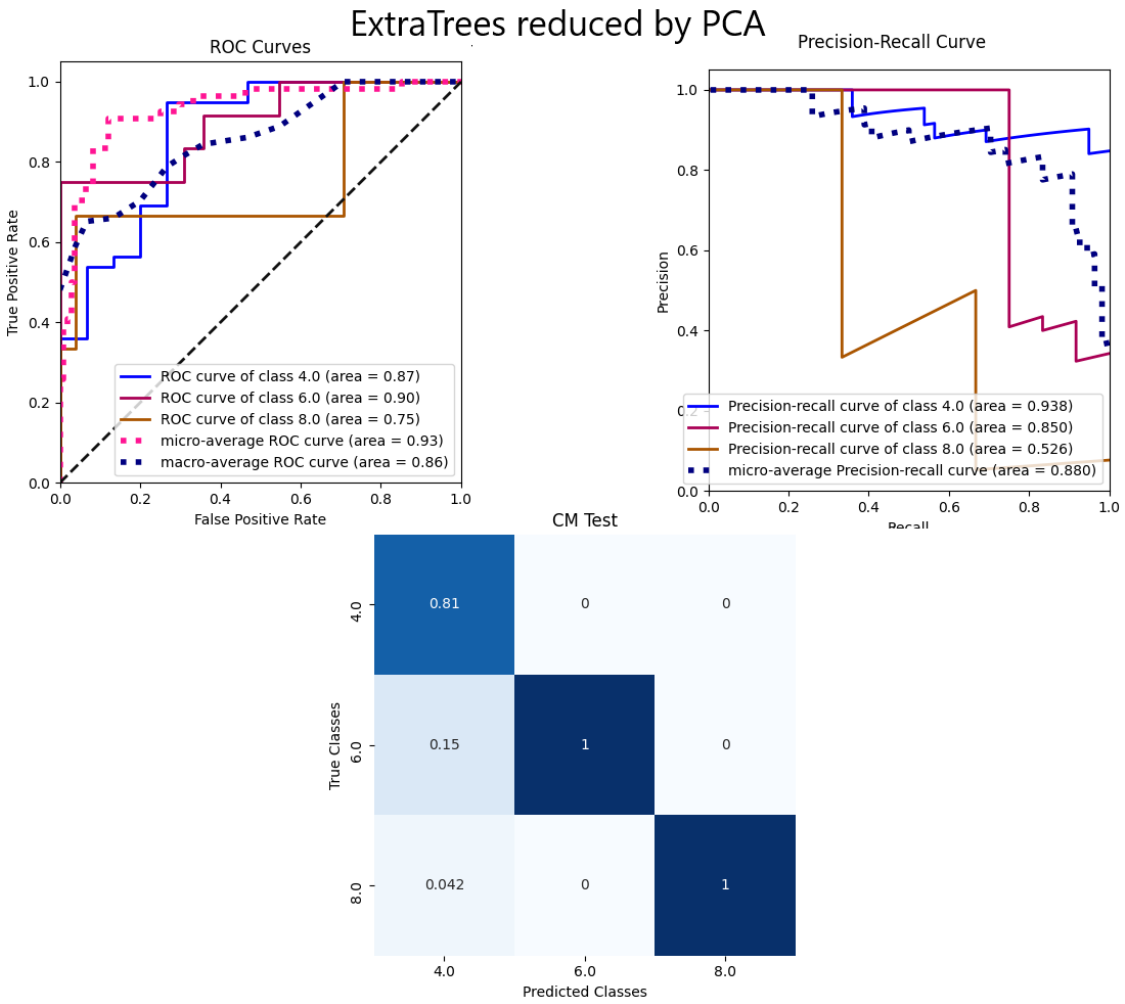


Figure 5.17. Report on ExtraTrees for **US Model**. Confusion Matrix features a dominant diagonal, and features importance is not related to original features as PCA combines them. Depending on the type of averaging, ROC-AUC goes from 0.86 and 0.93. Precision Recall Curve features an area underneath of 0.88, showing some weakness in the 8 cylinders class

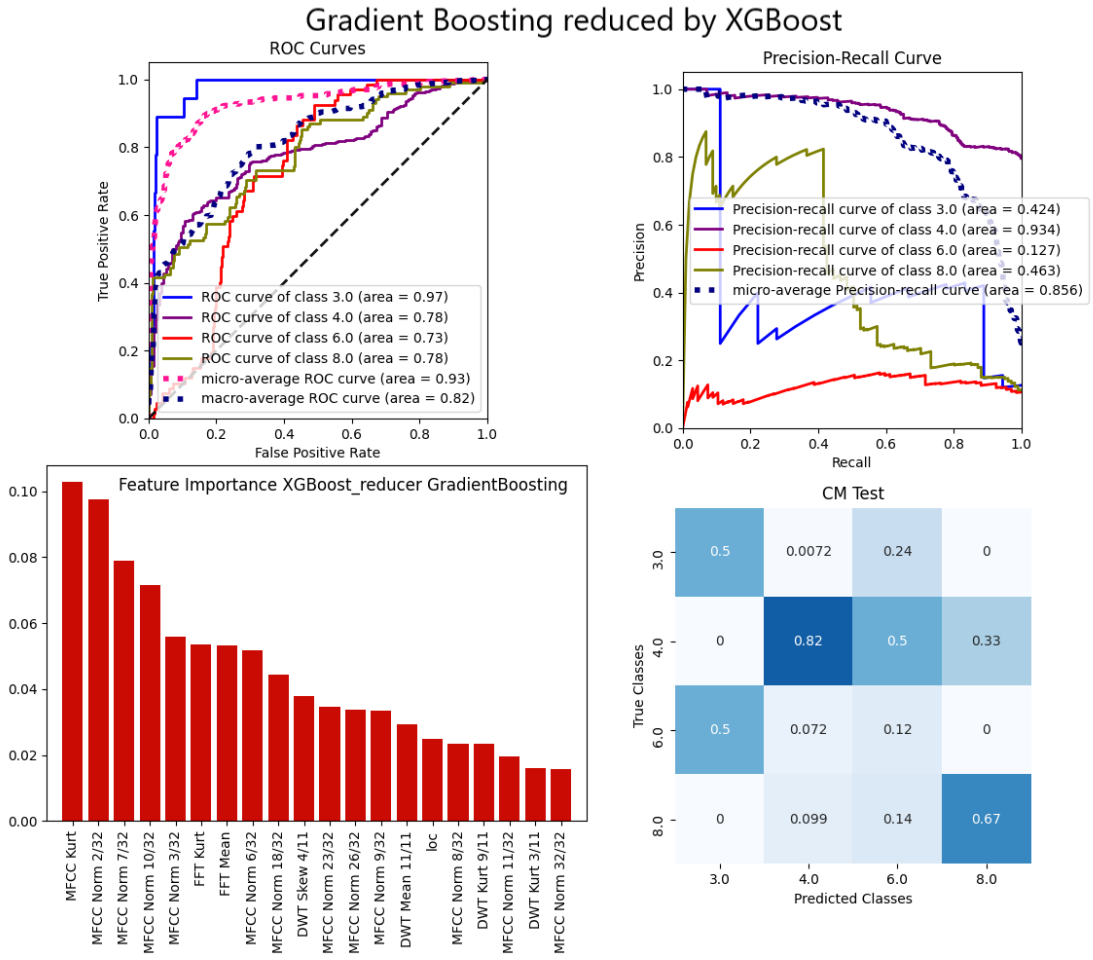


Figure 5.18. Report on Gradient Boosting for **General Model**. With 4 classes, the area under the curves reaches a 93% for ROC and 86% for PR, because of the dominance of 4 cylinders class driving the metrics. However, when looking at the confusing matrix, it is clear that performance is worse than the previous area-specific models. For this reason, it is suggested to use those models when possible.

5.4 Conclusions and Next Steps

With this Thesis, I showed the opportunity to develop an iterative process to recognize the characteristics of the engine (Turbo, Fuel type and finally Cylinder amount). By exploiting the correlation among the labels, it allows to achieve higher performances. Results obtained are robust enough to be applied in a context identification phase for vibroacoustic diagnostics purposes. In the following Table 5.4, I recall the major numerical results obtained with Thesis broken down label.

Label	**Classifier**	**Reducer**	**ROC***	**PR***
Turbo	ExtraTrees	Random Forest	0.86	0.857
Fuel	GBM	Univariate Selection	>0.995	0.997
Cyl (4 and 6)	Boosting (all)	Univariate Selection	1	1
Cyl (EU model)	Random Forest	Univariate Selection	0.95	0.9
Cyl (US model)	ExtraTrees	PCA	0.93	0.88
Cyl (Complete)	GBM	XGBoost	0.93	0.856

Table 5.4. Results Table.
$*$: *micro-averaged Area Under the Curves in multi-class problems*

Furthermore, it is important to stress the fact that this framework is working automatically from the raw sounds to the automatic production of the results charts, meaning that it could be used by other researchers to obtain results in this or other fields where an acoustic diagnostics is employed.
Hereby, I recall some of the next steps that could bring this framework to achieve even more powerful results:

- Explore regression strategies
- Automatize audio conversion process
- Build app for interfacing with the user
- Connect to diagnostics software
- Implement context identification procedure outlined in Chapter 1
- Unleash own imagination for innovative use-cases

The world of smartphone based diagnostics has the potential to become a relevant technology in every aspect of our lives in the coming years. I am proud to have given my contribution to the development of such an ambitious goal: making AI listen to the surroundings, learn to predict their status, and ultimately suggest procedures to improve the world.

Bibliography

[1] Irman Abdić et al. "Detecting road surface wetness from audio: A deep learning approach". In: *Pattern Recognition (ICPR), 2016 23rd International Conference on*. IEEE. 2016, pp. 3458–3463.

[2] Aceable. *Car Accident Statistics*. URL: https://web.archive.org/web/20190822225917/https://www.aceable.com/safe-driving/car-accident-statistics/.

[3] Christer Ahlström et al. "Using smartphone logging to gain insight about phone use in traffic". In: *Cognition, Technology & Work* (2019), pp. 1–11.

[4] Ali N. Akansu and Richard A. Haddad. *Multiresolution Signal Decomposition: Transforms, Subbands, and Wavelets*. 2nd ed. San Diego: Academic Press, 2001. ISBN: 978-0-12-047141-6.

[5] P Aksamit. "Adaptive Approach to Acoustic Car Driving Detection in Mobile Devices". In: *Acta Physica Polonica A* 124.3 (2013), pp. 381–383.

[6] Waleed Aleadelat, Cameron HG Wright, and Khaled Ksaibati. "Estimation of gravel roads ride quality through an android-based smartphone". In: *Transportation Research Record* 2672.40 (2018), pp. 14–21.

[7] European Automobile Manufacturers Association. *Average Vehicle Age*. URL: https://web.archive.org/web/20200416221830/https://www.acea.be/statistics/tag/category/average-vehicle-age.

[8] Andrzej BAKOWSKI et al. "Vibroacoustic Real Time Fuel Classification in Diesel Engine". In: *Archives of Acoustics* 43.3 (2018), pp. 385–395.

[9] Bhumika J Barai and Ghanshyam Kamdi. "Mechanical Condition Determination of Vehicle and Traffic Density Estimation Using Acoustic Signals". In: *2014 Fourth International Conference on Communication Systems and Network Technologies*. IEEE. 2014, pp. 259–264.

[10] Christopher Barlow et al. "Using Immersive Audio and Vibration to Enhance Remote Diagnosis of Mechanical Failure in Uncrewed Vessels". In: *Audio Engineering Society Conference: 2019 AES International Conference on Immersive and Interactive Audio*. Mar. 2019. URL: http://www.aes.org/e-lib/browse.cfm?elib=20428.

[11] L. Bedogni, O. Bujor, and M. Levorato. "Texting and Driving Recognition Exploiting Subsequent Turns Leveraging Smartphone Sensors". In: *2019 IEEE 20th International Symposium on "A World of Wireless, Mobile and Multimedia Networks" (WoWMoM)*. June 2019, pp. 1–9. DOI: 10.1109/WoWMoM.2019.8793032.

[12] Alex Beresnev and Max Beresnev. "Vibroacoustic Method of IC Engine Diagnostics". In: *SAE International Journal of Engines* 7.1 (2014), pp. 1–5. ISSN: 19463936, 19463944. URL: http://www.jstor.org/stable/26277743.

[13] Piotr Białkowski and Bogusław Krężel. "Early detection of cracks in rear suspension beam with the use of time domain estimates of vibration during the fatigue testing". In: *Diagnostyka* 16 (2015).

[14] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Information Science and Statistics. New York: Springer, 2006. ISBN: 978-0-387-31073-2.

[15] Huy Bui. *Decision Tree Fundamentals*. en. Mar. 2020. URL: https://towardsdatascience.com/decision-tree-fundamentals-388f57a60d2a.

[16] Lars Buitinck et al. "API Design for Machine Learning Software: Experiences from the Scikit-Learn Project". In: *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*. 2013, pp. 108–122.

[17] G. Castignani et al. "Smartphone-Based Adaptive Driving Maneuver Detection: A Large-Scale Evaluation Study". In: *IEEE Transactions on Intelligent Transportation Systems* 18.9 (Sept. 2017), pp. 2330–2339. ISSN: 1558-0016. DOI: 10.1109/TITS.2016.2646760.

[18] *CatBoost - State-of-the-Art Open-Source Gradient Boosting Library with Categorical Features Support*. en. URL: https://catboost.ai.

[19] Czesław Cempel. "Vibroacoustical diagnostics of machinery: an outline". In: *Mechanical Systems and Signal Processing* 2.2 (1988), pp. 135–151.

[20] Afroz Chakure. *Decision Tree Classification*. en. June 2020. URL: https://towardsdatascience.com/decision-tree-classification-de64fc4d5aac.

[21] Dongyao Chen et al. "Invisible sensing of vehicle steering with smartphones". In: *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*. ACM. 2015, pp. 1–13.

[22] Jim Cherian. "Mobile crowdsensing applications for intelligent parking and mobility". PhD thesis. 2019.

[23] Hoong-Chor Chin, Xingting Pang, and Zhaoxia Wang. "Analysis of Bus Ride Comfort Using Smartphone Sensor Data". In: ().

[24] Ship-Bin Chiou and Jia-Yush Yen. "A Survey of Suspension Component Specifications and Vehicle Vibration Measurements in an Operational Railway System". In: *Journal of Vibration Engineering & Technologies* (2019), pp. 1–17.

[25] T. Choudhury et al. "The Mobile Sensing Platform: An Embedded Activity Recognition System". In: *IEEE Pervasive Computing* 7.2 (Apr. 2008), pp. 32–41. ISSN: 1558-2590. DOI: 10.1109/MPRV.2008.39.

[26] A. Chowdhury et al. "Smartphone based sensing enables automated vehicle prognosis". In: *2015 9th International Conference on Sensing Technology (ICST)*. Dec. 2015, pp. 452–455. DOI: 10.1109/ICSensT.2015.7438441.

[27] Symeon E. Christodoulou, Georgios M. Hadjidemetriou, and Charalambos Kyriakou. "Pavement Defects Detection and Classification Using Smartphone-Based Vibration and Video Signals". In: *Advanced Computing Strategies for Engineering*. Ed. by Ian F. C. Smith and Bernd Domer. Cham: Springer International Publishing, 2018, pp. 125–138. ISBN: 978-3-319-91635-4.

[28] Gunjan Chugh, Divya Bansal, and Sanjeev Sofat. "Road condition detection using smartphone sensors: A survey". In: *International Journal of Electronic and Electrical Engineering* 7.6 (2014), pp. 595–602.

[29] Zbigniew Dabrowski and Jacek Dziurdź. "Simultaneous analysis of noise and vibration of machines in vibroacoustic diagnostics". In: *Archives of Acoustics* 41.4 (2016), pp. 783–789.

[30] Zbigniew Dabrowski and Maciej Zawisza. "Investigations of the Vibroacoustic Signals Sensitivity to Mechanical Defects Not Recognised by the OBD System in Diesel Engines". In: *Solid State Phenomena* 180 (Nov. 2011), pp. 194–199. DOI: 10.4028/www.scientific.net/SSP.180.194.

[31] Jiangpeng Dai et al. "Mobile phone based drunk driving detection". In: *Pervasive Computing Technologies for Healthcare (PervasiveHealth), 2010 4th International Conference on-NO PERMISSIONS*. Mar. 2010, pp. 1–8. DOI: 10.4108/ICST.PERVASIVEHEALTH2010.8901. URL: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5482295.

[32] A. Dasgupta, D. Rahman, and A. Routray. "A Smartphone-Based Drowsiness Detection and Warning System for Automotive Drivers". In: *IEEE Transactions on Intelligent Transportation Systems* 20.11 (Nov. 2019). ISSN: 1558-0016. DOI: 10.1109/TITS.2018.2879609.

[33] *Decision Tree Classification. A Decision Tree Is a Simple... — by Afroz Chakure — Towards Data Science*. URL: https://towardsdatascience.com/decision-tree-classification-de64fc4d5aac.

[34] Simone Delvecchio, Paolo Bonfiglio, and Francesco Pompoli. "Vibro-acoustic condition monitoring of Internal Combustion Engines: A critical review of existing techniques". In: *Mechanical Systems and Signal Processing* 99 (2018), pp. 661–683.

[35] Li Deng. "Deep Learning: Methods and Applications". en. In: *Foundations and Trends® in Signal Processing* 7.3-4 (2014), pp. 197–387. ISSN: 1932-8346, 1932-8354. DOI: 10.1561/2000000039.

[36] Benjamin Elizalde et al. "NELS-never-ending learner of sounds". In: *arXiv preprint arXiv:1801.05544* (2018).

[37] J. Engelbrecht et al. "Survey of smartphone-based sensing in vehicles for intelligent transportation system applications". In: *IET Intelligent Transport Systems* 9.10 (2015), pp. 924–935. ISSN: 1751-9578. DOI: 10.1049/iet-its.2014.0248.

[38] *Ensemble Learning — Ensemble Techniques*. June 2018.

[39] H. Eren et al. "Estimating driving behavior by a smartphone". In: *2012 IEEE Intelligent Vehicles Symposium*. June 2012, pp. 234–239. DOI: 10.1109/IVS.2012.6232298.

[40] Jakob Eriksson et al. "The pothole patrol: using a mobile sensor network for road surface monitoring". In: *Proceedings of the 6th international conference on Mobile systems, applications, and services*. 2008, pp. 29–39.

[41] *Extending Machine Learning Algorithms – AdaBoost Classifier — Packtpub.Com*. Dec. 2017.

[42] *Feature Selection Using Random Forest — by Akash Dubey — Towards Data Science*. URL: https://towardsdatascience.com/feature-selection-using-random-forest-26d7b747597f.

[43] Tomasz Figlus et al. "Condition monitoring of engine timing system by using wavelet packet decomposition of a acoustic signal". In: *Journal of mechanical science and technology* 28.5 (2014), pp. 1663–1671.

[44] Francis Galton. "Vox Populi". en. In: *Nature* 75.1949 (Mar. 1907), pp. 450–451. ISSN: 0028-0836, 1476-4687. DOI: 10.1038/075450a0.

[45] Pierre Geurts, Damien Ernst, and Louis Wehenkel. "Extremely Randomized Trees". en. In: *Machine Learning* 63.1 (Apr. 2006), pp. 3–42. ISSN: 0885-6125, 1573-0565. DOI: 10.1007/s10994-006-6226-1.

[46] Antonio Ginart et al. "Smart phone machinery vibration measurement". In: *2011 Aerospace Conference*. IEEE. 2011, pp. 1–7.

[47] Taesik Gong et al. "Knocker: Vibroacoustic-based Object Recognition with Smartphones". In: *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 3.3 (2019), pp. 1–21.

[48] Brandon Gozick. "A driver, vehicle and road safety system using smartphones". MA thesis. University of North Texas, 2012.

[49] Marco Grossi. "A sensor-centric survey on the development of smartphone measurement and sensing systems". In: *Measurement* 135 (2019), pp. 572–592.

[50] Marco Grossi. "A sensor-centric survey on the development of smartphone measurement and sensing systems". In: *Measurement* 135 (2019), pp. 572–592. ISSN: 0263-2241. DOI: https://doi.org/10.1016/j.measurement.2018.12.014. URL: http://www.sciencedirect.com/science/article/pii/S0263224118311576.

[51] Giuseppe Guido et al. "Estimation of safety performance measures from smartphone sensors". In: *Procedia-Social and Behavioral Sciences* 54 (2012), pp. 1095–1103.

[52] Gino Iannace, Giuseppe Ciaburro, and Amelia Trematerra. "Fault Diagnosis for UAV Blades Using Artificial Neural Network". In: *Robotics* 8.3 (2019), p. 59.

[53] Derick A Johnson and Mohan M Trivedi. "Driving style recognition using a smartphone as a sensor platform". In: *2011 14th International IEEE Conference on Intelligent Transportation Systems (ITSC)*. IEEE. 2011, pp. 1609–1615.

[54] Manu Joseph. *CatBoost*. en. Mar. 2020. URL: https://towardsdatascience.com/catboost-d1f1366aca34.

[55] Manu Joseph. *LightGBM*. en. Feb. 2020. URL: https://towardsdatascience.com/lightgbm-800340f21415.

[56] Manu Joseph. *The Gradient Boosters V: CatBoost*. en. Feb. 2020. URL: https://deep-and-shallow.com/2020/02/29/the-gradient-boosters-v-catboost/.

[57] Stratis Kanarachos, Stavros-Richard G. Christopoulos, and Alexander Chroneos. "Smartphones as an integrated platform for monitoring driver behaviour: The role of sensor fusion and connectivity". In: *Transportation Research Part C: Emerging Technologies* 95 (2018), pp. 867–882. ISSN: 0968-090X. DOI: https://doi.org/10.1016/j.trc.2018.03.023. URL: http://www.sciencedirect.com/science/article/pii/S0968090X18303954.

[58] Stratis Kanarachos, Jino Mathew, and Michael E. Fitzpatrick. "Instantaneous vehicle fuel consumption estimation using smartphones and recurrent neural networks". In: *Expert Systems with Applications* 120 (2019), pp. 436–447. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2018.12.006. URL: http://www.sciencedirect.com/science/article/pii/S0957417418307681.

[59] Adil Mehmood Khan et al. "Activity Recognition on Smartphones via Sensor-Fusion and KDA-Based SVMs". In: *International Journal of Distributed Sensor Networks* 10.5 (2014), p. 503291. DOI: 10.1155/2014/503291. eprint: https://doi.org/10.1155/2014/503291. URL: https://doi.org/10.1155/2014/503291.

[60] Iwona Komorska. "A Vibroacoustic diagnostic system as an element improving road transport safety". In: *International journal of occupational safety and ergonomics* 19.3 (2013), pp. 371–385.

[61] Elżbieta Kubera et al. "Audio-Based Speed Change Classification for Vehicles". In: *New Frontiers in Mining Complex Patterns*. Ed. by Annalisa Appice et al. Cham: Springer International Publishing, 2017, pp. 54–68. ISBN: 978-3-319-61461-8.

[62] Jennifer R Kwapisz, Gary M Weiss, and Samuel A Moore. "Activity recognition using cell phone accelerometers". In: *ACM SigKDD Explorations Newsletter* 12.2 (2011), pp. 74–82.

[63] Francisco Laport-López et al. "A review of mobile sensing systems, applications, and opportunities". In: *Knowledge and Information Systems* (2019), pp. 1–30.

[64] Gierad Laput et al. "Ubicoustics: Plug-and-play acoustic activity recognition". In: *Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology*. 2018, pp. 213–224.

[65] Igor Lashkov and Alexey Kashevnik. "Smartphone-Based Intelligent Driver Assistant: Context Model and Dangerous State Recognition Scheme". In: *Intelligent Systems and Applications*. Ed. by Yaxin Bi, Rahul Bhatia, and Supriya Kapoor. Cham: Springer International Publishing, 2020, pp. 152–165. ISBN: 978-3-030-29513-4.

[66] Gregory Lee et al. "PyWavelets: A Python Package for Wavelet Analysis". In: *Journal of Open Source Software* 4.36 (Apr. 2019), p. 1237. ISSN: 2475-9066. DOI: 10.21105/joss.01237.

[67] Namkyoung Lee et al. "A Comparative Study of Deep Learning-Based Diagnostics for Automotive Safety Components Using a Raspberry Pi". In: *2019 IEEE International Conference on Prognostics and Health Management (ICPHM)*. IEEE. 2019, pp. 1–7.

[68] Anders Lehmann and Allan Gross. "Towards vehicle emission estimation from smartphone sensors". In: *2017 18th IEEE International Conference on Mobile Data Management (MDM)*. IEEE. 2017, pp. 154–163.

[69] Hong Lu et al. "The Jigsaw Continuous Sensing Engine for Mobile Phone Applications". In: *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems*. SenSys '10. Zürich, Switzerland: Association for Computing Machinery, 2010, pp. 71–84. ISBN: 9781450303446. DOI: 10.1145/1869983.1869992. URL: https://doi.org/10.1145/1869983.1869992.

[70] Paul Lukowicz et al. "Recognizing Workshop Activity Using Body Worn Microphones and Accelerometers". In: *Pervasive Computing*. Ed. by Alois Ferscha and Friedemann Mattern. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 18–32. ISBN: 978-3-540-24646-6.

[71] *Machine Learning Textbook*. URL: http://www.cs.cmu.edu/~tom/mlbook.html.

[72] Irina Makarova et al. "Improvement of the Vehicle's Onboard Diagnostic System by Using the Vibro-Diagnostics Method". In: *2018 International Conference on Diagnostics in Electrical Engineering (Diagnostika)*. IEEE. 2018, pp. 1–4.

[73] Silvia Marelli et al. *Incipient Surge Detection in Automotive Turbocharger Compressors*. Tech. rep. SAE Technical Paper, 2019.

[74] Gabriel Matuszczyk and Rasmus Åberg. "Smartphone based automatic incident detection algorithm and crash notification system for all-terrain vehicle drivers". MA thesis. Chalmers University of Technology, 2016.

[75] Matthias Auf der Mauer et al. "Applying Sound-Based Analysis at Porsche Production: Towards Predictive Maintenance of Production Machines Using Deep Learning and Internet-of-Things Technology". In: *Digitalization Cases: How Organizations Rethink Their Business for the Digital Age*. Ed. by Nils Urbach and Maximilian Röglinger. Cham: Springer International Publishing, 2019, pp. 79–97. ISBN: 978-3-319-95273-4. DOI: 10.1007/978-3-319-95273-4_5. URL: https://doi.org/10.1007/978-3-319-95273-4_5.

[76] Brian McFee et al. *Librosa/Librosa: 0.8.0*. Zenodo. July 2020. DOI: 10.5281/ZENODO.3955228.

[77] Matthias Mielke and Rainer Brück. "Smartphone application for automatic classification of environmental sound". In: *Proceedings of the 20th International Conference Mixed Design of Integrated Circuits and Systems-MIXDES 2013*. IEEE. 2013, pp. 512–515.

[78] Vishal Morde. *XGBoost Algorithm: Long May She Reign!* en. Apr. 2019. URL: https://towardsdatascience.com/https-medium-com-vishalmorde-xgboost-algorithm-long-she-may-rein-edd9f99be63d.

[79] Max Morrison and Bryan Pardo. "OtoMechanic: Auditory Automobile Diagnostics via Query-by-Example". In: *arXiv preprint arXiv:1911.02073* (2019).

[80] Roy Navea and Edwin Sybingco. "Design and Implementation of an Acoustic-Based Car Engine Fault Diagnostic System in the Android Platform". In: *International Research Conference in Higher Education*. Oct. 2013.

[81] Z. Ouyang et al. "An Ensemble Learning-Based Vehicle Steering Detector Using Smartphones". In: *IEEE Transactions on Intelligent Transportation Systems* (2019), pp. 1–12. ISSN: 1558-0016. DOI: 10.1109/TITS.2019.2909107.

[82] Eleonora Papadimitriou et al. "Analysis of driver behaviour through smartphone data: The case of mobile phone use while driving". In: *Safety Science* 119 (2019), pp. 91–97. ISSN: 0925-7535. DOI: https://doi.org/10.1016/j.ssci.2019.05.059. URL: http://www.sciencedirect.com/science/article/pii/S0925753518300213.

[83] *Passive Aggressive Classifiers*. URL: https://www.geeksforgeeks.org/passive-aggressive-classifiers/.

[84] Geoffroy Peeters. "A Large Set of Audio Features for Sound Description (Similarity and Classification) in the CUIDADO Project". In: (Jan. 2004).

[85] Zhe Peng et al. "Vehicle safety improvement through deep learning and mobile sensing". In: *IEEE network* 32.4 (2018), pp. 28–33.

[86] *Pickle — Python Object Serialization — Python 3.8.5 Documentation*. URL: https://docs.python.org/3/library/pickle.html.

[87] *Quotes about Python*. en. URL: https://www.python.org/about/quotes/.

[88] E. Raviola and F. Fiori. "Real Time Defect Detection of Wheel Bearing by Means of a Wirelessly Connected Microphone". In: *2018 14th Conference on Ph.D. Research in Microelectronics and Electronics (PRIME)*. July 2018, pp. 233–236. DOI: 10.1109/PRIME.2018.8430303.

[89] Joseph Rocca. *Ensemble Methods: Bagging, Boosting and Stacking*. en. May 2019. URL: https://towardsdatascience.com/ensemble-methods-bagging-boosting-and-stacking-c9214a10a205.

[90] Nick Routley. *How the computing power in a smartphone compares to supercomputers past and present*. URL: https://web.archive.org/web/20190726152617/https://www.businessinsider.com/infographic-how-computing-power-has-changed-over-time-2017-11?r=US&IR=T.

[91] Sankar Kumar Roy. "Combustion Event Detection in a Single Cylinder Diesel Engine by Analysis of Sound Signal Recorded by Android Mobile". In: *Advances in Interdisciplinary Engineering*. Ed. by Mukul Kumar, R. K. Pandey, and Vikas Kumar. Singapore: Springer Singapore, 2019, pp. 121–129. ISBN: 978-981-13-6577-5.

[92] Stuart J. Russell, Peter Norvig, and Ernest Davis. *Artificial Intelligence: A Modern Approach*. 3rd ed. Prentice Hall Series in Artificial Intelligence. Upper Saddle River: Prentice Hall, 2010. ISBN: 978-0-13-604259-4.

[93] J. Salamon and J. P. Bello. "Deep Convolutional Neural Networks and Data Augmentation for Environmental Sound Classification". In: *IEEE Signal Processing Letters* 24.3 (Mar. 2017), pp. 279–283. ISSN: 1558-2361. DOI: 10.1109/LSP.2017.2657381.

[94] Claude Sammut and Geoffrey I. Webb, eds. *Encyclopedia of Machine Learning*. New York ; London: Springer, 2010. ISBN: 978-0-387-30768-8 978-0-387-34558-1 978-0-387-30164-8.

[95] Sanjay E Sarma, Stephen Sai-Wung Ho, and Joshua E Siegel. *System and method for providing road condition and congestion monitoring using smart messages*. U.S. Patent 8,566,010. Oct. 2013.

[96] Keerthana Chandrika Sasikumar, CG Siddalingayya, and Rajeswar Kuchimanchi. *Non-Invasive Real Time Error State Detection for Tractors Using Smart Phone Sensors & Machine Learning*. Tech. rep. SAE Technical Paper, 2019.

[97] Wojciech SAWczuk. "Application of vibroacoustic diagnostics to evaluation of wear of friction pads rail brake disc". In: *Eksploatacja i Niezawodność* 18 (2016).

[98] Stefan Sedivy, Lenka Micechova, and Pavel Scheber. "Mechatronics Solutions in Process of Transport Infrastructure Monitoring and Diagnostics". In: *International Conference Mechatronics*. Springer. 2019, pp. 141–148.

[99] Valentino Servizi et al. "Mining User Behaviour from Smartphone data, a literature review". In: *arXiv preprint arXiv:1912.11259* (2019).

[100] Sunil Kumar Sharma and Rakesh Chandmal Sharma. "Pothole Detection and Warning System for Indian Roads". In: *Advances in Interdisciplinary Engineering*. Springer, 2019, pp. 511–519.

[101] Changqing Shen et al. "An automatic and robust features learning method for rotating machinery fault diagnosis based on contractive autoencoder". In: *Engineering Applications of Artificial Intelligence* 76 (2018), pp. 170–184. ISSN: 0952-1976. DOI: https://doi.org/10.1016/j.engappai.2018.09.010. URL: http://www.sciencedirect.com/science/article/pii/S0952197618301969.

[102] J. E. Siegel, D. C. Erb, and S. E. Sarma. "A Survey of the Connected Vehicle Landscape—Architectures, Enabling Technologies, Applications, and Development Areas". In: *IEEE Transactions on Intelligent Transportation Systems* 19.8 (Aug. 2018), pp. 2391–2406. ISSN: 1558-0016. DOI: 10.1109/TITS.2017.2749459.

[103] J. Siegel et al. "Vehicular engine oil service life characterization using On-Board Diagnostic (OBD) sensor data". In: *SENSORS, 2014 IEEE*. Nov. 2014, pp. 1722–1725. DOI: 10.1109/ICSENS.2014.6985355.

[104] Joshua E. Siegel and Umberto Coda. "Surveying Off-Board and Extra-Vehicular Monitoring and Progress Towards Pervasive Diagnostics". In: (July 2020). arXiv: 2007.03759.

[105] Joshua E. Siegel, Yongbin Sun, and Sanjay Sarma. "Automotive Diagnostics as a Service: An Artificially Intelligent Mobile Application for Tire Condition Assessment". In: *Artificial Intelligence and Mobile Services – AIMS 2018*. Ed. by Marco Aiello et al. Cham: Springer International Publishing, 2018, pp. 172–184. ISBN: 978-3-319-94361-9.

[106] Joshua E. Siegel et al. "Air filter particulate loading detection using smartphone audio and optimized ensemble classification". In: *Engineering Applications of Artificial Intelligence* 66 (2017), pp. 104–112. ISSN: 0952-1976. DOI: https://doi.org/10.1016/j.engappai.2017.09.015. URL: http://www.sciencedirect.com/science/article/pii/S0952197617302294.

[107] *Smartphone-Based Wheel Imbalance Detection*. Vol. Volume 2: Diagnostics and Detection; Drilling; Dynamics and Control of Wind Energy Systems; Energy Harvesting; Estimation and Identification; Flexible and Smart Structure Control; Fuels Cells/Energy Storage; Human Robot Interaction; HVAC Building Energy Management; Industrial Applications; Intelligent Transportation Systems; Manufacturing; Mechatronics; Modelling and Validation; Motion and Vibration Control Applications. Dynamic Systems and Control Conference. V002T19A002. Oct. 2015. DOI: 10.1115/DSCC2015-9716. eprint: https://asmedigitalcollection.asme.org/DSCC/proceedings-pdf/DSCC2015/57250/V002T19A002/4446383/v002t19a002-dscc2015-9716.pdf. URL: https://doi.org/10.1115/DSCC2015-9716.

[108] Joshua Eric Siegel. "CloudThink and the Avacar: Embedded design to create virtual vehicles for cloud-based informatics, telematics, and infotainment". MA thesis. Massachusetts Institute of Technology, 2013.

[109] Joshua Eric Siegel. *System and method for providing predictive software upgrades*. U.S. Patent 9,086,941. July 2015.

[110] Joshua E Siegel et al. "Real-time deep neural networks for internet-enabled arc-fault detection". In: *Engineering Applications of Artificial Intelligence* 74 (2018), pp. 35–42.

[111] Joshua Siegel et al. "Engine Misfire Detection with Pervasive Mobile Audio". In: *Machine Learning and Knowledge Discovery in Databases*. Ed. by Bettina Berendt et al. Cham: Springer International Publishing, 2016, pp. 226–241. ISBN: 978-3-319-46131-1.

[112] Joshua Siegel et al. "Smartphone-Based Vehicular Tire Pressure and Condition Monitoring". In: *Proceedings of SAI Intelligent Systems Conference (IntelliSys) 2016*. Ed. by Yaxin Bi, Supriya Kapoor, and Rahul Bhatia. Cham: Springer International Publishing, 2018, pp. 805–824. ISBN: 978-3-319-56994-9.

[113] Sneha Singh, Sagar Potala, and Amiya R Mohanty. "An improved method of detecting engine misfire by sound quality metrics of radiated sound". In: *Proceedings of the Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering* 233.12 (2019), pp. 3112–3124.

[114] Smithsonian. *Charles Proteus Steinmetz, the Wizard of Schenectady*. URL: https://web.archive.org/web/20200102001210/https://www.smithsonianmag.com/history/charles-proteus-steinmetz-the-wizard-of-schenectady-51912022/.

[115] Marcin Staniek. "Using the Kalman Filter for Purposes of Road Condition Assessment". In: *Smart and Green Solutions for Transport Systems*. Ed. by Grzegorz Sierpiński. Cham: Springer International Publishing, 2020, pp. 254–264. ISBN: 978-3-030-35543-2.

[116] Statista. URL: https://www-statista-com.ezproxy.biblio.polito.it/statistics/693281/smart-intelligent-sensor-market-size-worldwide/.

[117] J. A. Stork et al. "Audio-based human activity recognition using Non-Markovian Ensemble Voting". In: *2012 IEEE RO-MAN: The 21st IEEE International Symposium on Robot and Human Interactive Communication*. Sept. 2012, pp. 509–514. DOI: 10.1109/ROMAN.2012.6343802.

[118] Vaidyanathan Subramaniam. *The Microsoft SQ1 is a custom version of the Snapdragon 8cx with 2x more GPU performance than an 8th gen Intel Core CPU*. URL: https://web.archive.org/web/20200131230216/https://www.notebookcheck.net/The-Microsoft-SQ1-is-a-custom-version-of-the-Snapdragon-8cx-with-2x-more-GPU-performance-than-an-8th-gen-Intel-Core-CPU.436786.0.html.

[119] R. Sun et al. "Combining Machine Learning and Dynamic Time Wrapping for Vehicle Driving Event Detection Using Smartphones". In: *IEEE Transactions on Intelligent Transportation Systems* (2019), pp. 1–14. ISSN: 1558-0016. DOI: 10.1109/TITS.2019.2955760.

[120] Yongkui Sun et al. "Strategy for fault diagnosis on train plug doors using audio sensors". In: *Sensors* 19.1 (2019), p. 3.

[121] *Support Vector Machines(SVM) — An Overview — by Rushikesh Pupale — Towards Data Science*. URL: https://towardsdatascience.com/https-medium-com-pupalerushikesh-svm-f4b42800e989.

[122] M. Szczodrak, K. Marciniuk, and A. Czyżewski. "Road surface roughness estimation employing integrated position and acceleration sensor". In: *2017 Signal Processing: Algorithms, Architectures, Arrangements, and Applications (SPA)*. Sept. 2017, pp. 228–232. DOI: 10.23919/SPA.2017.8166869.

[123] Mateusz Szumilas, Sergiusz Łuczak, and Błażej Kabziński. "MEMS Accelerometers in Diagnostics of the Articulation of an Articulated Vehicle". In: *International Conference Mechatronics*. Springer. 2019, pp. 292–299.

[124] Shixi Tang et al. "A Fault-Signal-Based Generalizing Remaining Useful Life Prognostics Method for Wheel Hub Bearings". In: *Applied Sciences* 9.6 (2019), p. 1080.

[125] The pandas development team. *Pandas-Dev/Pandas: Pandas*. Zenodo. Feb. 2020. DOI: 10.5281/zenodo.3509134.

[126] Teemu Tossavainen et al. "Sound based fault detection system". MA thesis. Aalto University School of Science, 2015.

[127] Bureau of Transportation Statistics. *Average Age of Automobiles and Trucks in Operation in the United States*. URL: https://web.archive.org/web/20190926121944/https://www.bts.gov/archive/publications/national_transportation_statistics/table_01_26.

[128] *Understanding Random Forest. How the Algorithm Works and Why It Is... — by Tony Yiu — Towards Data Science*. URL: https://towardsdatascience.com/understanding-random-forest-58381e0602d2.

[129] Ilyas Ustun and Mecit Cetin. "Speed Estimation using Smartphone Accelerometer Data". In: *Transportation Research Record* 2673.3 (2019), pp. 65–73.

[130] Prokopis Vavouranakis et al. "Smartphone-Based Telematics for Usage Based Insurance". In: *Advances in Mobile Cloud Computing and Big Data in the 5G Era*. Springer, 2017, pp. 309–339.

[131] Dinesh Vij and Naveen Aggarwal. "Smartphone based traffic state detection using acoustic analysis and crowdsourcing". In: *Applied Acoustics* 138 (2018), pp. 80–91.

[132] J. Wahlström, I. Skog, and P. Händel. "Smartphone-Based Vehicle Telematics: A Ten-Year Anniversary". In: *IEEE Transactions on Intelligent Transportation Systems* 18.10 (Oct. 2017), pp. 2802–2825. ISSN: 1558-0016. DOI: 10.1109/TITS.2017.2680468.

[133] Johan Wahlström, Isaac Skog, and Peter Händel. "Driving Behavior Analysis for Smartphone-Based Insurance Telematics". In: *Proceedings of the 2nd Workshop on Workshop on Physical Analytics*. WPA '15. Florence, Italy: Association for Computing Machinery, 2015, pp. 19–24. ISBN: 9781450334983. DOI: 10.1145/2753497.2753535. URL: https://doi.org/10.1145/2753497.2753535.

[134] J. A. Ward et al. "Activity Recognition of Assembly Tasks Using Body-Worn Microphones and Accelerometers". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 28.10 (Oct. 2006), pp. 1553–1567. ISSN: 1939-3539. DOI: 10.1109/TPAMI.2006.197.

[135] Gary Mitchell Weiss and Jeffrey Lockhart. "The impact of personalization on smartphone-based activity recognition". In: *Workshops at the Twenty-Sixth AAAI Conference on Artificial Intelligence*. 2012.

[136] *What Is a Power Spectral Density (PSD)?* URL: https://community.sw.siemens.com/s/article/what-is-a-power-spectral-density-psd.

[137] *What Is the PSD? - VRU Vibration Testing - Power-Spectral-Density*. en. URL: https://vru.vibrationresearch.com/lesson/what-is-the-psd/.

[138] Guangnian Xiao, Qin Cheng, and Chunqin Zhang. "Detecting travel modes from smartphone-based travel surveys with continuous hidden Markov models". In: *International Journal of Distributed Sensor Networks* 15.4 (2019), p. 1550147719844156.

[139] Koji Yatani and Khai N. Truong. "BodyScope: A Wearable Acoustic Sensor for Activity Recognition". In: *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*. UbiComp '12. Pittsburgh, Pennsylvania: Association for Computing Machinery, 2012, pp. 341–350. ISBN: 9781450312240. DOI: 10.1145/2370216.2370269. URL: https://doi.org/10.1145/2370216.2370269.

[140] Tony Yiu. *Understanding PCA*. en. Sept. 2019. URL: https://towardsdatascience.com/understanding-pca-fae3e243731d.

[141] B. Yuan, X. Zhou, and Y. Wang. "Identifying Vehicle's Steer Change via Commercial Smartphones". In: *2019 International Conference on High Performance Big Data and Intelligent Systems (HPBD IS)*. May 2019, pp. 181–185. DOI: 10.1109/HPBDIS.2019.8735446.

[142] J. Zhang et al. "Attention-Based Convolutional and Recurrent Neural Networks for Driving Behavior Recognition Using Smartphone Sensor Data". In: *IEEE Access* 7 (2019), pp. 148031–148046. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2019.2932434.

[143] Boyu Zhao, Tomonori Nagayama, and Kai Xue. "Road profile estimation, and its numerical and experimental validation, by smartphone measurement of the dynamic responses of an ordinary vehicle". In: *Journal of Sound and Vibration* 457 (2019), pp. 92–117. ISSN: 0022-460X. DOI: https://doi.org/10.1016/j.jsv.2019.05.015. URL: http://www.sciencedirect.com/science/article/pii/S0022460X19302780.