Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics
Department of Measurement and Information Systems

Modeling Explanations of Artificial Intelligence Solutions

Bachelor's Thesis

Author
Balázs Márk Hain

Advisor
Prof. Dr. András Pataricza

December 16, 2021

Contents

HALLGATÓI NYILATKOZAT

Alulírott *Hain Balázs Márk*, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy autentikált felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Budapest, 2021. december 16.

Hain Balázs Márk
hallgató

Kivonat

A kiberfizikai rendszerekben rohamosan terjedő beágyazott mesterséges intelligencia alapú megoldások a komplex alrendszerek viselkedésmodelljét gépi tanulási eszközökkel határozák meg. Ezen felül, a teljes megfigyelt rendszert átfogó magasszintű áttekintést ideális, ha - a hibrid modellezés koncepciójának megfelelően - egy diszkrét modell nyújtja. Ezáltal a modell alapú rendszerfelügyeletben kitüntetett szereppel rendelkezik a kvalitatív modellezés, hiszen ezen paradigma magában hordozza a diszkrét rendszerek validálására jól bevált formális módszerek teljes repertoárja használatának lehetőségét.

Azonban a kvalitatív modellezés CPS-sel kapcsolatos rendszerfelügyeletre való alkalmazása sajátos kihívásokat vet fel. Első sorban, a potenciálisan veszélyes állapotokat leíró működési tartományok nem teljes lefedettsége veszélyes vezérlési hibákhoz vezethet. Továbbá, a kiugró értékek különös figyelmet igényelnek a kritikus alkalmazásokban. Legfőképpen pedig a modell minőségét - mint megbízhatóságának döntő tényezőjét - kell garantálni. Különösen figyelmet igényelnek azon hibrid modell kompozíciók, melyek gépi tanulási módszerekből eredő következtetésekkel támogatottak. Ezen mechanizmusok elterjedésének alapvető akadálya, hogy pontos modellezést csak komplex, emberi megfigyelő által átláthatatlan modellekkel lehet kielégíteni. Ennek okán különösen kritikus rendszerekben problematikus a modell interpretálhatósága illetve viselkedésének megmagyarázhatósága.

Jelen diplomamunka az univerzális modell interpretáló eszközök és módszerek alkalmasságára fókuszál a beágyazott mesterséges intelligencia modellek V&V célokra történő kvalitatív absztrakcióihoz.

A témához szorosan kapcsolódó bemutatott tárgyterületeken kívűl általános áttekintés olvasható a magyarázható meseterséges intelligencia tudományág mögött meghúzódó főbb motivációkról, szándékokról és kihívásokról és az elvárásokról. Melyek indokolják a jelenleg zajló erőteljes kutatásokat az adatkészletek és a származtatott modellek magyarázhatóságának javítására (XAI). Számos elérhető magyarázható mesterséges intelligencia-könyvtár közül, két korszerű keretrendszer (IBM AIX360, DALEX) segítségével a különböző modell magyarázó megközelítéseket áttekintjük, bemutatva hatékonyságukat a különböző értelmezhetőségi jellemzőkkel rendelkező gépi tanulási modellek széles skáláján. Egy különálló fejezetben az olvasó betekintést nyerhet saját közvetlenül magyarázható osztályozó algoritmus tervezésének folyamatába is.

Végső soron a kutatás olyan megközelítést javasol, amely lehetővé teszi a fekete doboz szintjén is ismert beágyazott modellek kvalitatív absztrakciós szintű validálását. Ez a technika olyan dimenzióredukciós és klaszterezési módszereket kombinál, amelyek pontosan szeparálják az egyes működési tartományokat, miközben jól illeszkedő klaszterhatárok segítségével felismerik a kiugró értékeket. Különféle értelmezhetőségi módszerek alkalmazásával a működési régiók közötti kohéziós tényezők felismerése és a belső funkcionalitás megértése is elérhetővé válik. Beigazolódik, hogy a modellezési részbe egy bizonyos szintű autómatizáció bevezetése az elemzett adatok jobb megértéséhez vezet.

Abstract

The rapidly spreading solutions based on embedded artificial intelligence in cyber-physical systems have defined the behavioural model of complex systems with machine learning tools. In addition, a high-level overview of observed system is ideal - according to the concept of hybrid modelling - if a discrete model provides it. Thus, qualitative modelling has a prominent role in model-based supervisory control, as this paradigm implies the possibility of using the full repertoire of well-proven formal methods for the validation of discrete systems.

However the application of qualitative models in CPS control raises specific challenges. Firstly, insufficient coverage of potentially dangerous operational domains may lead to hazardous control errors. Moreover, outliers need special care in critical applications. Especially, the quality of the model - as the decisive factor of its faithfulness - must be guaranteed. Hybrid model mixtures supported by machine learning provided inferences require particular attention. A fundamental obstacle to their prevalence is that accurate modelling can only be satisfied with complex models that are opaque to the human observer. For this reason the interpretability of the model and the explanation of its behaviour are problematic, especially in mission-critical systems.

This dissertation focuses on the suitability of universal model explanatory tools and methods for qualitative abstractions of embedded AI models for V&V purposes.

In addition to the domain fields closely related to the topic, a general overview will be given on the major motivations, intentions, and challenges behind the concept of explainable artificial intelligence. These ideas justify the growing demand to improve the explainability of data sets and derived models (XAI). From the various available XAI libraries, different model explanatory approaches will be showcased with the use of two highlighted state-of-the-art explainability libraries (IBM AIX360, DALEX), demonstrating their effectiveness in a wide range of machine learning models with different interpretability characteristics. In a designated chapter, the reader can also gain insight into the process of designing a unique directly explainable classification algorithm.

Ultimately, the research proposes an approach that allows for qualitative abstraction-level validation of embedded models known even at the black box level. This technique combines dimensionality reduction and clustering methods that accurately separate the operational domains while also recognising outliers using well-fitting cluster borders. Various interpretability methods will be used for highlighting the cohesive factors amongst the operating regions and guide the understanding of the functionality as well. It will be shown that the introduction of a certain level of automation in the modelling part leads to a better understanding of the analysed data.

Chapter 1

Introduction

In recent years, machine learning (ML) greatly impacted a wide variety of scientific fields with its successful and powerful applications. This unprecedented advance is due to the high level of performance and precision of ML algorithms when they are required to solve increasingly complex computational tasks. The broad selection of the applications consists of spam or fraud detection, natural language processing, gene sequencing, or revolutionising the field of image recognition.

Deploying ML models in real-life applications is a practical choice since they do not require any human intervention to make predictions constantly. Therefore these models are often utilised in such environments, where the models' predictions would inevitably influence human lives. As an example, there is also an emerging tendency to digitalise administration and thereby speed up bureaucracy related processes. However, when a ML algorithm declines loan application, several understandable issues are raised. The user might argue that the adjudication was unfair, how can the bank prove the opposite towards legislation forces? Or even in case of the customer excepting the decision, no justification will be provided for him to improve his credit score.

One of the major criticisms of ML algorithms concerns the lack of understanding of the decision-making process that establishes their use. As a trade-off for immense performance, ML models tend to be very complex, therefore they are often regarded as *black-box* models. The decision-making aspect of a black-box model (i.e. deep neural networks) is opaque and alien not only to a human observer but to the developers as well. Applying these models in a mission-critical context entails the danger of using decisions that are unjustifiable since these algorithms do not allow obtaining detailed insights into their behaviour.

Corporates have both legal and moral obligations to explain their decision-making process in situations of great responsibility, that frequently occur in these regulated industries. Furthermore, extracting explanation of their models guides the developers to understand and fundamentally improve their workflow and infrastructure.

The topic of interpreting machine learning models first appeared in 2017 when the Defense Advanced Research Projects Agency (DARPA) published its Explainable Artificial Intelligence Program[16]. The publication defined requirements for building ML models and designing explanation interfaces and also proposed solutions in regards to these challenges. According to the paper, the following design principles should drive the implementation of such interfaces, as interpretability:

- helps ensure impartiality in decision-making (by excluding bias from the system),

- facilitates the provision of robustness (by suggesting certain domains, where the model confidence is low),
- guarantees truthful causalities in the model reasoning,
- assures that the sample set is representative.

In the following years, open-source AI explainability libraries have become available to the industrialisation of machine learning-based application development. At first glance, algorithms consisting of these frameworks provide immediate and easy-to-grasp visual aids and statistical analysis, to improve the observability of the results. Nevertheless, since the topic of interpretability can be approached in various ways, there is no "one size fits all" solution, implying that a deeper understanding of these methods is required to explain models thoroughly. These models often inherit ideas from fields of cognitive and social sciences to complement human understanding and oversight of a large-scale system.

Artificial intelligence modules can also operate as embedded components of a compound cyber-physical system to enhance the system's autonomous capacities or to fulfil highly complex tasks.

Modelling of such a system encompasses the entire architecture as well as modules relying on AI-assisted reasoning. While these high-level abstract models also serve as a target of validation and verification processes. Since the AI component is heavy influenced by the containing environment - in the form of inputted datasets, it is crucial to inspect whether the dataset is representative of the modelled environment, otherwise the system's behaviour is unfaithful as part of the V&V. A traditional way to conceptualise a compound system is to define its state space. This space can be partitioned into operating regions where the system's behaviour is homogeneous. These states can be further analysed to gain a deeper insight into the operation or to detect hazardous states that can lead to an erroneous behaviour. The modelling process should also cover arising (causal, differential) relationships between entities and events.

The key principles in qualitative modelling entail the extraction of an abstract representation from detailed (partial) models and observations by aggregating and discretising all the continuous features. Qualitative abstraction maps entire subsets of the continuous state space corresponding to an operational domain exposing similar behaviour to a single qualitative state. Similarly, ordinal qualitative values represent subdomains of individual continuous variables (e.g., *low, medium, high*) into ordinal variables, thus preserving their relative magnitude and (partial) ordering.

The structure of qualitative models preserve the dynamic architecture of the target system. Thus, they are natural choices to build a digital twin serving as the core for supervisory control of complex cyber-physical systems. The major drive behind qualitative modelling is expressing conceptual knowledge about the system structure, dynamics, causality relations in its functionalities, assumptions about its operation, and qualitatively distinct operational domains. Common engineering thinking motivates this modelling paradigm, resulting in easy-to-understand and well-interpretable models.

Structure of the thesis

This thesis aims to interrelate the tools drawn from the field of explainable intelligence and qualitative modelling. Thereby methods are proposed and described that allow the extraction of a qualitative model describing a dynamic system. The suitability of these tools for validation purposes will also be inspected.

Firstly the thesis is concerned with presenting the qualitative modelling (Chapter 2) paradigm, which is a form of knowledge representation, allowing the reasoning of continuous systems in a human-like manner. It describes the key notions of the model building for forming operational states and establishing qualitative relationships. Eventually, a brief demonstration shows how to create the qualitative model of a dynamic system. The rest of the thesis tries to find a solution, how to mimic this process with the use of XAI tools.

The subsequent (Chapter 3) overviews general machine learning concepts. It presents commonly used knowledge representation and reasoning techniques to showcase the advantages and disadvantages of different forms of storing information about the environment. The ML models act according to the inferred reasoning from their knowledge base. AI explainers use the same knowledge-base to create easy to explain models and explanations. Later on, ML algorithms will be classified based on their problem formulation and learning setting. Different evaluation metrics are shown as part of the model validation to quantify the quality of the model's prediction. A collection of commonly used ML models will be overviewed, ranging from different parts of the interpretability spectrum. Eventually, concerns that require special attention from the data scientist upon model fitting will be presented as well.

Chapter 4 introduces the reader into the constantly growing field of XAI, highlighting the major motivations, objectives, and challenges that the field frequently faces.
There are two distinct universal model-agnostic methods to provide explanations of an arbitrary black-box model:

- creating a directly interpretable surrogate model to mimic the behaviour of the black-box model in a much simpler form,
- using post-hoc interpretability techniques to extract the model's key characteristics.

For each approach, algorithms originating from separate state-of-the-art frameworks will be used. Neural networks deserve special attention, as they are typically explained with model-specific tools, due to their relevancy and cumbersome explanation capacities.

In a dedicated chapter (Chapter 5) a newly introduced AI explainer, of great impact regarding the field domain, is dissected to allow its implementation-level analysis. Leveraging its individual problem formulation, a directly explainable classifier is proposed that has the ability to surpass its counterpart. The developed algorithm adapts an already existing method to generate interpretable surrogate models with gradually increasing accuracy based on computation time. It draws ideas from the field of mathematical optimisation and also from a publication presenting a unique way to approximate N-dimensional subspaces with convex-hulls.

Eventually, research was conducted to substantiate the claims made in regards to the suitability of XAI tools and qualitative abstraction. The workflow was outlined in Chapter 6. Its primary intention is to showcase the ability to define a qualitative membership function for each operational region. The process combines dimension reduction and clustering methods, while it also pays special attention to careful feature selection and outlier detection. The cohesive factors forming the resulted operational domains will be eventually analysed with XAI tools, that provides a straightforward justification to the observer.

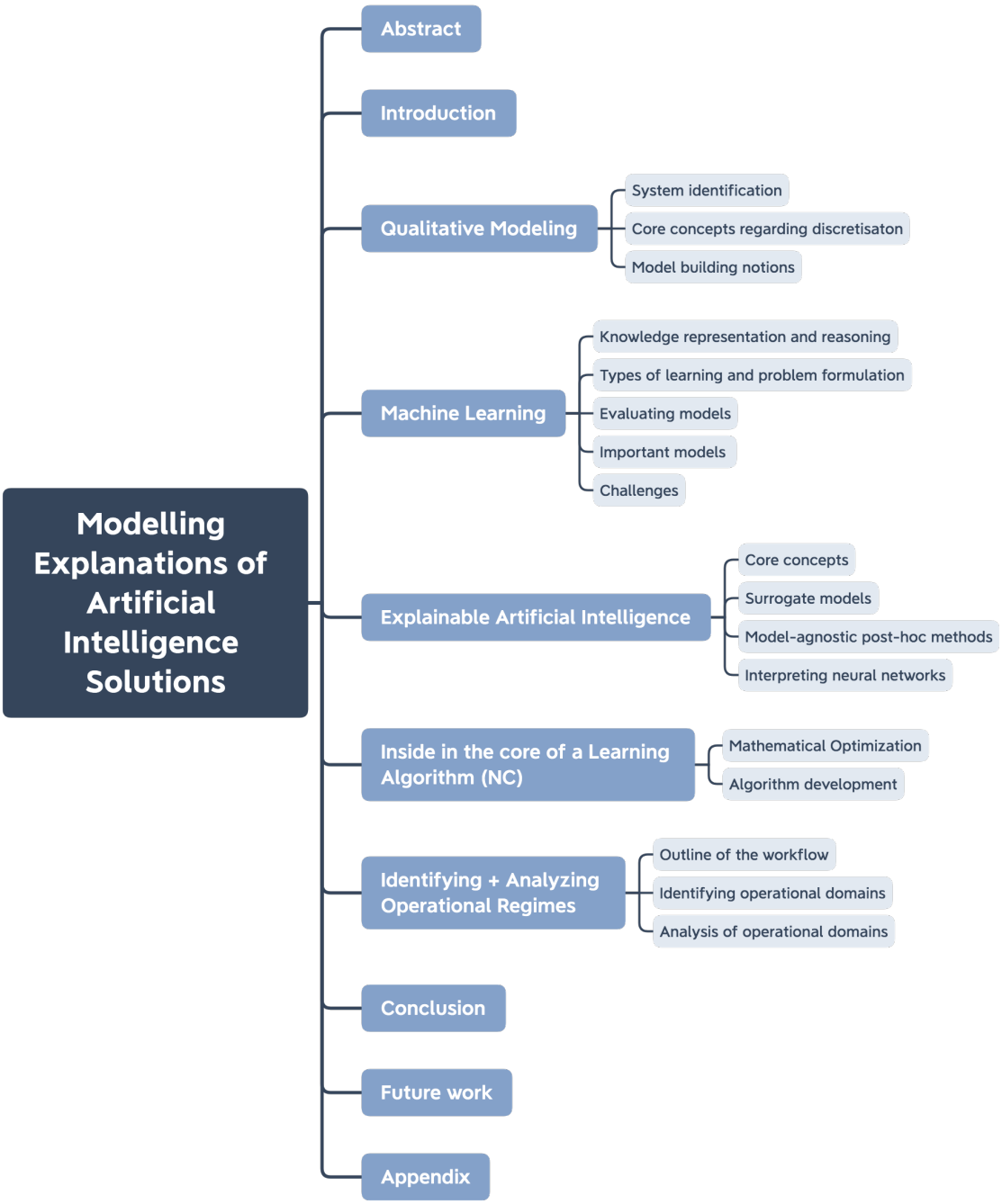


Figure 1.1. Structure of the thesis.

Chapter 2

Qualitative modeling

Qualitative modelling is a model-based reasoning paradigm that aims to generalise the properties of a complex continuous system on an abstract level while inferring from minimal information.

The constructed model expresses conceptual knowledge about the system structure, dynamics, causality relations in its functionalities, and distinct operational domains. Its core idea is to extract a high-level representation from detailed observations by aggregating and discretising all the continuous features. The resulting qualitative state-space has the ability to characterise broad categories in order to identify both distinct behavioural patterns and outcomes. Common engineering thinking motivates such modelling paradigm, resulting in easy-to-understand and well-interpretable models.

This proposed methodology has a long tradition in different fields of science as descriptive means, such as finance, ecology, and natural language semantics.

2.1 System identification

2.1.1 Hybrid modeling

Simple, control systems often operate based on the *sense-think-act paradigm*, resulting in a closed-loop physical system. Such systems are realised with the means of pure mathematical models. Typically the dynamic nature of these systems is expressed as numerous differential equations. This way, the designed system can be described accurately, thereby its resilient operation is validated through simulation.

However, modern complex distributed IT systems can only be identified via complex models since they often integrate COTS components[1] and external services. Models with such complexity would result in a vast number of equations, which would be computationally unmanageable at this scale.

Pure analytical models have often constrained the use-cases due to the limited applicability of the distributions they impose. It is caused by the fact that some real-life phenomena can only be approximated with a selected distribution, but all of its constraints will not be met fully. Prime examples for this would be the non-independence of variables or the incomparable behaviour in ranges of different domains. It can also be challenging to find the distribution that best describes the problem. Phenomena may also arise that can

[1] **Commercial off-the-shelf** products provide out-of-the-box hardware or software solutions that are tailored for the organisation's aftermarket needs, instead of using custom / in-house solutions.

only be described with the fusion of multiple distributions. Therefore, due to the limited faithfulness of pure, statistical models, their exclusive use would also be imprudent.

The modelling of (extra)functional properties of complex CPS systems frequently demands hybrid system modelling in the form of *empirical system identification*. Mathematical functions are extracted from the system's in- and output during system identification by observing the measured data. When the deep understanding of the underlying structure is unwanted or not feasible, these relationships are constructed empirically. Empirical system identification establishes an engineering model, describing the a priori knowledge on the system, such as physical constraints, causal relationships or data flow. This extension complements the subsequently constructed visual and statistical analyses, which gain insight into the model's main characteristics, revealing relationships between variables or discovering typical behavioural patterns.[20]

This model fusion is assisted by reasoning provided by an embedded Artificial Intelligence (AI) component found in the cyber-physical system. (Chapter 3) For instance, this can be an image processor, which has such a complex behaviour that only an AI can realise it. However, its validation and modelling are still crucial. Therefore the model is extended with an AI explainer which checks the compliance with system requirements, providing a direct explanation of the underlying complex model, which either can be black-box like or semi interpretable.

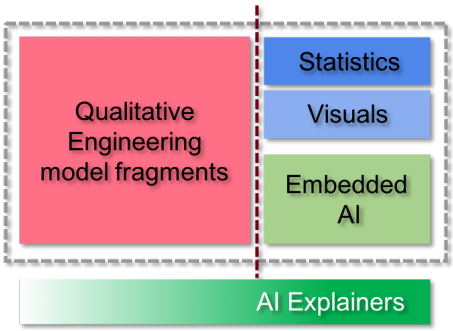


Figure 2.1. Common representation of a model fusion in Hybrid Modelling.

Eventually, the resulting discrete, qualitative model is a fusion of the engineering design model and several mathematical model fragments based on the quantitative properties of the target system. This methodology ensures the ability to use the entire repertoire of well-established formal methods to validate discrete systems, which is fundamental in operation-critical systems.

The concept of empirical system identification allows creating compound scalable and hybrid models that provide a guaranteed level of service under the typically fluctuating workload.

2.1.2 Supervisory control

In various IoT applications in the field of industrial manufacturing and automation, the concept of "Digital Twin" (DT) has been heavily integrated into operation-critical systems. Its primary objective is to maintain a high level of overall monitoring and control of the individual processes. During its operation, it is able to create high-fidelity models, to simulate how the system operates in different circumstances while also virtually recreating real-life properties (such as the dynamics) of the physical environment. [27]

A bi-directional connection between the hardware instance and the mirrored image allows the given CPS solution to predict several events/states through mathematical models based on the perceived real-time sensor data. With the awareness of these predictions, it is trying to optimise its operation and avoid malicious behaviour.

The simulated model (DT) requires the same hybrid modelling needs as underlined above. Firstly the distinct operational domains need to be established to capture similar behavioural patterns and define the relationships / transitions between them. This allows for model extraction from minimal information. The level of abstraction - how fine the separation between distinct states was - is determined by the relevance of each to the central problem. In many cases deeper understanding of concrete, the dynamic structure of the system is unnecessary. However, a set of operational regimes can be eventually analysed individually, which will be a primary subject of a subsequent chapter in this thesis. When modelling a distributed stochastic environment, precise operational regularities are often unavailable for the engineer; however, correct qualitative models can be built without them.

Therefore the structure of qualitative models well-reflect the dynamic architecture of the target system. Thus, they are natural choices to build a digital twin serving as the core for supervisory control of complex cyber-physical systems. There are several factors, which must be taken into account when integrating into supervisory control:

1. The quality of the model must be guaranteed as the decisive factor of its faithfulness,
2. Insufficient coverage of potentially dangerous operational domains may lead to hazardous control errors,
3. Outliers need special care in critical applications.

2.2 Core concepts

Qualitative modelling, at its core, represents human-like thinking, how we intuitively approach everyday tasks as life.

Let us take making a mug of coffee as an example with a Moka pot. There are very complex physical theories (*Ideal gas law, Darcy's law*) behind it, how the Moka chamber is constructed to control and lead the pressure inside, which eventually brews the coffee. Nevertheless, fortunately, their intuition (apriori knowledge), which was gathered either from the user guide (system specification) or through experiences (empirically), suggests the correct use of the pot without understanding advanced physical equations. It is known that the more heat is added to the stovetop, the hotter the beverage will be, the faster the coffee brews. Experienced users might define more insightful relationships between the entities and the events, like brew time and level of extraction, or grind size and acid levels, resulting in a better-tasting coffee. However, notice that the primary objective (only to brew a coffee) is feasible without knowing these advanced relationships.

The formalisation of the example mentioned above shows the following conclusion. Qualitative abstraction maps entire subsets of the continuous state space corresponding to an operational domain exposing similar behaviour to a single qualitative state. Similarly, ordinal qualitative values represent subdomains of individual continuous variables (e.g., *low, medium, high*) into ordinal variables, thus preserving their relative magnitude and (partial) ordering. Qualitative functions and relationships can be defined on properties that hold true in each qualitative state to trace causality or some algebraic connection.[14]

2.2.1 Discretization

The first layer of abstraction is represented as a form of discretisation. Instead of using an infinite range of interpretation, properties of the continuous system are described with only a set of essential values. The substantial value of a selected property is irrelevant, only its magnitude that matters. This allows for inferring as much as possible from minimal information.

The foundation of this modelling paradigm lies in how the discrete ranges were formed. Since the constructed domains can be thought of as operational regimes, where the system is expected to show similar behavioural patterns, these qualitative domains are separated by operating area boundaries (landmarks). Considerate selection of landmarks is vital because they serve as a base of (eventually formed) relationships and other algebraic functions between entities.

Landmarks can be chosen based on either of the constraints from both the nature of the system or the reasoning to be done about it. Therefore their extraction can be approached from the division of either the perception regions or the operation regions. The bottom-up approach takes several domain ontologies into account, where the expert, by using apriori knowledge on the environment, the expert tries to divide the input data manually into segments and then maps it to the qualitative state space. Whereas the other approach conducts the post-hoc analysis of the system behaviour, thereby realising the boundaries. Using the first method, various constraints or criteria dependent on the given task can be imposed on the model, while the other results in a more precise description and is usually applied in black-box system identification.

2.2.2 Relevance

All prominent values from the discrete domain are chosen to be relevant for some classes of tasks. Since a specific system can be reasoned meaningfully with different sets of landmarks, during model creation, the modeller has to consider not only the structure of the system but also the task-specific criteria that are the objective of the optimisation / analysis.

This level of abstraction determines how much depth or detail should each qualitative state carry.

2.2.3 Ambiguity

As highlighted earlier, it is often preferred when the model extraction concludes in an automated manner from the analysis of other model fragments or the fusion of statistical models. Automation can identify relevant phenomena or anticipated states of behaviour. Without strict mathematical formalisation and boundaries, traditional statistical models would impose ambiguous predictions made by qualitative models.

This modelling concession has the advantage of allowing the engineer to build a model with incomplete knowledge of the given system.

2.3 Notions of building qualitative models

2.3.1 Qualitative states

Most qualitative systems support some form of propagation of valuable information through qualitative relationships, enabling information about one part of a system to infer information about other parts. In contrast, others support more (in)equation-like algebraic manipulations, such as ordering different domains.

The formulated relationships between two or more qualitative entities offer insight into how we want to reason the system, but also influences the finite set of values describing the system.

The first striking idea that one could consider when describing a selected state is **binary classification**. Behaviour is either called *normal*, when the standard operation is present with some slight deviation from the optimal range, or *abnormal* when the represented variables are completely diverged from their expected values. The use of a binary state is very advantageous since binary classification models are widespread, and also, some more complex models (e.g., Decision trees) or model evaluation metrics can only be built based on them.

A slightly more refined way of thinking appears via the introduction of **signs**. Here a state can be viewed as *positive, negative, or zero*. Notice that the concept of continuity is present here since the represented attribute can only become negative from a positive state after it passes zero. Algebraic operations and comparison can be defined on this state-space. For instance, it is known that the addition of the positive values results in also a positive value; or a positive value that is bigger than a negative one. However, some level of ambiguity is also present since the result of the addition of a positive and a negative value is unknown. This can be resolved by introducing a new state, called unknown, or by qualitative simulation. Qualitative simulation computes all possible outcomes on a separate branch or by propagating the uncertainty further into the following states of the simulation, however it can also make the model very complex as a byproduct.

Figure 2.2. How to be certain if this is a representative finite value system for the feature `latency`?

A more generalised approach is the creation of a **finite value system**. It is constructed by quantising the initial continuous state-space by more nuanced standards. These considerations must rely on either deep domain knowledge, or are inferred from hypothetical model fragments. The thesis aims to find solutions for the latter topic. The main objective in qualitative modelling is finding division that is representative of the system. An example is shown in Figure 2.2. If landmarks defining the finite value system is not available, it is a good practice to opt for a higher resolution. Later, when a more profound behavioural model of the system is available, (i) states showing similar behaviour can be merged, and (ii) redundant/unexplored states can be ruled out. Even though adding redundant states to the model results in a more flexible model, it defeats the purpose of the qualitative abstraction, as it induces even more ambiguity and loses tractability. The more detailed a system is modelled by qualitative variables, the more difficult it is to follow the simula-

tion. More specifically, for qualitative features, each of which can take on a value of M variables, M^N possible next states are present during the simulation.

2.3.2 Qualitative relationships

Qualitative relationships express environment imposed limitations and characterise a system's dynamic behaviour. Algebraic relationships describe static situations, while integral or differential relationships capture changes that occurred to the system over time. Both can be represented in a qualitative model at the same type, depending on the level of detailedness and required reasoning.

The weakest algebraic relationship that encapsulates dynamic characteristics are linear monotonic relationships or qualitative proportionality. These are defined as:

$$A \propto_{Q^+} B$$

implying that A can be determined as an increasingly monotonic function of B.
In qualitative modelling, all functions are designed to entail causal relationships between such events. Here B is also presented as the cause that results in the increment of A. Relationships are constructed according to the closed-world assumption over the collection of remaining qualitative proportionalities that restrict A, unaffected by the cause. The reasoning behind it is twofold: they allow inferring from an initial minimal information, and in case of a wrong deduction the other assumptions can be re-examined.

Derivative relationships enclose the dynamic behaviour of the system, by describing changes over time. ∂Q - similarly to the mathematical notation - shows the qualitative value of the time-derivate of Q. E.g., $\partial Q \in \{decreasing, constant, increasing\}$.

A brief example - regarding the Mokka pot - is shown the summaries these concepts. Let H be the level of heat applied to the stove, and T_C is the temperature of the coffee being made. It is clear that $T_C \propto_{Q^+} H$, implying that an increase in the coffee's temperature *can be* caused by an increment in the heat setting. However $\partial T_C - \partial H \neq 0$ since the event $\partial T_C > 0 \wedge \partial H = 0$ can occur when the water's temperature has reached its boiling point.

The last remaining question stands as: h*ow complete are qualitative representations relative to ordinary differential equations?* The resolution to this is proposed by the field of qualitative differential equation (QDE), where non-monotonic functions are decomposed into monotonic segments. The introduction of QDE is not part of this given thesis, however, a sizeable literary source can be found about it here: [21].

2.3.3 Validation

A soundness of a qualitative model can be validated with qualitative simulation in addition to well-proven formal methods for the validation of discrete systems. The aim of a qualitative simulation is to generate qualitative states from a selected initial state by exploring the predictions made on the future behaviour of the modelled system. Qualitative simulations are generally used to reveal interesting occurring events that can be a focal point of subsequent analysis, reasoned by more detailed models.

A qualitative state space is traversed, by finding transitions that can lead to yet unexplored states. These can fire, when a qualitative value or a condition based on it changes. A transitions can arrive at more than one possible next states, due to the ambiguity of a finite value system.

Well-designed qualitative simulation algorithms can generate the entire space of possible behaviours, but can include predicted futures which are not actually possible, due to the level of abstraction, resulting an unsound but complete simulation. By inducing constraints representing the environment, it is possible to rule out certain incorrect states.

Garp3[9] is the only widely recognised workbench for constructing, simulating, and observing qualitative models.

2.3.4 Demonstration

One of the key intentions of qualitative reasoning is to provide a more comprehensive representation than a collection of differential equations do. When a differential equation describing the behaviour of a system is available, than it can effortlessly be transformed into qualitative relationships. An example model of a simple system shows this process. [15]

Differential equations can be transformed into qualitative relationships so easily, since the physical laws define valid environmental states and transitions between. Below the example shows the behavioural model of a simple valve.
Let Q be the rate of the water flow passing through the valve, P the pressure difference between the start end the endpoint of the observed canal, and A the cross-sectional area of the valve. The equation defining the flow through an orifice is given by:

$$Q = CA\sqrt{2P/\rho}$$

where C is a constant and ρ is the density. The following model described by a dynamic equation will be transformed into a qualitative sign algebra. Let [Q] be the sign of the continuous parameter Q.

$$[Q] = [C][A][\sqrt{2P/\rho}]$$

where since $Q > 0 \implies [C] = + \wedge [A] = +$ and $[\sqrt{2P/\rho}] = [P]$.
This reduces the equation to $[Q] = + + [P] \implies [Q] - [P] = 0$. To reveal (differential) qualitative relationships between the reasoned quantities the time derivative of the equation should be taken.

$$\frac{\partial Q}{\partial t} = C\sqrt{\frac{2P}{\rho}}\frac{\partial A}{\partial t} + CA\sqrt{\frac{1}{2P\rho}}\frac{\partial P}{\partial t}.$$

By transforming the equation into sign algebra and assuming that each observed quantity is positive ($P, A, Q > 0$), then the equation simplifies to the required qualitative differential relationship: $[\partial Q] = [\partial A] + [\partial P] \implies [\partial A] + [\partial P] - [\partial Q] = 0$.
The described model now can be fully simulated given an initial state (closed valve: $[A] = 0, [Q] = 0$), and environment induced knowledge (e.g., if the valve opens, the water will flow), and different (qualitative) states of the target event (flow through an orifice) can be observed.

One of the main objectives of this thesis is to provide means to firstly extract a tangible behavioural model (decision-making process) of a black-box representation, that maps the parameters of the system (model's features) to an observed event (target variable), and provide a precise qualitative abstraction of the system based on the extracted qualitative relationships. This method is an ideal approach in case of the absence of such precise description.

Chapter 3

Machine learning

Machine Learning is a subset of artificial intelligence (AI), a scientific field that tries to acquire new knowledge based on identifying already existing knowledge - usually in the form of datasets. This method is a form of "indirect programming" where the programmer does not explicitly state how the model can achieve the expected results, rather than letting the algorithm find underlying patterns or deep insights to improve its predictions or behaviours based on the inputted data.

While learning these behaviours, the algorithm builds an engineering model - an abstract generalisation - of the given dataset/problem. Based on the model's features, it can promptly conclude a newly emerging sample. This representation, however, can be arbitrarily complex depending on the chosen ML algorithm or on the dataset. Therefore, AI explainability plays an immense role in the system's decision-making, mainly if applied in a highly critical corporate field such as medicine or banking.

The primary focus of this chapter will be the acquisition and representation of knowledge in different forms (First-Order-Logic, Qualitative models, and other Model-based techniques) and their reasoning with ML techniques.

3.1 Knowledge representation and reasoning

3.1.1 Relationship between knowledge and intelligent behaviour

The primary purpose of operating such systems, endowed with some level of Artificial Intelligence, is to solve complex real life problems by constantly making decisions. These decisions are made based on the following aspects:

- perceiving information from the environment - while also adapting to its changes,
- retrieving inner conceptual knowledge (beliefs),
- deducing new connections between the initial beliefs and the freshly occupied intel.

These systems will be deciding for the best-expected outcome to the best of their knowledge. Therefore, the aggregated knowledge's quality, correctness, and quantity are essential for adequate prediction. To put it simply, the fewer opportunities occur for the AI model to look at examples and infer knowledge from them, the narrower its knowledge base will be. Furthermore, inconsistent belief-states (resulting from a flawed way of perceiving the environment or incorrect deductions) will lead to proportionate uncertainty.

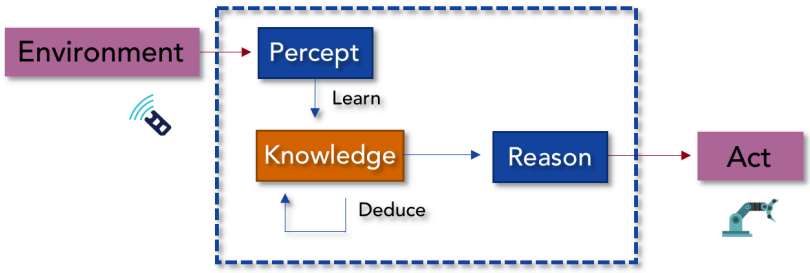


Figure 3.1. Cycle of Knowledge Representation in AI.

The process of knowledge representation (KR) and reasoning encapsulates crucial elements from the interdisciplinary field of cognitive science, mimicking human-like thinking. The decision-maker's actions (the system) are solely justified by sensing the environment and using the acquired experience. Rather than sequential, procedural code, where the course of the operation is hardwired in the shape of an algorithm, the system conceptualises several domain ontologies and eventually decides while reasoning about the occupied knowledge. This paradigm also narrows the gap between the developers and the business personnel, making the discourse slightly more interpretable. [22]

ML is inherently related to data analysis and statistics. In this thesis, all learning models and other proposed techniques will be viewed from a data scientist point of view. Meaning, any type of information obtained via interaction with the environment (e.g., sensor data aggregation) that serves as the input of an ML algorithm will be thought of as simple relational knowledge. This data is provided in the form of a labelled training dataset, which refers to any type of preliminary information available for the learning algorithm in a database scheme.

Since the success of the learning algorithm is heavily dependent on the provided data, the primary aim of ML model creation is to find a representation from which knowledge regarding the environment and the given task accurately and practically can be queried and stored in. A common way of representing knowledge has been already discussed in Chapter 2, a paradigm on which all subsequent research will be built. This modelling concept has outlined some of the conventional aspects of knowledge representation regarding the induced level of abstraction, relationships, reasoning, interpretability etc. Now it is high time to formalise these expectations of a knowledge representation model generally. These requirements are:

- **representational adequacy**: system's ability to represent the required knowledge,
- **inferential adequacy**: system's ability to manipulate the knowledge represented to produce new knowledge corresponding to that inferred from the original.,
- **efficiency**: system's ability to direct the inferential mechanisms into the most productive directions by storing appropriate guides,
- **acquisitional efficiency**: system's ability to acquire new knowledge using automatic methods wherever possible rather than reliance on human intervention.

3.1.2 Logic representation

The involvement of logical theories in AI was clear from the very beginning of the emerging of this field due to the profound connection of theoretical computer science and mathematics. Long before the computer era, mathematicians used logic concepts to formalise

declarative knowledge. This representation was a solid fit for their needs since it provided a very robust way of describing ground truths of the modelled world on a high level, meantime having essential inferential properties. [25]

This KR paradigm will serve as one of the general foundations of research work concluded in the scope of the thesis, therefore it is necessary to overview some of its core concepts.

Logic representation is immensely applied in the field of electronic circuits as well. The widespread use provides many tools to model logic relationships and functions, one of which is **pyEDA**, which will be used in the research work.

Propositional logic

The semantics of language define the truth of every predicate for each **model**. Formally, models here are thought of as all possible assignments to the **variables** (*proposition symbols*) representing attributes of the modelled environment. Each assignment records the truth of any arithmetic statement that contains those variables.

In this language, the inference is expressed with the means of logic entailment relation. That relationship between two modelled entities represent the fact that predicate **b** is logically followed from predicate **a**. This fact is denoted in the following way:

$$\mathbf{a} \models \mathbf{b}$$

By utilising the inference property of the language, new (possibly simpler) predicates can be deduced. Eg.: $(\mathbf{a} \wedge \mathbf{b}) \vee \mathbf{b} \models \mathbf{b}$. Inference also plays an essential role in model checking.

Predicates can be *atomic*, if it consists of a single variable, or *complex* if simpler predicates are connected with **logical connectives**. Logical connectives can be thought as logical operations, these are: $\neg$ (negation), $\vee$ (disjunction), $\wedge$ (conjunction), $\Rightarrow$ (implication), $\Leftrightarrow$ (equivalence). (Negated) Atomic predicates are considered **literals**, whereas complex predicates are referred to as **clauses**.

A typical approach to evaluate logic formulas happens via simple substitution, truth tables, or SAT-solvers.

Rulesets

In the business world, propositional logic is applied in a much simpler setting by using rulesets. While providing a more straightforward and interpretable way of abstraction, rulesets still have the ability to encapsulate complex mathematical functions indirectly. This makes it a powerful asset for both machine learning and explanatory methods.

Rulesets solely capitalise on the implication logical connective, thus creating a set of

$$\mathbf{IF}[\mathit{condition}] \rightarrow \mathbf{THEN}[\mathit{consequence}]$$

structures.

Conditions appear as a logical AND connection of *literals* resulting a *clause*, whereas a set of clauses is a ruleset. There are two canonical ways to structure a ruleset, depending on which logical connections were used during construction: *conjunctive normal form* (**CNF**), and *disjunctive normal form* (**DNF**). CNF is a logical formula consisting of a disjunction of conjunctions (OR of ANDs), whereas DNF consists of a conjunction of disjunctions

(AND of ORs). When a given condition is met, then the ruleset is satisfied, the learning algorithm will arrive at a decision as a consequence.

Rulesets can be characterised with simple metrics such as **accuracy** (how well the constructed model approximates the system) and **complexity** (how many rules are required to describe the system).

Complexity is an essential factor to consider because as these sets of rules grow in number, their complexity will also rise especially fast. Therefore some applications of rulesets are required to remain manageable. That is why sometimes it is necessary to formulate constraints on their complexity. A common way to construct a cost-function for the complexity of a given ruleset is pricing the number of clauses, and the number of terms.

$$c = \sum_{k \in K} \delta_0 + \sum_{i \in k} \delta_1 \tag{3.1}$$

where K is the set of clauses, k is a clause, i is a literal, δ_0, δ_1 are pricing constants.

First Order Logic

Propositional logic is a clear starting place for describing declarative knowledge. However, it is inadequate for modelling domains with a large number of objects — especially when formalising ground-truths for an entire subset of entities.

First-order logic reasons **objects** instead of simple propositional symbols, Objects in the model, representing domain elements, can have attributes, and they can be in **relation** with each other (relations are referred to as a set of simple tuples). Relationships between objects are mostly considered **functions**, where one object is in a relationship only with another object. Which requires that each input tuple must have an assigned value.

Since it is often uncomfortable to name or assign each described entity to an object, therefore in first-order logic, they can be referred with a **term**, with a logical expression related to the object. (*Eg.: Instead of representing a selected module - which also happens to be a submodule of ModuleMain - as SubModule756, it can be referred with a term like: SubModule(ModuleMain).*)

First-order logic also introduces two quantifiers (**existential** $\exists$**, universal** $\forall$) to reason an entire collection of objectives.

This KR paradigm is heavily used in declarative (such as Prolog) and inductive programming languages. The latter is used for learning data cleaning and transformation programs by data scientists.

Higher-order logic languages are also available (second-order, modal, temporal logic, etc.), where each logic adds a new dimension to modelling different aspects of the world. However, only these three concepts will be thoroughly utilised in this work.

3.1.3 Model-based Problem Solving

Systems that apply causal rules are called model-based reasoning systems because causal rules model the environment. The concept of model-based systems has emerged to complement the flaws of purely empirical model extraction methods. This subsection aims to

provide a brief overview of these limitations to guide the understanding of the challenges that empirical system identification faces, which was proposed in Chapter 2.

Firstly, the experience is always obtained in a specific context during system identification. This implies that the concrete structure of the system is implicitly compiled into the rule model. Moreover, in addition to the context, the application of knowledge to a specific task (diagnosis) is also captured. The resulting model fragment becomes not versatile and reusable in altered conditions due to the underlying structural properties enclosed in each rule. This is an undesirable attribute of the modelling paradigm since that same fragment of knowledge may be causally related to other events. Therefore its use in different tasks would be impactful in terms of problem-solving.[26]

Furthermore, in critical systems (e.g. during FMEA[1]), it is vital to identify all hazardous states leading to failure. That objective immediately implies two problems:

1. **Required coverage.** Typically, during the identification of relatively complex systems, the number of observations required to discover all relevant states is undeterminable. Even if it was known, observing each state would be impractical due to the high combinational complexity of the possible independent scenarios.
2. **Avoidance of certain scenarios.** In a case where only hardware-in-loop system simulation is available, the observation of certain states that result in severe damage or even destruction of the system cannot be afforded (e.g.: nuclear plant).

To overcome the mentioned obstacles, model-based systems consist of such **collection of models that are versatile and reusable** for several distinct tasks, while also providing **model-based problem-solving engines** to automate certain procedures. Hybrid modelling (Chapter 2) completely entertains this methodology, by maintaining the proposed model fusion (Figure 2.1). The **domain-specific knowledge** can be acquired from the engineering model, whereas the **task-specific knowledge** is either inferred from the qualitative knowledge-base (model fragments) or reasoned by automated problem-solving methods (ML).

One significant subfield of model-based problem solving is *Diagnostic theories and systems*, which is an essential concept in numerous areas of applied artificial intelligence research. Medical diagnostics, for instance, is a particularly active area of research, which aims to identify the associations between models of disease development and the manifestation of disease symptoms.

3.2 Taxonomy

As discussed, the fundamental goal of machine learning is to generalise the given dataset, discover common behavioural patterns and relationships to make a decision in connection to a specific objective (problem). The algorithm achieves that by generating accurate predictions for yet unseen entities in an efficient manner. Since the concrete process of learning is heavily dependent on how the task to be solved is formulated and how the initial data is structured, a more detailed description will be given in the subsequent subsections. In this introduction, basic terminology and general concepts will be summarised.[23]

The dataset, as the input of the learning algorithm, is stored in relations. Each relation consists of observations (*samples*) on the environment, each represented as a row of the relation. Each observation has attributes (*features*) that describe specific states of the

[1] failure modes and effects analysis

environment related to one object (to the sample). Features can be **interval** (numerical values), **categorical** (values taking on a discrete domain, but orderliness cannot be defined between values), and **ordinal** (values taking on a discrete domain, but orderliness can be defined) variables.

Since data is usually acquired and aggregated from sensor data, erroneous data (e.g., missing values, incorrect values) can always be present in the initial dataset due to instrumentation, network or human error. Therefore the learning phase is always preceded by **preprocessing** of the data. This process also varies based on the problem formulation and the learning scenario but usually consists of the following steps: *extract, transform, load (ETL), data cleansing, outlier detection, feature selection, initial exploratory / confirmatory data analysis (EDA / CDA).* The goal of the process is always to make sure that the values of the samples comply with not only the feature's domain but also with the system constraints and to establish an initial hypothesis. The relevant preprocessing steps will be reviewed in detail for the proposed work.

A deep understanding of the data, domain and the objective, and the data scientist's experience guides the selection of a learning algorithm that well suits the problem. Hyperparameter tuning is another critical step run prior to learning in order to select a set of optimal **hyperparameters**[2] for the algorithm. Subsequently, the ML algorithm learns the provided data, resulting in a generalised inner model (a form of knowledge representation) that can predict other unseen samples from the same problem domain (reasoning). The learning process stops if the model is good enough, determined by the evaluation phase based on specific metrics, or the model cannot generalise itself more or else it would be overfitted.

The conventional ML model building process ends here, however a crucial part of this thesis is incorporating AI explanation methods into the model verification and validation phase (V&V). These tools provide algorithms and proxy metrics to inspect the insights of the constructed models by understanding how each prediction was made.

3.2.1 Learning scenarios

Prior to the learning phase, the data is partitioned into three distinct subsets: **training, testing and validation data**, according to each learning phase where it is going to be used. It is necessary since it is crucial for the learner to be validated on samples that were not accessible during learning.

Learning scenarios mainly differ in (1) the data itself available during learning, (2) the method and order how it is provided for the learner, and (3) in the evaluation of the constructed model.

- **Supervised learning**
  A supervised learning method is applied when the solution for the problem is also provided for a set of objects, and this behaviour that each observation - solution pair encloses should be generalised by the learner for all objects of this domain. Here samples are labelled, since each of them consist of **features** (input), and a **prediction** (output / target variable). The learner aims to approximate the mathematical function between features and the prediction if a substantial relationship is present between them in the modelled environment. During the evaluation, the learner needs to predict the target variable for unlabelled samples, and the predic-

[2]External variables that tailor the resulting model and aim to improve its performance or accuracy and are set by the programmers.

tion is cross-validated with the original labels. Prime instances of this method are classification and regression.

- **Usupervised learning**
  This scenario is typically used when an initial hypothesis is required to be established by the learner since there is no concrete aspect based on which the data points would be separated. Therefore the provided samples are unlabelled, and the algorithm is only reliant on the hidden underlying connection between groups of samples that the features enclose. Since there are no target variables, the evaluation of the learner is only feasible in the means of statistical analysis. Prime instances of this method are clustering and dimension reduction.

In practical use, many further somewhat more complex learning scenarios are imaginable, such as *active learning* (where the learner itself seeks for data by querying the environment), *reinforcement learning* (where the learner only receive a reward right after it explores and interacts with the environment), etc. However, in the scope of this work, only the mentioned two scenarios are relevant.

3.2.2 Learning problem types

ML tasks can be divided into segments, depending on the continuity of the output variable or whether there is any.

- **Classification**
  In classification tasks, only a finite number of values can be assigned to the prediction as categories. The cardinality of the predictor variable determines whether the given task is a *binary* or *multi-class* classification problem. Usually, the output of an ML algorithm only provides a class-length tuple representing the probabilities for each class. The sample is labelled after the class with the highest probability.
- **Regression**
  Algorithms using regression try to approximate a continuous property (from an infinite domain) of a sample as a prediction. Accurate predictors provide a low deviation from the original value.
- **Clustering**
  The algorithm aims to partition samples into distinct regions, where the members of each region show similar observed properties or behavioural patterns. Here the assumption is that similar data points lie close to each other (and the level of similarity can be approximated by the distance from the cluster centre point). Since clustering expose operational domains and boundaries of a system while discretising its continuous properties, this type of learning aligns well with the paradigms of qualitative modelling and will be thoroughly analysed in later chapters.

3.3 Validation

System verification and validation (V&V) provides a collection of procedures that ascertain that all system components comply with system requirements and specifications while it fulfils its intended purpose. Since scalability is a significant aspect in the CPS domain, analysing the performance of ML models is a research area that has been developed for a long time and is being assisted with the massive repertoire of statistical tools. This

section deals with this subfield of V&V, *AI model evaluation* to introduce various metrics and concepts for measuring the quality of the fitted model.

3.3.1 Evaluating classifiers

All classification related metrics can be derived by inspecting the four possible outcomes that can occur upon a prediction of a binary variable ($Y \in \{Positive, Negative\}$).

- **True positive prediction**: The model correctly predicts a positive sample.
- **True negative prediction**: The model correctly predicts a negative sample.
- **False positive prediction**: The model incorrectly predicts a negative label.
- **False negative prediction**: The model incorrectly predicts a positive label.

These outcomes can be summarised for all samples in the training dataset and depicted in tabular form, resulting in the **confusion matrix**. Notice that the matrix can be easily extended to plot multi-class classification predictions.

CONFUSION MATRIX		MODEL-PREDICTED CLASS	
		Positive	Negative
ACTUAL CLASS	Positive	# True Positives	# False Negatives
	Negative	# False Positives	# True Negatives

Figure 3.2. Confusion matrix of a binary classification problem.

One of the most easy-to-grasp and essential characteristics of a model, which can be read from the figure, is **accuracy**. By definition, it is the *ratio of correct predictions to the total number of input samples*. However, examination of more traits is necessary to judge a model truly, since accuracy will yield misleading results if the data set is unbalanced[3].

When it is vital to know how confidently a model predicts a specific class, our attention should be aimed at the concepts of **sensitivity** (also called recall or true positive rate) and **specificity**. Sensitivity measures the *proportion of correctly identified positives relative to all positive samples*. (Whereas specificity uses the same concept only for negatives.)

$$sensitivity = \frac{true\ positives}{true\ positives + false\ negatives}. \tag{3.2}$$

If a particular class is relevant in terms of the given problem (e.g., considering an event relevant in the presence of a disease), then an important issue may be the relevancy of the positively retrieved samples.[4]. Relevancy is expressed by the metric called **precision**, which is *the fraction of true positive (relevant) instances among the retrieved positively predicted instances*.

[3] when the numbers of observations in different classes vary greatly

[4] To put it simply, if the positive samples predicted by the model are evenly distributed among both the positive and negative actual classes, then the model does not bear any new information when the sample is positively predicted.

$$precision = \frac{true\,positives}{true\,positives + false\,positives}. \tag{3.3}$$

With the close examination of these metrics, models can be fine-tuned to fit any problem formulation. Let us take task exploration of Covid contacts as an example. This problem requires a model, which has a high *recall*, because misclassifying cases that involve connection with a person who is actually infected with the virus are not allowed. On the other hand, the model can allow a lower *precision*, because increased precautionary measures will not result in a greater spread of the infection.

The combination of the metrics mentioned above, resulting in a single value that describes the quality of the model's approximation, is the **F-score**.

$$F_\beta = (1 + \beta^2)\,\frac{precision * recall}{(\beta^2 * precision) + recall}. \tag{3.4}$$

Here a more generalised version of the F-score is shown, expressing the importance of precision / recall in our model with beta.

3.3.2 Evaluating clustering methods

Prior to the evaluation of the quality of cluster borders, an important step is to ensure whether the provided dataset shows **clustering tendency** - real relationships are present in the modelling environment between samples with low intra cluster distance. The clustering is infeasible if points in the data space are distributed uniformly. One way to show clustering tendency can be done by *Hopkins statistic test*, which assumes that the given dataset was generated by uniform data distribution and then verifies this null hypothesis. [24]

After the clusters are formed, measuring the algorithm's performance depends on whether ground-truth labels are available for the samples.

In a *supervised learning* settings, the cardinality of the target variable (number of clusters) and the membership function are present, therefore the separation quality can be assured with the following metrics:

- **Completeness**: measures the portion of all samples that are assigned to a given class, are also elements of the same cluster.
- **Homogeneity**: measures the portion of all samples that are in a specific cluster, are only assigned to a single class.

In an *unsupervised learning* setting, only pure statistical tools can try to approximate the intra- and the intercluster distance and measure the similarity among samples belonging to the same cluster. Some methods that show an extensive practical use are *Silhouette Coefficient* and *Davies-Bouldin Index*.

3.4 Highlighted learning models

This section showcases ML models that play an important role in the work presented in this thesis. In addition to their core learning concepts, their interpretability factors are examined as well. Their behaviours and performances are presented on the *Phone Price Classification* problem. (Section A.1.1).

3.4.1 Decision tree

A decision tree is an essential ML model, generally applied as a visual and analytical decision support tool. It has a flowchart-like structure, consisting of conditional control statements, that guides the understanding of what factors each decision is based on. Thus, the algorithm becomes **directly interpretable**, which is one of the major aspects why this tool is presented here with strong emphasises.

Each intermediate node of the tree represents a logical literal that splits the data (starting from the root), resulting in leafs as a collection of homogenous samples in terms of initial classes. (With a slight alteration, this model can be used for regression tasks as well). Eventually, a specific data point follows a particular path when making a prediction.

For data splitting, this ML algorithm utilises the concept of *Information Entropy*. Upon constructing the tree's hierarchy, a node at a higher level should always provide more information gain (change in the entropy) than its children. Therefore, the classes are separated from each other as much as possible, as in we go down the tree. Note that since the evaluation of a split happens locally, the algorithm that constructs the tree is *greedy*, therefore the model is not necessarily optimal.

It is discernible that the level of leaf-wise homogeneity (*purity*) - viewed as the accuracy of the classifier - can be arbitrary high depending on the number of newly introduced splits. However, as the tree grows, the model can be exponentially complex, compromising its interpretability, while it also tends to overfit the data.

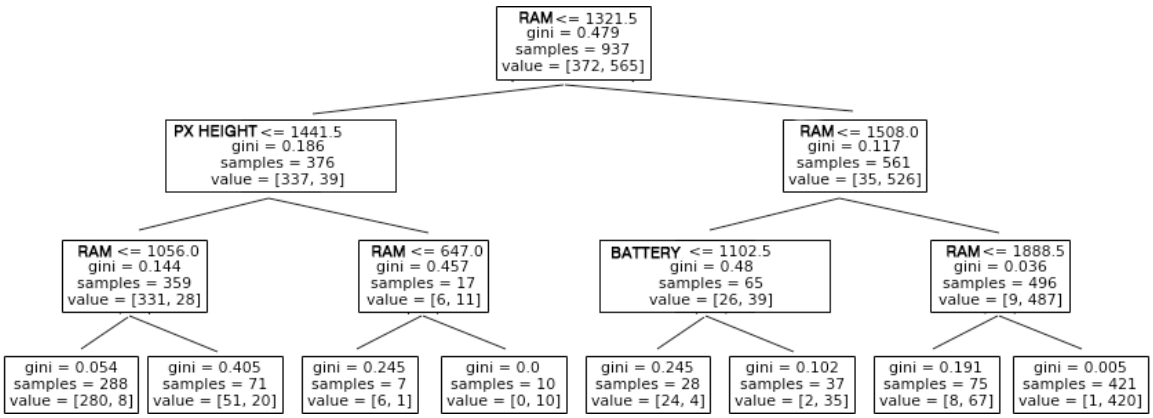


Figure 3.3. Decision tree model fitted to Phone Price classification dataset.

Figure 3.3 depicts a decision tree classifier that generalises the Phone Price classification task. At each (internal) node, the decisive conditional statement, an information entropy-like metric (Gini index), the size of the subset of the data that is split, and their separation according to their class labels. While the leaves are still impure because the tree's depth is constrained at 3, the classifier performs well. Note that a more complex tree would be harder to visualise and conceptualise for a non-expert.

It is also apparent that this representation is indeed interpretable even without external AI interpretability tools. It can be deduced that (according to the model) a phone's *RAM, battery, and screen size* are the most influential features that justify its price since splits on these features provide the most information gain and separate the samples the best. This knowledge already forms an initial hypothesis for feature engineering.

Decision Trees can also be viewed as *hierarchic rulesets*, therefore it shares many aspects of this knowledge representation tool. Due to this fact, a tree can be linearised into decision rules by selecting the terms on the path to each leaf and merging them into a clause as a conjunction of those terms. Therefore this method can be combined with other logical or decision techniques. The extracted ruleset from the classifier presented in Figure 3.3 can be found in Section A.1.2.

3.4.2 Linear tree

Model trees introduce a slight alteration to conventional decision trees in order to overcome their disadvantages (lack of robustness, tendency for overfitting) while retaining all of their preferential properties. As learned, decision trees use **simple constant approximations** in leaf nodes to evaluate the goodness of the splits. In model trees, these are evaluated by **fitting models**, a more specific instance for it a linear tree that utilises linear models.

To put it simply, a well trained linear tree does not necessarily have a homogenous collection of samples in the leaves. Rather than upon prediction of a selected sample, it firstly follows a certain path (where the logical terms are satisfied), then the final split is done by a linear model.

As a consequence, the model encapsulates not only a specific value, rather than the **relative gradient** (changes) of each feature value as well. (Accordingly, linear trees enclose the first derivatives.) This aspect results in a more robust model that better handles shifts in both feature and target distribution over time. [5]

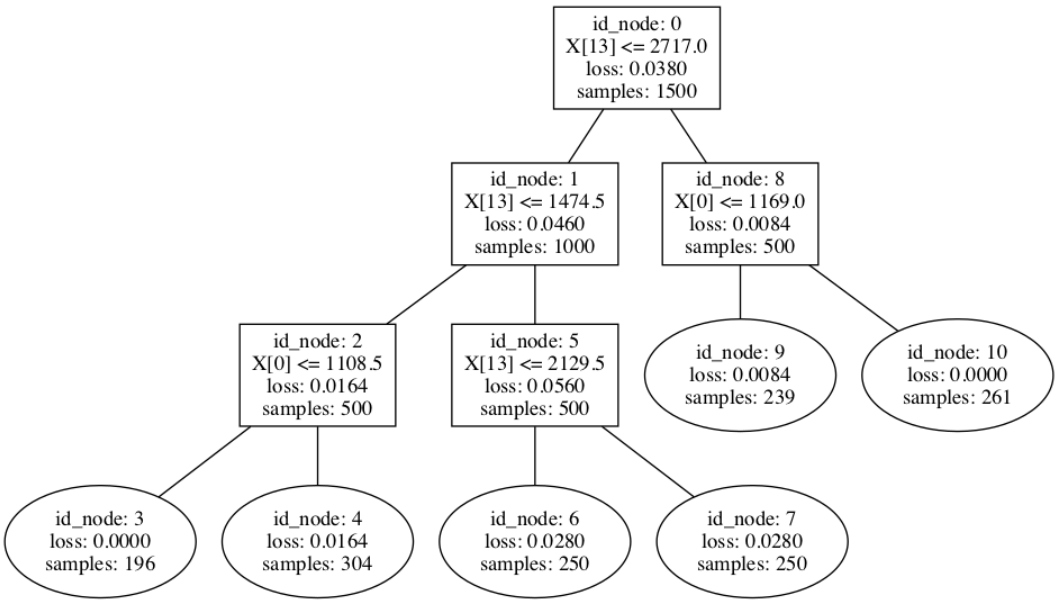


Figure 3.4. Linear tree model fitted to Phone Price classification dataset.

[5] https://towardsdatascience.com/model-tree-handle-data-shifts-mixing-linear-model-and-decision-tree-facfd642e42b

Comparison

Model	CPU Time	F-Score	Complexity
Linear Tree	2.6261	0.974	142
Decision Tree	0.0061	0.914	24

Table 3.1. Performance comparison between Decision & Linear Trees.

Figure 3.4 depicts a linear tree classifiers on the same problem as the example above. **Table 3.1** shows statistics on the performance both of algorithm (Section A.1.2). To sum up the linear tree has generalised the model at a much higher level than the decision tree. Regarding the huge difference in computation time, it is presumably caused by poor implementation decisions, since a combination of a few linear models cannot be that computation demanding. For the model construction `scikit-learn`, and `linear-tree` (published by an individual[6]) packages were used for the decision and linear tree model respectively.

In the picture, the first major difference lies in the apparent simplicity of the linear model. Both trees' depth was constrained at 3, this way, the linear model produced two fewer intermediate splitting nodes than the Decision tree.

On account of the leaf-wise linear models, the resulting tree is simpler than the previous one and performs marginally better. The problem with conventional decision trees is that it only can construct a single comparison on a selected feature in each node (that is why the feature RAM is present many times on fig. 3.3), whereas a fitting model can conceptualise several at once, resulting in a simpler structure. However, the difference in complexity is truly apparent since inner linear models' complexity located in each leaf is proportionate with the dimension of the dataset, since every feature is considered with its respective coefficient in the prediction.

Direct interpretability of the model is also not compromised - but definitely lower than decision tree's - since a linear model is still comparably interpretable as the exponentially growing clauses by the introduction of multiple criteria on features high in the hierarchy. Surely, substituting the linear model for a more complex one would truly damage the interpretability factor of the model.

3.4.3 Neural network

Problems related to such domain fields arising from the ICT[7] sector dealing with high dimensional, unstructured data[8] (e.g., image processing, natural language processing) cannot be efficiently solved by using simple ML models (just like the ones mentioned above).

Neural networks (NN) mimic the nature and function of the brain since they are composed of an extensive amount of *neurones* as processing units. Arbitrary neurones are structured into layers, and elements of each layer are interconnected with neurones from adjacent ones. [1]

A single neuron receives values from other units (with which it is associated) and computes an output propagated forwards to other nodes. Each input is multiplied with a specific weight, which assumes the relative importance of each node. Each neuron firstly summaries

[6]https://github.com/cerlymarco/linear-tree

[7]information and communication technology

[8]in conventional terms, where attributes of objects are represented implicitly as relations

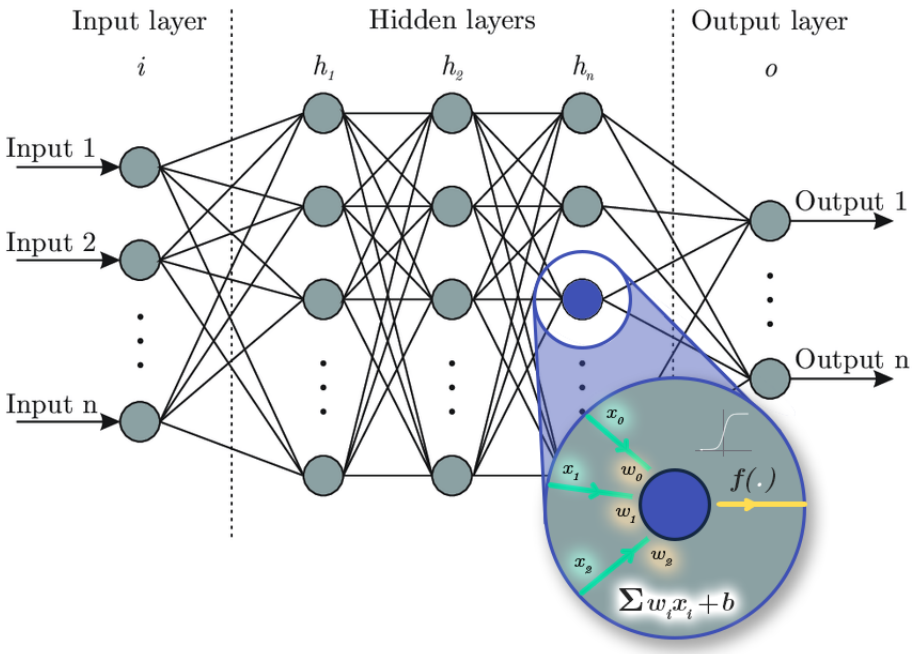


Figure 3.5. The structure of a Neural Network.

the incoming weighted values and adds a custom trainable constant (*bias*), then computes the same *activation function* on the result. The activation function is an easily computable, non-linear mathematical function that constrains the result between 0 and 1. In practice, the following functions are used: *Sigmoid, ReLU, Softmax, etc.*

The goal of the training is to learn correct weights, with that as a collection, the network can **conceptualise a very complex non-linear mathematical function** that maps the input layer to the output layer. The learning takes place by calculating an error function in each node and propagating (either backwards or forwards) between layers that acquired information on how the weight should change to minimise that function.

The fact that layer-wise computations are independent implies the massive parallel implementation of a network which leads to high-speed processing, complementing its immense accuracy.

Since layers can contain thousands of neurones - without regards to the possible number of layers - understanding the concrete decision-making mechanisms of a specific model by just simply trying to understand its parameters is an infeasible task. Neural networks are considered the least interpretable ML models, and their understanding has been a great challenge for researchers so far.

3.5 Challenges

In this section, a couple of concepts in ML will be reviewed that require special attention by the data scientist during model creation to ascertain the quality of the models.

3.5.1 Outlier detection

Even if the measurement data received from a sensor is syntactically correct, its value may be unrepresentative of the modelling environment, erroneous data. Incorporating faulty

samples may degrade the measurement result, thereby distorting the distribution of the data model so that the system may draw erroneous conclusions during the subsequent data processing phase.

A data record that is significantly different from the model of the examined dataset is called an **outlier**[9].

Causes of origin

Examining outliers upon system identification is vital, as it can carry underlying information about the abnormal behaviour of the system or the environment being tested. Although such values can be generated in several ways (human error, instrument error, data processing error, sampling error, intentional or experimental error, natural error), the following two groups should be distinguished in terms of origin: [18]

- **errors**: frequently occurring invalid data, acquired from a faulty component (sensor),
- **phenomena**: extreme, uncommon, but faithful data in the presence of external environmental impacts.

It is important to note that the data cleansing phase outlined in this work will filter out any outliers from the model of any type. However, we should only do so in the event of an *error*. It can be observed that the data caused by the *phenomena* correlate in time and space. Thus, data records marked as outliers should be re-evaluated in a separate refined procedure to avoid measurement bias. In this case, data that do not show spatial (generated by sensors close to each other) or temporal (generated almost simultaneously) correlation can be considered *errors*.

Types

Depending on the dimension of the context in which a selected sample is examined[10], **univariate** or **multivariate** (collective) outliers can be defined. In the case of univariate outliers, **global** (the data record simply immensely deviates from the distribution) and **contextual** (the data record cannot be interpreted in the given context and differs from the common patterns) outliers can be distinguished.

Detection

The most fundamental outlier detection (OD) techniques are *proximity-based, projection-based* OD methods. [8]

Proximity-based methods firstly define a metric that measures the distance between points between the data space, then try to find sparse regions in which isolated data points lie far from the rest of the sample. Usually, data points are considered isolated if there are no other datapoint within a predefined radius. If the data were clustered beforehand, this metric is already provided, and outliers can be easily detected using the cluster membership function.

[9] Barnet and Lewis (1994): an outlier is an observation that appears to deviate markedly from other members of the sample in which it occurs

[10] whether a single variable or a combination of multiple variables show extremities.

Notice that if the distance metric is derived from considering all features, the method can only detect global outliers. Multivariate outliers can be discovered if the metric is computed only from a subset of the features, resulting in a so called *varied density*.

Projection-based techniques however rely on projecting the entire dataset into a lower-dimensional space (just like dimension reduction methods), which is structured in a way that OD can be carried out with much more efficient algorithms rather than computing pair-wise distances in quadratic time in the original data space.
The same concept applies for detecting multivariate outliers, it is feasible if only a subspace of the dataset is projected.

It is worth mentioning, *machine learning assisted* methods are also available, however, supervised outlier detection relies on labelled training data. These are often cannot be obtained in practical use since the primary objective is to identify anomalous patterns that were never encountered before.

3.5.2 Overfitting

During a model's training, the objective is to conceptualise a complex mathematical function between the in- and the output. However, there is a major difference in approximating and generalising this function or just memorising all the examples. The cardinal idea behind partitioning the obtained dataset into train - test - validation set is to evaluate the model on yet unseen items and see if it performs accurately on them as well.

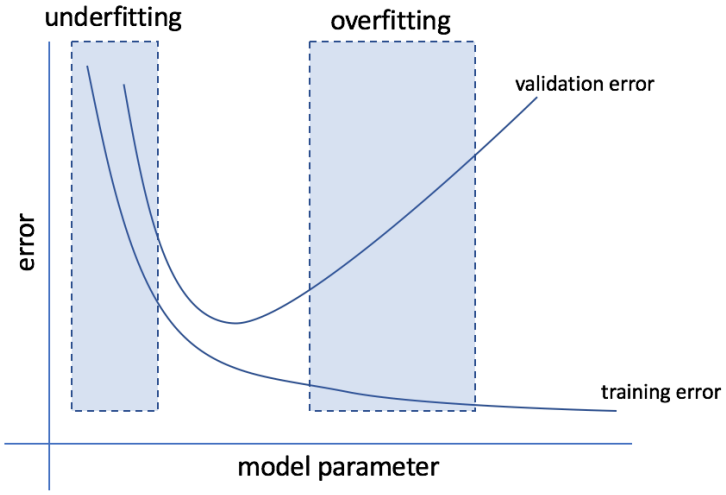


Figure 3.6. Typical validation curve representing lifecycles of the training phase.

Figure 3.6 shows a typical validation curve which provides an intuition when should the model fitting stop to avoid overfitting, signalling the model starts to lose its generalisation skills. It happens precisely when the model performs well on the training set but fails on the test set. Underfitting showcases the exact opposite phenomenon when the model has not learned the function yet.

A validation set is used to fine-tune hyperparameters to avoid finding a set of parameters that only perform well on the test set.

Chapter 4

Explainable artificial intelligence

In recent years, as the emphasis has grown on integrating machine learning concepts into corporate projects, the need to explain opaque models has also expanded significantly. The reason behind that can be influenced by several factors. Corporates have both legal and moral obligations to explain their decision-making process in detail arising in situations of great responsibility, which can frequently occur in regulated industries such as medicine, criminal justice, and business. Therefore, currently explainability stands in front of the barrier of the widespread application of AI.

Explainable artificial intelligence (XAI) is a field that provides various tools, techniques, and methods to explain complex ML models by making them directly interpretable. Explainers serve to highlight the main features of data sets and models derived from them. As it is currently a rising field, there are many open-source interpretability libraries available for developers, in this thesis two of them will be previewed with great emphasises. Since the topic of interpretability can be approached in various ways, a different library/algorithm is provided for each approach.

On the other hand, XAI also helps data scientists understand and improve their workflow and infrastructure fundamentally. Paramount contribution of the conducted research relies on utilising the rapidly growing repertoire of XAI in system identification - more precisely, in *qualitative state analysis*.

4.1 Core XAI concepts

4.1.1 Primary objectives

Explainable Artificial Intelligence is the key for AI deployment in a wide range of corporate environments. It is vital to firstly understand the variety of expectations and obligations are set for an explainable AI model to know how to implement Explainers for V&V purposes.

These intentions are essentially distinguished based on *target audience profiles* seeking different explainability purposes. **Domain experts** or **developers** have the ambition to trust the model, extract scientific knowledge and verify its functions with unique constraints. Functionalities only relevant to their point of view will be considered in the further discussion of XAI assets. Four significant concepts cover these needs:[5]

- **Informativeness**. This criterion is fulfilled when an XAI tool can describe the decision-making process of an AI model. XAI method's informativeness will be examined mostly in this thesis.
- **Trustworthiness**. This concept is closely related to V&V procedures, as it conceptualises the confidence in the model to follow environmental/system requirements at each individual decision. Recognise that most of the further showcased XAI tools only provide an approximate description of the behaviour, but that hypothesis is never necessarily enforced by a mechanism in all scenarios. A prime example for such a mechanism would be a Physics-constrained NN - previewed in Section 4.4.2. However, in the operational critical domain it is essential not to brake the laws of physics (for instance). This aspect justifies the use of an AI explainer as an extension in the proposed hybrid model (Figure 2.1). Such an external component would validate the model's actions with the set constraints.
- **Confidence**. Uncertainty emerging at each prediction quantifies the confidence profile of a model. The model's accuracy as well as the quality of the predictions can be viewed differently when uncertainty is taken into account. Decisions with low confidence could foreshadow potential adversarial perturbations or other ambiguous states that could be a subject of risk analysis.
- **Causality**. Another typical intention of XAI is to infer certain causal relationships between variables or - in an even more complex scenario - between events from observational data (e.g., as sensor fault is inferred from temporally correlated outliers). Even though correlation analysis plays a huge role in the exploration of causal relationships, one cannot be simply deduced from the other, due to the lack of directionality and other intermediate factors.
  Figure A.2.1 depicts the results of a correlation and a causality analysis on the Phone Price Classification dataset, and illustrates the difference between the two concepts. The results lead to the second conclusion, that the model can only discover relationships that can be learned from the data, which is not necessarily representative of the environment.

Managers and **regulatory forces** on the other hand are inclined to verify the model's compliance legal obligations. The following points raise marginal issues[1] in ethical AI communities that legislation forces need to take into consideration:

- **Fairness**. Explainability serves as an asset to avoid unfair or unethical use of the model's output, as it ensures impartiality by detecting and counterbalancing bias in the dataset. Fairness is not only a concern in applications that affect human lives (e.g., race, gender bias), rather than bias towards any class compromises the quality of the model. Notice that by distinguishing correlation and causality, a certain source of bias can already be eliminated.
- **Accessibility**. In some applications (e.g., credit administration) it is important to make the reasoning behind a certain decision available to the end-user.
- **Privacy awareness**. If a model is not explainable the structure and the detail of the stored information is unknown. Despite of strict GDPR legislations, ML algorithms can reveal private relationships or knowledge regarding entities that could lead to a privacy breach. On the other hand if sensitive information must not be obtainable by a third-party via XAI methods.

[1] but less relevant in terms of the research work, therefore it will only be discussed in a supplementary manner.

4.1.2 Transparency as the metric of interpretability

Clarifying the definition

Quite a few literature on the topic uses both term *interpretability* and *explainability* interchangeably. It is important to clarify the underlying concept behind each term respectively, as there are remarkable differences.
Interpretability can be expressed as a descriptive attribute of the model, which mirrors how much sense it makes when perceived by a human observer.
Explainability on the other hand, is a characteristic that refers to the model's intention to clarify and analyse its internal functionality.

For instance a decision tree with a low depth is considered interpretable, whereas a Neural Network that provides ruleset in addition to a prediction would be an explainable model.

It is clear that understanding a model is a double-edged concept: it relies not only on the contained knowledge that the model made accessible to the user, but also on the capacity of the observers to process it.

Interpretability spectrum of models

The previous section might give the intuition to some, that firstly, there are models that are interpretable by themselves. These models are called **transparent**, and they are directly interpretable.
Characteristics of a transparent model: *linear, well-structured relationships, long computation time, moderate accuracy.*

Decision or *linear trees* are perfect illustration for transparent models. Its visual representation and telling structure gives a thorough overview over the decision making process even for an untrained eye. Moreover, features at a high level in the hierarchy reveals details about the model's *feature importance* metric.

Notice that even the understanding transparent models can be further elevated by XAI methods. Here are couple of examples:

- To get a grasp of a decision tree's classifier function rulesets need to be extracted at first. However when these clauses are compound and arbitrary complex the task is not trivial.
- Even much more simpler linear trees can be demanding to interpret indirectly. Since the tree's decision path is not fully extracted, because the final decision is made by a fitted linear model in each path. The model is explainable, since the coefficients of the linear models are extracted and visualised to see their impact on the final decision.

Figure 4.1 shows a proposed method to explain the decision making process in a leaf (Leaf #9) of a linear tree - from Figure 3.4 - combining the concepts mentioned above.

On the right side there are logical statements that lead to Leaf #9 and the coefficients of the corresponding linear model are shown on the right.
This decision path describes that a sample is classified positive - a phone with a high price - if the ruleset and the $\sum_i c_i * f_i > 0$ expression[2] are satisfied.
Meaning in case of a phone that has $>$ 2717 Mb *RAM*, and a $<=$ 1169 Mah *battery power*,

[2] which describes the linear model, where c_i is a coefficient and f_i is a value of a feature

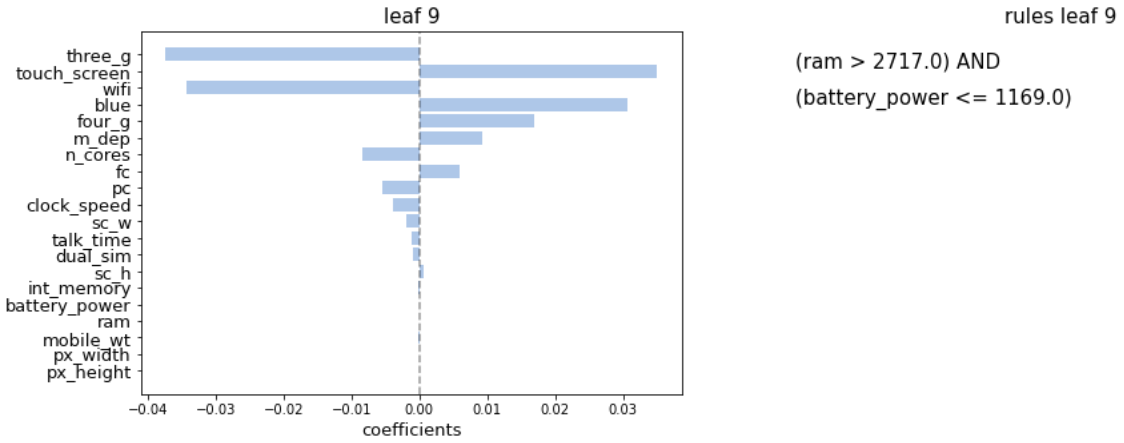


Figure 4.1. Extracting a decision path from a linear tree.

its attributes like *touch screen, bluetooth* positively, wheres having a *3G module* only, and *Wifi module* negatively influence the prediction - with the respective importances/weights.[3]

In addition there are also models that need external methods or metrics to even have the chance to understand their behaviours. These are called **black-box** models, referring to its opaque inner structure. Models in this category maintain a high level of performance and efficiency but as a trade-off generally operate with immense parametric space. Neural Networks are the first ones to come to one's mind when considering black-box models, due to hundreds of contained hidden layers.
Characteristics of a black-box model: *non-linear relationship, low computation time, high accuracy.*

Degree of transparency

The preceding paragraphs show that the level of transparency is varied amongst models, as some can be easily understood by a human observer (simple decision tree), while others can be quite more challenging (linear tree). Let us formalise this metric by describing three different stages of interpretability, dependent on human understanding capacity, and the detailedness of the extracted functionality.

The lowest expectation set for a transparent model is that its inner functionality is extractable, and it can be reasoned with mathematical means. A model is considered **algorithmically transparent**, if mathematical analysis provides an overview, representing how the model will behave for each individual input without making computations that are proportional with the algorithmic runtime of the model. (E.g., high dimensional linear model.)
In addition to this requirement if each part of the model is explainable - as well as its input and parameter space - then the model is **decomposable**. (A good counterexample would be preprocessed, tokenised sentences from the NLP field as an input.)
The greatest level of transparency is achieved when a model can be mentally visualised or simulated in a human mind (**simulatability**). Generally, models that can be expressed by written or visual methods fulfil this requirement. This aspect is closely related to model complexity, meaning that a single neuron falls into this category, while even a compound ruleset cannot be precisely thought through.

[3]Note that this shows only a certain decision path (out of 6), therefore this rule is not exclusive.

4.1.3 Validation-based automation

As discussed in Section 2.1.1, during empirical identification of a black-box-like system, it is ideal when environmental effects and the modelled system itself formulate the model boundaries. After all, it is often advised that data scientists should not force their initial hypothesis to a yet unknown system. The use of purely statistical models is also avoided for the same reason.

Therefore the subsequently proposed workflow aligns with the current ongoing trend to shift from the extensive domain knowledge, (computationally heavy) construction, and fitting of models towards the V&V and interpretation of the model. I believe that introducing increasing (semi-)automation to the modelling part leads to a better understanding of the analysed data. All this is achieved by transforming model extraction into an iterative process, and inserting a feedback loop that propagates information from the validation stage towards domain exploration - as Figure 4.2 suggests. Resulting in the conversion of exploratory data analysis (EDA) to confirmatory data analysis (CDA), which aims not only to refine the initial hypothesis, but also elevates the quality of feature selection, outlier, risk, anomaly detection and eventually excludes the bias from the dataset. This summarises the motivation behind the application of XAI tools in model building approaches. [7]

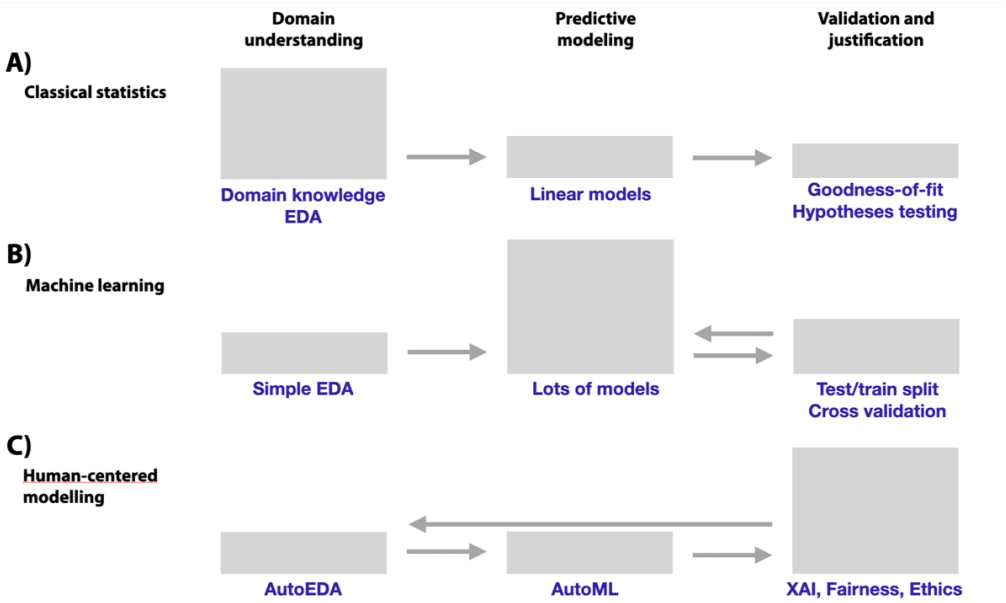


Figure 4.2. Importance of domain knowledge, model fitting validation in different stages of ML.[7]

Figure 4.2 summarises how the increase in the model complexity affects the relative importance of domain understanding, the choice of a model, and model validation. In classical statistics the biggest emphasises was on thorough domain understanding, and the quality of the models were solely reliant on preliminary statistical hypothesises, which must be stated prior to model construction. In the early days of ML, however, the focus has shifted towards the building of better performing ML models, that came with immense complexity as a trade-off that compromises interpretability.

4.1.4 Taxonomy

After understanding the definition and major motivations of interpretability, this section attempts to overview the ways *how* XAI tools provide explanations.

The first notable difference is based on whether the entire **dataset** or the **model** derived from it will be the target of explanation. Notice that some model explanation tools reviewed later (e.g., BRCG) also relies on data as their input, and it is not implemented as an extension of the model that needs to be explained. Here the samples should be relabelled according to the model's predictions to fully represent the model's behaviour.

These tools can also be partitioned whether the given algorithm is only suitable for interpreting a specific model (**model-specific**) or an arbitrary constructed model (**model-agnostic**). Model-specific tools generally specialise in explaining NNs, due to their particular structure and impact on ML. On the other hand model-agnostic tools provide general visual aids and proxy metric to to highlight the main features of the models, while treating all models equally, as black-boxes.

Depending on whether the target model already carries internal qualities that allow the extraction of knowledge **intrinsic** or **post-hoc** algorithms can be used. Intrinsic tools relies on model that was trained in a controlled process to develop this trait (just like in constrained NNs, or already transparent models) to reduce complexity. While in post-interpretation, the explanatory factors are extracted after teaching.

The scope of interpretation also separates the XAI tools into two category: **global** or **local** (instance-level) explanations. The interpretation method can either explain an individual prediction or the entire model behaviour.

Eventually, notable differences can be recognised in XAI tools based on the **outcome of the interpretation method** (that can be feature statistics, feature visualisation, model internals - e.g., learned weights).

Established practical use

So far, many open-source explainable AI libraries are available, including a comprehensive set of algorithms that cover different dimensions of explanations and proxy explainability metrics. There are three main general approaches targeting black-box models along which explainability tools are implemented, for each purpose a Python algorithm package is accessible.

1. *Constructing a directly interpretable surrogate model.*
   The surrogate model mimics (or at least approximates) the behaviour of the black-box model, while it remains simplified and directly interpretable. Surrogate models can be derived from black-box models with the variety of IBM's AIX360 framework.
2. *Using post-hoc model-agnostic techniques.*
   For this general purpose visualisations, several proxy explainability metric will be reviewed, as the most natural way to understand complex system. DALEX library contains these required tools.
3. *Using intrinsic model-specific methods for NN explanation.*
   Due to the high performance and impact on the scientific field of Neural Networks, they were always treated with special attention, but also with little success until recently. A couple of ongoing researches achieved a breakthrough in explaining NNs, which greatly influenced the field of XAI on the whole.

Necessity of a black-box model

One final important take away of this section is that black-box models does not provide the highest performance amongst ML machine learning models in an arbitrary learning scenario. On structured relational data a gradient boosting algorithm (xGBoost) outperforms any Neural Network in accuracy and learning time as well, while providing a much more interpretable model as well. Also for purposes like demonstrating a concept with a prototype a standard interpretable model is adequate. Therefore there are some cases when the application of a black-box model is neither necessary nor preferable. Explainability begins with choosing the right model that suits the specific problem.

Demonstration the use of XAI methods

In the next sections all three explainability approaches will be examined - consisting algorithms that play a primary role in this research - and also demonstrated on a black-box model (Section A.2.1). The topic of the tool's suitability for analysing qualitative abstractions of AI models will be separately addressed as well in each section.

As a black-box model a recurrent Neural Network were trained on the Phone Price Classification dataset, consisting 4 linear hidden layers with 120 neurones per each, and connected with several different activation function. The model was constructed to be efficient, but also very complex, to represent a truely opaque model. More detailed description of the NN can be found in the appendix. The NN achieves 91.2% accuracy on the testing dataset at its peak.

To reflect to the previous paragraph a Neural Network was unsound choice to solve the given task since - as seen in Table 3.1 - a linear tree completely outperforms the NN in all aspects (performance, complexity, accuracy). The rationale behind this choice was the intention to showcase black-box level explainability techniques, while the problem remains transparent for the observer.

In Section 4.2 the input of the algorithms are tabular data, that represents the predictions of the black-box model. For that the purpose, the initial dataset was relabelled according to the predictions of the model, and provided for the explainability algorithm.

4.2 Constructing surrogate interpretable models

IBM was among the first ones, to publish an AI explainability framework in this capacity, with the intention to support the decision-making process for high-stakes applications. They made the framework available for the XAI community as a Python package.[4] Figure A.2.2 summarises the package, and provides use-cases for each algorithm.

These methods target a black-box model and build another directly explainable model on top of the opaque model. The newly emerged model is a simplified version of the original one, approximating its inner functionalities. Here it is crucial to highlight the approximation, as a marginal gap might exist between the two, which needs to be considered if used for V&V purposes. The surrogate model uses knowledge representation formats as discussed in Section 3.1.

Methods previewed in this section mainly focus the extraction of the core functionality and decision making instead of measuring and highlighting different traits.

[4] https://github.com/Trusted-AI/AIX360

4.2.1 Boolean Decision Rules

Boolean Decision Rules via Column Generation (BRCG)[12] algorithm provides an abstract representation of the underlying domain in the form of DNF rulesets.

Rulesets in general and all of their characteristics were described in Section 3.1.2, and terminology described there will be used. They are simple, easy to understand, and interpretation-friendly, as proven by many popular ML modelling paradigms.

The algorithm seeks to build models that are simple, yet highly accurate. At low complexity it outperforms linear models of the same complexity. However it performs poorly if a very precise overview of the system is required. This characteristic is shown in Figure 4.3, which depicts the model's trade-off between accuracy and complexity.

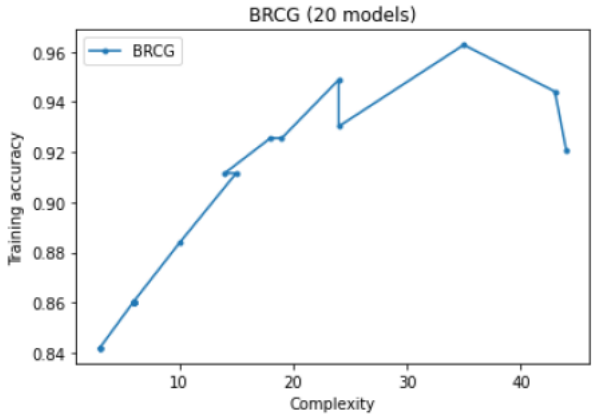


Figure 4.3. The algorithms accuracy in terms of complexity.

Functionality

The exact implementation of the algorithm will be detailed in the next chapter, since BRCG will serve as the foundation of a proposed explainable classifier model presented there. However a brief overview of it is given here.

The algorithm's core idea is the composition of the domain model from a set of (qualitative) subdomains, each defined individually by a decision rule.[5] The complexity of the resulting ruleset can be adjusted by c_1, c_2 parameters - limiting the number of clauses, and the number of terms in a clause respectively.

The algorithm reasons data points as its input, which can either be a *dataset* or a model represented as derived *observation - prediction pairs*. Firstly continuous variables in the dataset must be discretised. This introduces another layer of abstraction to the model. Unfortunately, the preprocessor of the algorithm quantises the dataset independently of the model's characteristics. That should be taken into serious consideration, when the explainer is used for analysing qualitative abstractions of a system. In this case, a custom discretisation must be done according to the qualitative landmarks extracted from the system. Otherwise a relatively high resolution is ideal in order not to influence the division of the domain. Since the newly constructed expressions - as a selected decision rule in the ruleset - will **form the qualitative boundaries**.Otherwise the model will provide suboptimal results.

[5]Decision rules here are regarded as conjunctive clauses of literals. E.g., $(\texttt{RAM} \leqq \texttt{1100}) \wedge l_2 \wedge l_3$ ...

The quantised properties of the dataset constitute the search space of the algorithm, from which a subset of literals (like `RAM >= 1100`) can be selected into the ruleset. At first an integer program (IP) is formulated on a restricted search space (consisting of only simple expressions (literals)). An optimal solution of the restricted IP ensures the model's high accuracy at low complexity. Subsequently, further candidates are chosen into the ruleset by beam search, which is a greedy algorithm. Since the entire search space can be exponentially large based on the dimension of the dataset, a beam search provides an efficient, but suboptimal solution to the problem. That aspect accounts for the low accuracy at higher complexity when compared to other classifiers.

A given label eventually will be predicted as positive in case the resulted ruleset is satisfied. That limits the problem formulation space only for binary classification tasks.

Demonstration

The algorithm in operation will be also demonstrated on the Phone Price Classification dataset - Section A.3.1.

Here using the default preprocessor satisfies the needs of demonstrative purposes. Without any adjustments it divides each feature's domain into 10 equally-sized subsets. To get an insight to the algorithm and illustrate the discretisation process Figure 4.4 shows the quantised properties (examined feature, value took on that feature, some of the discretisation thresholds, constructed logic terms/literals, and whether they are satisfied by the sample) of a selected sample.

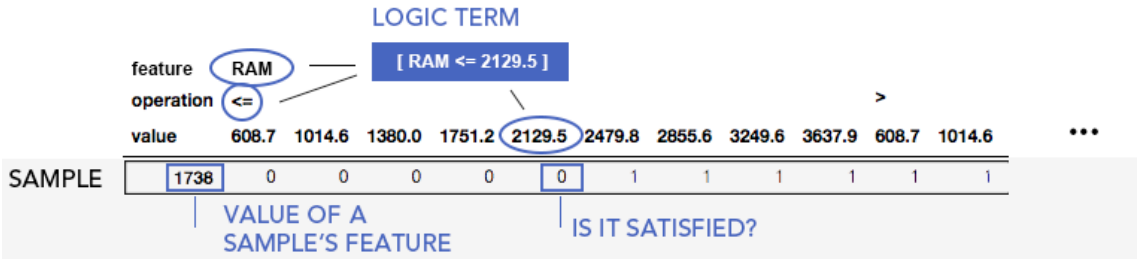


Figure 4.4. Properties of a binarised sample.

Below there is the acquired explanation in a form of a ruleset related to the problem, which explains directly the dataset itself for the sake of simplicity:

$$ram > 2479.80 \vee \tag{4.1}$$

$$pxHeight > 223.00 \wedge pxWidth > 1244.00 \wedge ram > 2129.50 \vee \tag{4.2}$$

$$batteryPower > 1403.00 \wedge pxWidth > 669.90 \wedge ram > 1751.20 \tag{4.3}$$

The explanation suggests if the RAM of a phone is *extra high* (> 2479.8), its should also be high. If its RAM is *high* ($2479.8 \geqq \text{ram} > 2129.5$), then the model takes into account the resolution, which also needs to be moderately high to classify the phone expensive. Eventually if the RAM is *medium* ($2129.5 \geqq \text{ram} > 1751.2$) then in addition to the resolution the phone's battery power also cannot be small. It can also be inferred by the explanation that if the phone's RAM is *low* ($\leqq 1751.2$), then the phone is immediately considered inexpensive.

Notice how naturally qualitative states appear in the model. Since each clause designates a specific subdomain, with similar operational behaviour, a new qualitative discretisation for

each feature is provided - as landmarks. For instance the RAM property of the illustrated model is divided into four distinct qualitative domains (low, medium, high, extra high), where the model requires different aspects to fulfil the task.

The resulted ruleset approximates the black-box model with 94.6% accuracy, and the complexity of the ruleset (using Equation 3.1, with $\delta_1 = 1, \delta_2 = 1$) is 10. Given these aspects BRCG provided a simulatable explanation of a black-box model that describes the model very accurately.

Further analysis of the ruleset

With the large repertoire of logic further analysis of the targeted domain is obtainable. Two proposed simple algorithms (attached in the Appendix) enhances the transparency of the resulted ruleset by visualising them as (1) decision trees, (2) binary decision diagrams (BDDs). While there could be many different decision trees built up from a given ruleset, BDDs remain unique (as long as the order of the variables in the rules are fixed), since BDDs are the canonical representation of boolean functions. That allows for easier observation of differences between two given ruleset. (Section A.3.1)

4.2.2 ProtoDash

ProtoDash[17] is a data explainability tool in the AIX360 framework. It relies on the concept of prototype generation.

A *prototype* is a single point of a subdomain, best representing the characteristics of the context containing it. These characteristic reveal cohesive and separation factors within the given context in contrast to the entire dataset. Since the algorithm can only reason the selected datapoint in a given context - by definition - generally, a preliminary clustering should precede its use. This way the algorithm can reason a single point in the entire cluster, if it is sufficiently homogeneous (similarly to an equivalence class leader in automata theory).

Confidence profiles can be extracted from the model, that shows how well the prototype represents its entire cluster. If this metric is high, then in addition to global explanation tools, an instance-level analysis on the selected point can provide more in-depth interpretation of the observed operational domain at its entirety. This is a crucial advantage of this AIX tool, that can improve the interpretability of any qualitative model.

Besides that, for a non-expert human observer this is a natural and well-interpretable first step to represent a group of samples with a single prototype. Oftentimes it is an important question whether similar examples lead to the same outcome. ProtoDash can illustrate spectacularly if an ML prediction follows the expected operation.

Figure 4.5 shows prototypes from both high and low price range classes in the Phone Price classification dataset. One could easily recognise that models with better specifications tend to be more expensive.

battery_power	1526.0
blue	0.0
clock_speed	1.1
dual_sim	1.0
fc	2.0
four_g	1.0
int_memory	38.0
m_dep	0.3
mobile_wt	116.0
n_cores	5.0
pc	6.0
px_height	798.0
px_width	1418.0
ram	3317.0
sc_h	7.0
sc_w	1.0
talk_time	8.0
three_g	1.0
touch_screen	1.0
wifi	1.0

(a) High-price prototype

battery_power	1954.0
blue	0.0
clock_speed	0.5
dual_sim	1.0
fc	0.0
four_g	0.0
int_memory	24.0
m_dep	0.8
mobile_wt	187.0
n_cores	4.0
pc	0.0
px_height	512.0
px_width	1149.0
ram	700.0
sc_h	16.0
sc_w	3.0
talk_time	5.0
three_g	1.0
touch_screen	1.0
wifi	1.0

(b) Low-price prototype

Figure 4.5. Derived prototypes from the dataset.

4.3 Knowledge extraction via model-agnostic techniques

Drawn from the statistical repertoire of mathematics, DALEX ModelStudio[7] framework[6] provides a unified model-agnostic toolkit to highlight the main characteristics of black-box model as explanatory metrics and visualisations. The framework interfaces with the most widely used implementations of machine learning libraries (scikit-learn, keras, xgboost, etc). ModelStudio can be used as an extension to an arbitrary black-box model built in these environments to provide a collection of local and global explainers. Since these techniques measure different model properties explicitly - rather than observing an intermediate model - the derived statistical models are entirely accurate.

4.3.1 Instance level explanation

Local explanations can come in handy when a data scientist wants to investigate a highlighted datapoint to understand how it differs from the average model prediction and why.

Properties of observed sample as a subject of analysis in each demonstration can be seen in Figure 4.6. Both prediction and original label for this sample is `negative`.

battery_power	blue	clock_speed	dual_sim	four_g	int_memory	m_dep	mobile_wt	n_cores
615	1	0.5	1	0	42	0.6	163	6

fc	pc	px_height	px_width	ram	sc_h	sc_w	talk_time	three_g	touch_screen	wifi
0	4	104	1664	2211	7	4	18	0	0	1

Figure 4.6. Properties of the observed instance.

[6] https://github.com/ModelOriented/DALEX/

Break down

Break down plots (or SHAPLY values) summarises all the features at a certain value with their respective **contributions to the prediction**.

Break downs are incredibly useful upon the construction of a *scoring system* related to the prediction (e.g., risk analysis). Therefore each particular instance can not be labelled with a tangible metric - massively increasing the interpretability of the system - but it also can also be broken down to subcomponents, reasoning the score.

They also can be applied in a refined second layer of the *outlier detection* process to distinguish phenomenon from errors. After filtering all deviated samples in the first stage, breakdowns can guide the understanding individual factors has driven the wrong prediction.

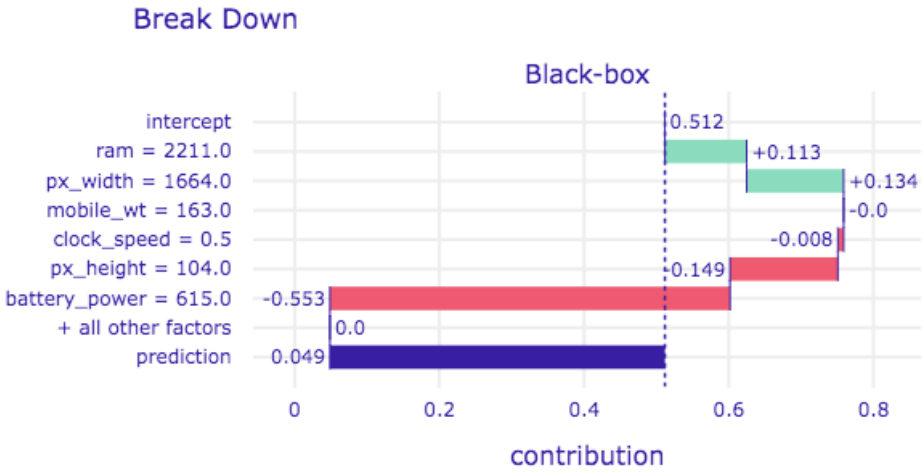


Figure 4.7. Break down profile of an observed negative sample.

Figure 4.7 shows a break down plot, that explains the level of involvement in each factor resulted in this prediction. Contrary to the intuition that a phone with large RAM capacities can almost always be considered expensive, to this sample a negative label was assigned. This is due to mainly its poor battery performance followed by the minuscule height aspect of the resolution (negative factors highlighted by red bars), which could not counter balance the high RAM and the resolution width (positive factors represented as green bars). The baseline of the prediction was 0.49, as expected from a balanced dataset.

Ceteris paribus

From the break down plot the observer can have an estimate of the critical factors that resulted in a certain prediction, thereby knowing the *targets* for slight adjustments that can lead to change in the model prediction. Ceteris paribus (CP) plots ascertain the degree of alteration to be made on a selected feature to influence the prediction for this instance.

CP profiles are designed to show partial model responses solely around a candidate point in the feature space, while assume that the rest is held constant. Therefore this concept can be viewed as the partial derivate of the predictor function.

Features of greater impact (deduced from the break down plot) are more exciting subjects of the sensitivity analysis as they present a larger gradient and amplitude on the plot, while a CP plot of an irrelevant will appear as a constant function around the baseline.

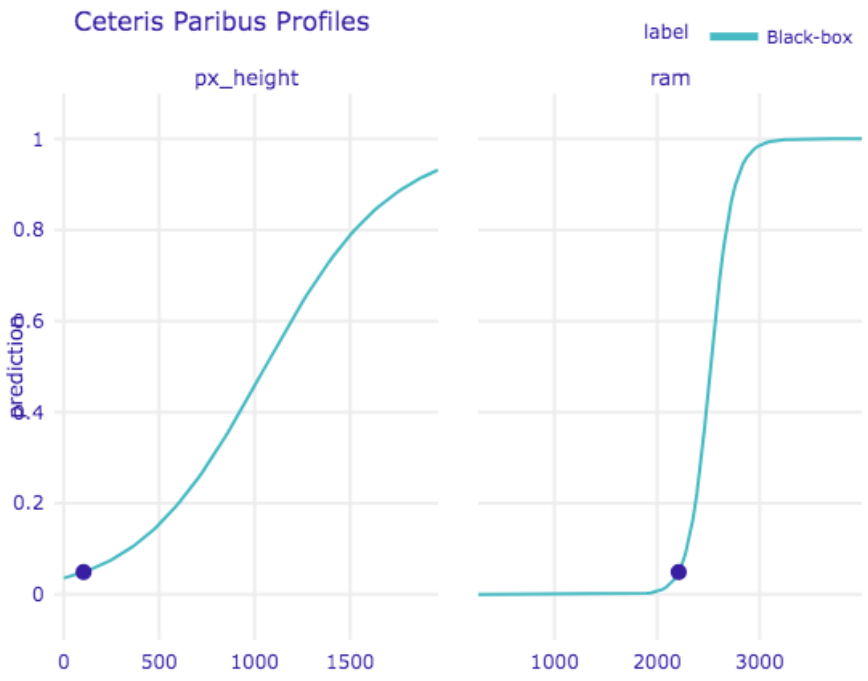


Figure 4.8. Ceteris paribus profile of an observed negative sample.

Figure 4.8 shows two CP profiles of the negative candidate point related its RAM and pixel height feature. The observed sample is displayed as a highlighted point, representing the concrete value that the selected prototypical sample takes on the showcased feature's domain.

RAM is clearly the more influential feature with greater gradient. The plot suggests that among the two properties of the phone, the manufacturer could boost the phone's RAM capacity slightly to be able to sell it at a higher price point. On the other hand the hight proportion of the resolution would require much larger improvements.

Another important aspect that can be revealed by analysing CP plots for all variables, is that these functions are all monotonically increasing, and they converge to 1 as the variable is close to its upper bound. This is conform with the hypothesis, suggesting that phones with better specification are more expensive, and also regardless of other poor qualities, a single extraordinary characteristic of a phone justifies its high price.

4.3.2 Model level explanation

Model evaluation (Section 3.3) is one of the most pristine approaches that measure various properties of an ML model, and also it quantises the quality of the fit. However other than general measurable qualities, it is hard to develop a one-fit-for-all solution with good performance.

Variable importance

Variable importance enumerates the most influential features of the models as an ordered list. It quantifies the extent of how reliant the model is on a specific variable to make an accurate prediction.

This AIX metric is a fundamental asset for **feature selection** during the model construction phase. That aligns with the assumption that the use of XAI methods in the V&V stage inevitably leads to better model construction (Section 4.1.3).

The variable importance can be extracted by exploiting specific elements of the structure of the model. (Such as hierarchy in decision trees, or coefficients in linear models.)

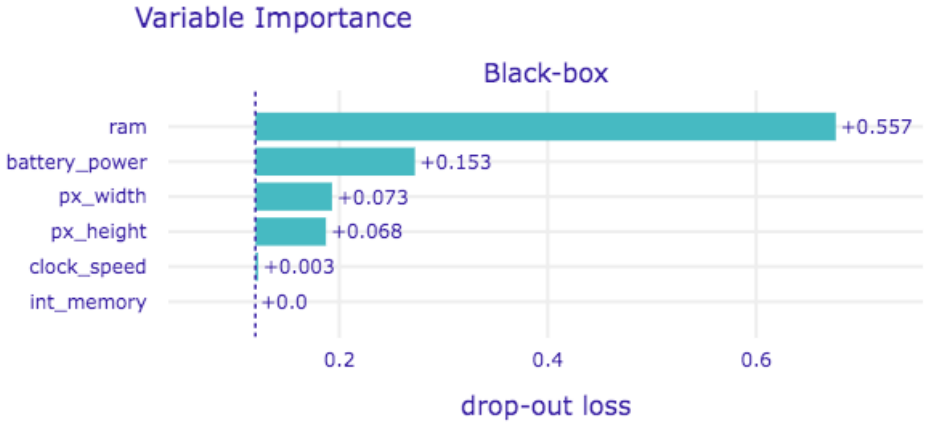


Figure 4.9. Variable importance plot shows the most influential features of the model.

Figure 4.9 depicts the important features of the model. This further supports the initial hypothesis suggested by all XAI methods so far, as RAM being the most impactful variable.

Partial dependence

Partial dependence (PD) profiles show the behaviour of the target variable as a function of a selected feature. PDs are formed by averaging Ceteris paribus profiles for all observations.

PD profiles can characterise classes based on the explanatory variable, since the members of a class show similar behaviour. They serve as a mean of **cross-cutting projection** of the observed class by symbolising a membership function to it. If this function takes a constant 1 value on a given domain, then any given sample with the attributed characteristics from that domain will be assigned to the class, described and explained by these aspects.

Furthermore, contiguous intervals of the feature domain represent distinct types **qualitative regions**, on which the prediction function is held constant. Sharp and short changes in the function's value between constant intervals (signalled by a steep gradient) suggest clear separation of qualitative states.

Let x be the explanatory, and y the target variable.
The PD profile is then $f(x) = y$.
Qualitative domains are defined by on $I \subseteq \mathbf{D}_x$ intervals, where $\forall i \in I \iff f(i) = c_1$.

However qualitative states might also be revealed in long alternating intervals, upon the inspection of the target variable as a function of other impactful variables restricted on the interval in question.

On $L \subseteq \mathbf{D}_x$ intervals where the target variable is not held constant, qualitative states can be defined on such $J \subset I$ intervals, observing other important variables ($v_i \in \mathbb{V}$) where the PD profile is shown as $g(\mathbb{V}, x) = y$, $\mathbf{D}_g = L \cap \mathbf{D}_{v_1} \cap \mathbf{D}_{v_2} \cap ...$, and $\forall j \in J \iff g(j) = c_2$ requirement is satisfied.

As a result, the versatility of PD profiles not only allows for analysing already existing qualitative regions, but also forming its boundaries.

PD profiles can also measure the quality of an extracted surrogate model in relation to the initial one. Since they can visualise the captured relationship between the target and an arbitrary variable, similarity among the profiles ensures the correctness of the surrogate model.

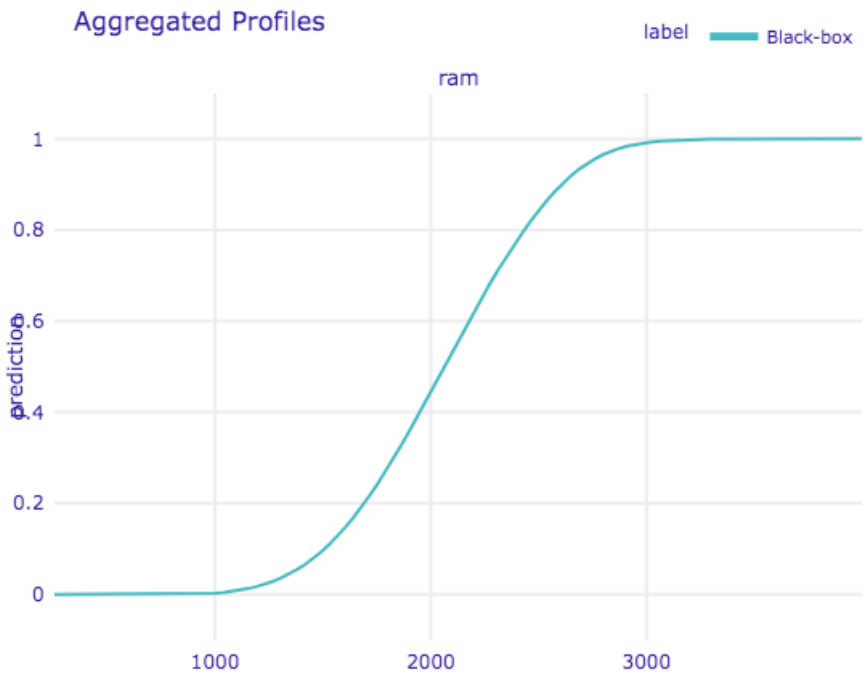


Figure 4.10. Variable importance plot shows the most influential features of the model.

Figure 4.10 shows a Partial dependence profile for the RAM feature as a function of the phone's price. As expected, due to its activation functions, the Neural Network has captured a smooth non-linear function between the target and the observed variable.
The plot supports the conclusions we draw from the previous featuring the BRCG algorithm. The PDP profile forms three distinct qualitative regions for the feature RAM: $<$ 1000 (`high`), 1000 $-$ 3000 (`medium`), $>$ 3000 (`low`) - as the constant, increasing, and constant parts of the function shows. The medium segment should be further analysed in the function of other variables as well. As BRCG suggested a good separation of `medium` is given as $m_1 \cup m_2$, when `pxHeight, pxWidth + RAM` form m_1, and `batteryPower, pxWidth + RAM` form m_1 subdomains. Similarly as the domains described in Equation 4.3.

The deceptive property of a PD profile is that correlated variables can have an additive effect on the prediction, which the PD profile does not take into account. The observer could not be sure whether the observed accounts solely for the certain positive prediction or it is an effect of correlated multiple variable's combinations. To resolve this issue, alternate PD versions were published (Local dependence plot and Accumulated local profile), which excludes the effect of correlated variables from the model.
Fortunately, in this case, all plots coincide.

Residual diagnostics

Residual diagnostic analysis highlights two related concepts of the model: *confidence in predictions* and the *quality of the fit*.

Since Neural Networks output probabilities for each class, instead of simply assigning labels to samples, these probabilities can be considered the model's confidence in a given prediction. *Residuals* quantify the deviation between the model's confidence and the actual predicted value of the label, (which can be 1 or 0 based on whether this particular label was assigned to the sample). As a result of each prediction, the lower the residual, the better the quality of the fit.

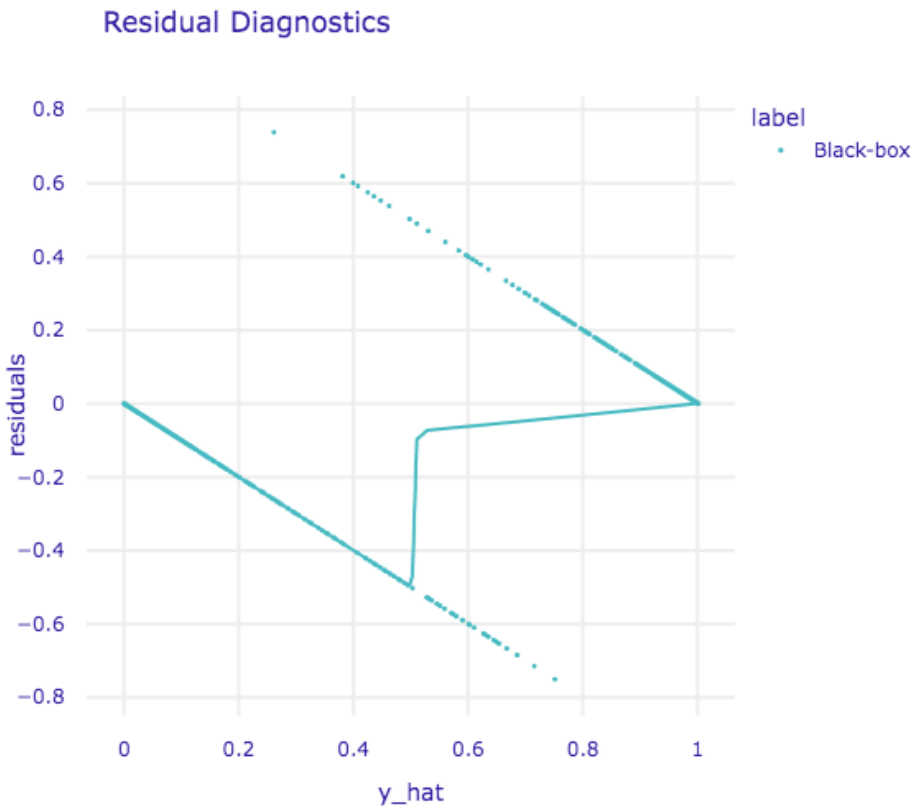


Figure 4.11. Residual diagnostic shows the goodness of the fit.

This implies that this tool is unsuitable for models that do not assign probabilities for each predicted class.

A single substantial deviation from zero does not reveal the general characteristics of the model, as it is presumably an outlier. Therefore the distribution of these residuals should be observed, where areas of high variance suggest a low-quality fit.

Figure 4.11 shows a residual diagnostic plot of phone price predictions.
The Y-axis displays the deviations from a perfect fit, while the domain of the predicted values of the dependent target variable appears on the X-axis. Values close to 0 represent phones predicted to have a low price with the highest confidence. As a result, the horizontal axis can be seen as the **continuous** domain of the target variable, and indeed the price of a phone is a continuous variable. This explains the low confidence of the model in the proximity of the boundaries of each class. Since phones in the medium segment share similarities with lower- and higher-end phones.
Another exciting aspect of the model is that it has a hard time classifying phones at very high prices. This may be due to the fact that a single exceptional quality of a product can justify its price. Meaning that phones with an extraordinary front camera and lacklustre other specifications are sold at a higher price point, however, these qualities vary heavily that the model cannot capture.
The model quality is almost perfect in the mid to high range.

A residual diagnostic analysis therefore can verify whether the qualitative discretisation appeals to the modelled system.

4.4 Explaining Neural Networks

Despite of the broad and increasing demand of deep learning, their deployment into safety-critical domains still faces thick barriers, due to their complex architecture. They are placed at the end of the interpretability spectrum among other ML models, as black-boxes whose decision processes are too complex to be explainable to users.

So far researches were struggling to explain complex NN, as only model-agnostic tools where available (ones presented in Section 4.3), with less success.

To illustrate this point, a NN classifier that aims to solve an NLP (natural language processing) problem will be showcased. I have trained a convolutional Neural Network (CNN) to classify the authenticity of COVID-19 related news tweets. Then the accurate (with 89% accuracy) decision-making process of the CNN is attempted to be explained with simply model-agnostic tools (SHAP library).

Figure 4.12 shows the not-so-convincing results of local analysing methods. A human observer can have an overall abstract view regarding features that can make news look fake (excessive capitalisation, special characters amidst the sentence, informal speech, use of a certain vocabulary) or genuine (mentioning geographical locations, or statistical data). However explanations heavily rely on the domain knowledge, imaginative and causality inferring capacities of the observer, as these explanations are not really telling. Furthermore there are instances which explanation does not make sense at all, hence the explainer highlighting numbers or punctuation as a decisive factor in the prediction.

Lastly the explanation of similarly performing Neural Networks varies tediously based on the model's architecture. For instance, the shifting window of a convolutional layer is able to capture the connection between differently sized ngrams (#n adjacent words), where as a fully connected linear layer that propagates information in both directions is able to understand the grammatical connections (relationships between words from the start and end of the sentence).

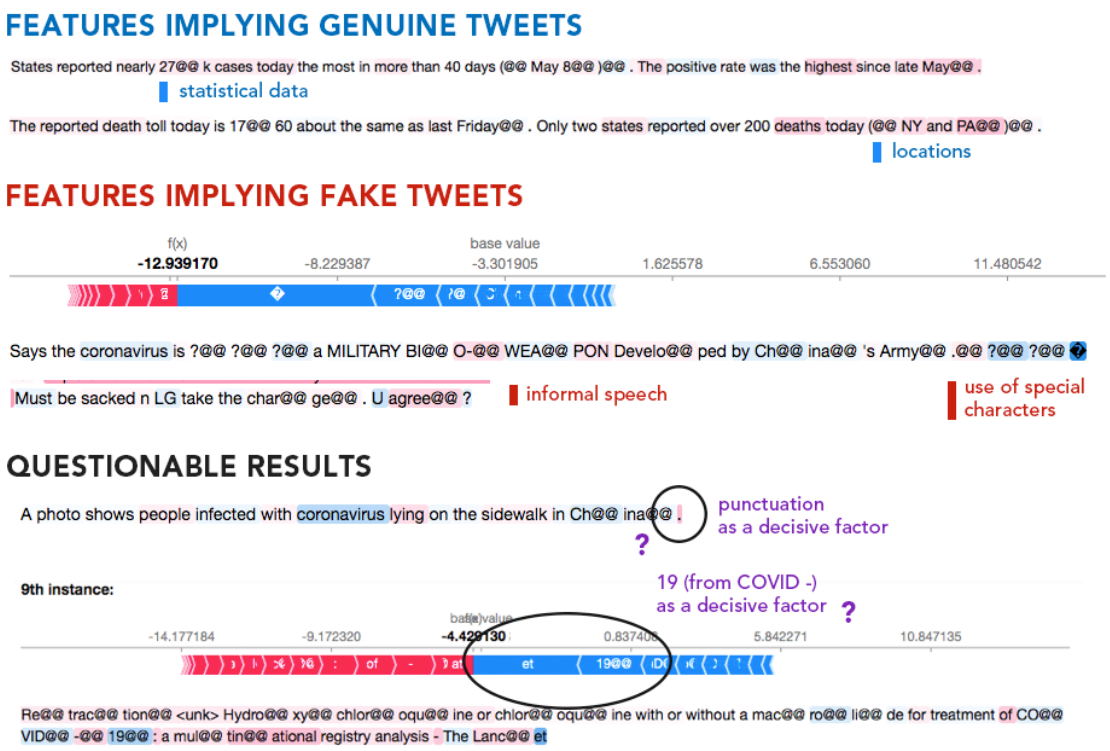


Figure 4.12. Break down for COVID-19 related tweets.

In this year (2021) researchers made a theoretical breakthrough in the field of explaining Neural Networks, by explaining their behaviour with logic formulas. Also additional ideas regarding intrinsic methods to explain NNs were published recently. These contributions will be reviewed in the following subsections.

4.4.1 Logic Explained Networks

Logic explained networks (LENs)[10] such Neural Networks that can both provide accurate predictions and explain them at the same time. They provide a high-level overview of the decision-making of the Neural Network as **first order logic** (FOL) formulas. LENs can be either on its own, as a directly explainable classifier, but it is also able to extract information from already constructed networks. It is suitable for implementing and explaining networks with most commonly applied activation functions: μ, ψ, $ReLu$ networks.

Functionality

The core notion of this explainer is very closely related to the BRCG algorithm. In order to provide explanations they both rely on human-understandable categories - *concepts* as their input learning criteria. These concepts (binary variables) will be the atomic elements of the resulting ruleset. The quality of the model heavily relies on the concept learning process, as both the degree of resolution and the sensible choice of landmarks separating the subdomains play an immense role in making accurate explanations.

Therefore the initial step of providing explanations are learning how to produce concepts from the input feature space in the following way:

$$g : X \to C$$

where $C = [0, 1]^k$ are concepts, and X is the input.

There are several different approaches to achieve this:

1. If $C = X$ the input is already provided in a form of concepts.
2. The IBM proposed way was to quantise the feature space to equally-sized subdomains. This is a practical but not ideal approach.
3. If the qualitative landmarks has been already extracted from the system, or they have been a part of the a priori knowledge initially, then the discretisation process it trivial.
4. An additional Neural Network can learn, how to map the features to the concepts due to its outstanding feature learning capacities. Thereby, the NN creates a division (**another tool for forming qualitative boundaries**) that is tailored to the model's aspects. However this approach would require formal verification, due to the unsupervised setting.

Eventually the explanations are generated by learning how to map the input concepts to the output concepts ($f : C \to Y$), where each output concept can be directly translated to a logic rule. While BRCG tries to bypass finding the optimal ruleset in an exponentially large search space, LENs solve this problem by feature learning and computationally heavy fitting. To generate the ruleset LENs leverage its individual *architecture* and *loss function* but also introduces the *parsimony criteria*. After fitting, the activation strengths of neurones can be extracted from the network. At the end a combination of active neurones in the output layer will symbolise a clause in the emerging ruleset.

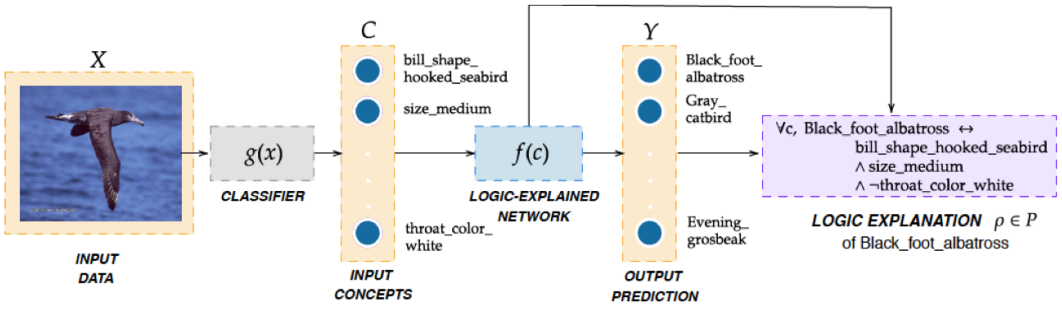


Figure 4.13. Architecture of a Logic Explained Network.

LENs provide explanations based on the following assumption (drawn from the field of cognitive psychology), that humans tend to have an implicit bias towards the simplest hypothesis when outlining different outcomes of a system. Therefore parsimony criteria bounds the complexity of the explanations - just as BRCG sets it with constraints. It is implemented by neglecting the output of some neurones with low importance, thereby simplifying the FOL explanation.

Concerns related to its applicability

The concept-base approach raises a number of concerns regarding the practical use of the Neural Networks.

Since NN are good feature learners, they are already used to learn very high dimensional bigdata datasets. When the size of their input dimension is multiplied as a result of discretisation, it will take a massive impact on both of their performance and the initially large complexity of their architecture.

A concept-based NN is unsuitable for additional XAI methods since they require explanatory variables as the model input. Therefore concepts appearing on model-agnostic plots bear zero explainability for a user.

Furthermore on several target domains of deep learning (NLP, image processing) defining such concepts that are interpretable is almost an infeasible task. (E.g., vast vocabulary of a language or pixels of an image.)

Therefore, as of now, it seems that LENs have more of a theoretical impact on the XAI field rather than a practical one.

Demonstration

The LEN package is currently - as of the writing of the thesis - is under development, therefore its functionalities are fairly limited. The developers have transferred the project under a new repository[7], with the ambitions to extend the framework's capabilities, but the current version is not functioning at all - as confirmed in an issue ticket. The original framework[8] supports only the directly interpretable model creation, without the ability to adjust the ruleset complexity.

The built LEN provided the following explanation for the problem - Section A.2.2 :

$$ram > 2146.5$$

which is an accurate (approximates the predictions with 90.8% accuracy) and simple overview, but does not contribute any additional knowledge to the so far gained hypothesis of the system.

4.4.2 Constrained Networks

Another collection of newly emerging ideas in verifying Neural Networks consists of intrinsic methods, that regulate the internal structure of the model during learning to enforce that the model's behaviour complies entirely with a set of criteria. These methods do not explain the operation of the model directly, however it gains insight for the developers *what it does not do*, thereby supporting the V&V analysis of the system.

Logic Explained Networks loosely fit into this category, as they even though incorporate custom components during the training process (a logic entropy loss function and entropy layer) their effect is not enforced to the prediction, due to model pruning[9]. This can be resolved if the explanation is used a classifier (applying the surrogate membership function). That applies for other surrogate methods as well - like BRCG.

[7] `https://github.com/pietrobarbiero/logic_explainer_networks/`

[8] `https://github.com/pietrobarbiero/pytorch_explain`

[9] neglecting irrelevant weights

A recent (2020) contribution[6] had surfaced the XAI community, when a group of researchers published a framework that can enforce analytic constraints in NNs in order to avoid the system violating the laws of physics. This enforcement can be achieved in the following two ways:

- Building an **Architecture Constrained Network**, which incorporates an additional *residual layer* for each constraint to ensure the prediction satisfies them.
- Constructing a **Loss Constrained Network** which penalises predictions that violate any constraints in the loss function.

The architecture constrained network can generalise the dataset just as much as an unconstrained network would do, as the implied constraints helped it to capture relationships truly existing in the modelled environment. The loss constrained network performed worse on the other hand, and also only provided soft constraints.

Eventually IBM also proposed a method to create directly explainable Neural Networks, as the TED algorithm. TED stands for **Teaching Explanations for Decisions**, which suggests that the explainer learns the (reference from a finite predefined set for an) explanation in addition to the input features, as the model input. Therefore it can not only predict but also provide explanation for the passed instance.

Chapter 5

Inside the core of a learning algorithm

In the previous chapter, a general overview has been given on primary motivations, principles of operation, and criteria set for XAI explainer algorithms. In order to gain further insights into this particular branch of science, this dissertation will take the reader through the process of the development of an explainable ML classifier.

The developed algorithm was highly motivated by BRCG explainer, in view of the very efficiently provided practical and accurate abstract representation of an arbitrary black-box model, due to its logical knowledge representation.
The BRCG is capable of precisely generalising the underlying domain only by means of simple rulesets, otherwise, its accuracy is inversely proportional to its complexity. Therefore it is regarded as highly competent AI explainer, but it is unideal for classification purposes. The current development would like to complement this aspect of the model. This is achieved by after casting the problem in a geometrical setting, an *iterative anytime algorithm* - with gradually increasing complexity - is sought to achieve data separation by efficient convex-hull approximation. This aspect of the process implies the name of the algorithm: **Nested cavities** (NC).

Firstly, an implementation-level analysis of the original algorithm is necessary as a foundation of NC. This observation inevitably leads the reader to study the field of mathematical optimisation, which allows the selection of the best fitting rulesets from a given search space. Eventually the developed and described algorithm will be evaluated to highlight both its superiority over the BRCG and its deficits.

In this form the algorithm was made for theoretical and educational purposes to showcase a detailed process of creating an explainable classifier, and to illustrate the brilliancy of the core geometric idea. However it also has great potential for further development and efficient operation.

5.1 Mathematical optimisation

Mathematical optimisation deals with the task of determining the best solutions to problems provided in a mathematical form, while while enforcing a set of constraints regarding the solution. The problem can be models of a physical phenomena, or complex operating systems, or even learning agents. For instance the problem formulation can serve as

a decision making asset for an intelligent agent to understand the main objective of its behaviour while restricting its operation.

This concept relies on sets of of *linear inequalities* that define a polyhedra, over which a linear function can be optimised. In case of *linear programming* (LP) only rational solutions of the inequalities are sought, then they can be always solved efficiently even for large-scale problems - given a feasible solution. Oftentimes it is desirable to find integer solutions to linear programs (*integer programming* - IP). Even though they are considered as NP-hard problems, there are practical approaches proven to be successful. [11]

There many solvers available to optimise integer or linear problem formulations. During the current work the CBC solver[1] will be used, implementing a version of a branch and bound algorithm. The developer's primary task is to conceptualise a mathematical model that best describes a given problem and provide a search space on which the problem is interpreted.

Integer programming

An integer problem is formulated as the following:

$$\text{objective function:} \quad min \ c^T x, \tag{5.1}$$

$$\text{constraints:} \quad A\underline{x} \leq b, \tag{5.2}$$

$$\text{bound constraints:} \quad l \leq x_i \leq u, \tag{5.3}$$

$$\underline{x} \in \mathbb{Z}^n \tag{5.4}$$

where c^T is the model of the problem - provided as input data, and $\underline{x}$ vector contains the *target variables* to be optimised. *Indicator variables* can be also introduced to the model as a subset of target vector to decide whether an action is taken. $b_i \in \underline{x}, b_i \in [0, 1]$.

Since to find the optimal solution of an integer program is NP-hard, usually a *linear programming relaxation* of the initial problem is constructed, which can solved efficiently in practice. The relaxation dismisses the integrity constraints from the problem formulation, thereby providing a tight approximation to the problem. Typically integer programming problems are solved using a linear-programming based *branch-and-bound* or *cutting plane* algorithm.

Branch and bound method

Firstly the LP relaxation is solved, which objective value is a lower bound for the IP, as if the imposed integrity criteria can only increase the value of the function.

The optimal solution for the IP is found by selecting a target variable whose value in the relaxation is fractional. On this variable the algorithm can branch by restricting the range of the variable based on the assumption that the optimal value of the variable in the IP will lie close to the current value. (E.g., $x_i = 6.7$ will be either be 6 or 7 in the optimal IP solution.) Therefore the problem is partitioned into two subproblems with $n - 1$ fractional variable to compute. This aspect of the algorithm is called **branching**. By generating such subproblems a search tree is created. The primary aim here is to traverse to tree to find the optimal solution, but also pruning the tree in the meantime to reduce computational time.

[1] https://github.com/coin-or/Cbc

Then the resulted two linear relaxations are solved recursively until every integrity criteria is satisfied. These solutions to IP - in the leaves of the tree - will serve as an upper-bound for the optimal solution. Therefore if in a given branch the solution of the LP relaxation is greater than the upper-bound the current branch can be disposed and there is no need to compute it further since there is already a better solution to the problem. This way the tree is pruned.

Increasing computational efficiency

Solvers in order to limit the search space even further thereby increasing performance introduce several mechanisms such as presolving, heuristics or cutting planes.

5.2 Algorithm development

5.2.1 Detailed implementation of BRCG

This section will provide further insights into the inner mechanisms of the BRCG algorithm as an extension to the description presented in Section 4.2.1. As the initial step, the algorithm binarises the input features to map them into corresponding concepts. As the algorithm the outputted concepts from them, they will form the emerging ruleset. More precisely, if a DNF and binarised feature space is given, a clause corresponds to a conjunction of features and a sample satisfies a clause if it has all features contained in the clause. []

Therefore the search space of the problem consists of all the quantised features that can appear in the emerging ruleset. This search space is transformed into a matrix form to serve as the model of the problem and to present it in a suitable form for the IP solver.

The aim is to select a collection of features that best describes the underlying problem (according to the features values). A candidate feature is picked into the ruleset, whether it was satisfied more frequently in positive samples than in negatives. This frequency of containment can be characterised in the loss function.

The core notion behind the explainer is to conceptualise a complex system in form of simple, interpretable rules. Therefore, a restricted search space is derived from the original one, consisting only one-length (and empty) conjunctions. On this restricted space a Master Integer Program (MIP) is formulated. The problem can be solved efficiently, since the size of the input space is equal to the dimensions of the quantised dataset.

Eventually further features are selected into the simplified ruleset, that has a positive contribution to the objective function. This expansion of the search space is carried out by a greedy Beam Search algorithm.

In terms of the NC algorithm, only the MIP problem is relevant.

Preprocessing

Firstly the values of the continuous features are discretised in the following way, similar to One Hot Encoding methods.

$$F \to \{F \oplus l_i \mid l_i \in L, L \subset \mathbb{D}_F\} \tag{5.5}$$

Each column representing a feature is substituted by the given $|L|$ number of columns, where L is the set of landmarks which is a subset of the feature's domain, and $\oplus$ is an appropriate operator ($\leq, >$ for ordinal, $=, \neq$ for categorical variables). A value of the resulting features are binary variables $\underline{f} = \{F \oplus l_i\} \in \{0,1\}^k$ whether the sample satisfies the expression.

The default Feature Binarizer component can be used for the extraction of landmarks if no other method is available. In case of continuous variables it will divide its domain into n ranges with equal length, that can be set by a parameter. As for categorical values, since there are always finite number of them, and usually have a low cardinality, here the division is represented by the elements of the set.

These rulesets are then generated in the following way. All $\underline{f}$ expression is collected into $\mathbf{C}$. Each $c \in \mathbf{C}$ candidate term can be selected to be a literal in a decision rule.

Input matrices

As shown in Equation 5.2, IPs are defined as linear inequalities. Therefore subsequently the model of the problem formulation should be given in a matrix form.

Let *feature matrix* $I \subseteq \mathbb{R}_{fxl}$ where f is the number of features, l is the number of the samples which will form the search space for the integer programming task. In this matrix each row represents a possible clause as conjunctions of the features, and their values signal whether a feature is selected into the clause. $\underline{i} \in I, \underline{i} \in \{0,1\}^k$.
If we would use a global solver $l = 2^f$ since all possible value should be examined.

Let the *conjunction matrix* be $A \subseteq \mathbb{R}_{dxl}$ where d is the number of samples in the dataset. In this matrix each row represents a data point, and their each of their feature value signals wether the feature is satisfied.

Master Integer Program

Since BRCG intents to explain a supervised binary classification, the dataset can be divided into positive and negative samples $N = P \cup Z$. The objective function is constructed in a way to minimise the occurrence of misclassified samples. If a sample was classified incorrectly, the function summarises the number of clauses that should have been selected into or removed from the ruleset to classify it correctly, thereby it it is able to identify the false negatives and the false positives.

K denotes any combination of clauses as conjunctions that occurs in the dataset. $K_i \subseteq K$ are conjunctions that are satisfied by a $i \in N$ sample.

Indicator variables

$\xi_i : i \in P$: denote the positive samples classified incorrectly

$\omega_k : k \in K$ denote whether clause k is used in the rule set

Objective function

$$min \sum_{i \in P} \xi_i + \sum_{i \in Z} \sum_{k \in K_i} \omega_k \tag{5.6}$$

Where the first sum identifies the false negatives, whereas the second one can identify the false positive. The fact that the IP solution is formulated with strong connection of these confusion values, provides the great opportunity for the improvement for this algorithm.

Constraints

1. $\xi_i + \sum_{k \in K_i} \omega_k >= 1,\ \xi_i >= 0,\ i \in P$
2. $\sum_{k \in K} c_k * \omega_k < C$

The first constraint helps to identify false negatives, because if it is a negative, it was not selected into the ruleset as a clause, therefore the second sum is 0.

The second constraint bounds the complexity of the ruleset. It is used since the explainer learns to fit the training set as much as it can, and no mechanism is here implemented that would guide the generalisation process and avoid overfitting. Each selected clause in each conjunction is penalised by 1.

5.2.2 Nested cavities

The following idea originates from the "Parallel Coordinates: Visualization, Exploration and Classification of High-Dimensional Data" publication by Alfred Inselberg.[19]

Nested cavities classifier relies on *convex-hull approximation* method, that is able the **wrap** data points in an N-dimensional space based on a certain set criteria. The primary aim of the wrapper is to efficiently separate positive (P) and negative (Z) samples, given a binary classification problem. $S = P \cup Z$

It separates them in an iterative way, that in each iteration more complex cutting surfaces are used for separation, resulting in an *anytime algorithm*. An anytime algorithm, as its name suggests, can be stopped at any time period and provide a result to a given task, which quality depends on the computation time that it was able to perform. During the separation process the amount of observed samples are decreasing in each iteration, due to the wrapper's characteristics - otherwise the algorithm would stop as no further separation can be made. Due to this fact, the complexity of the cutting surface is inversely proportional with the input space. This allows for a very precise but also efficient classification process, as algorithms with large complexity and computation time are able to process small set of samples in a practical manner. Whereas a classifier with low complexity is considered good enough to separate big amounts of data - to draw trivial conclusions.

The wrapper alternates between identifying positive and negative samples with a margin of error regarding the currently applied complexity setting.

- In first part of the iteration it aims to enclose positive samples, but generally it is not feasible, therefore a small subset of negative samples (false positives) are separated as well. $P \cup S - P \subset S_1$.
- Then the resulting subset from the previous one is processed (S_1). Here the same process applies but now targeting negative samples and some positive ones as a margin of error (false negatives). $(P \cap S_1) \cup ((S - P) \cap S_1) \subset S_2$.

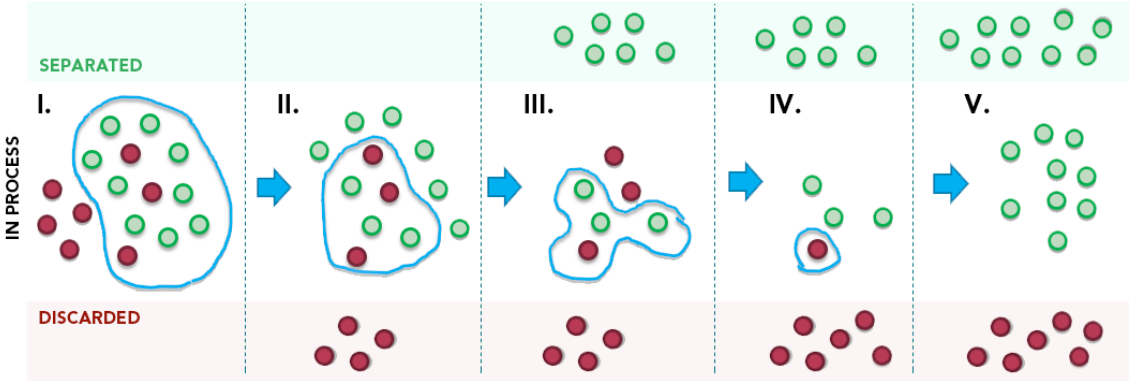


Figure 5.1. Separation process according to the Nested cavities algorithm.

- Samples identified by the second part of the iteration (S_2) are the subjects of the subsequent iteration. Samples that have not been selected into that set ($S_1 - S_2$) are classified as positive, whereas samples not wrapped in the first part of the iteration are discarded ($S - S_1$)
- The iterative process continues in this alternating manner until one of the terminating criteria has not been reached.

Figure 5.1 illustrates the separation process.

Notice that wrapper is heavily reliant identify false negatives and false positives precisely, to make sure positive samples are not discarded ($Z \subset S - S_1$) or negative samples are not selected in, simply to ensure the correctness of the algorithm. This is feasible however since the cutting surface can be permissive rather than being strict.

A terminating criterion can be one of the following:

- A pure separation has been achieved between the two classes.
- $S_j \cap Z \neq 0$, but any further iteration cannot grant a better separation between the two classes, as $S_j = S_{j+2}$.
- The complexity of the ruleset has reached a user set constraint to avoid overfitting.

Assuming that containment to any wrapped subspace S_w provided by the wrapper can be defined by $\varphi(S_w)$ decision rule, that the resulting ruleset can be approximated by

$$\bigcup_{0 \leq i \leq n} \varphi(S_{2i+1}) - \varphi(S_{2i+2}) \tag{5.7}$$

over n iteration.

It is important to make one final remark of the algorithm's interpretability, which is superior than an average rule learner's. The resulting ruleset is provided as a conjunction of the **included** ($\varphi(S_{i+1})$) and **excluded** ($\varphi(S_{i+2})$) intervals, which grants insights for the developer regarding the distribution of the classes as well as the cohesive factors of those regions.

5.2.3 Developing a classifier

Realising that the way how MIP formulation in the BRCG algorithm (Equation 5.6) was constructed, allows the identification of the false positives and false negatives instances.

Therefore the respective terms in the objective function can be weighted to control the ratio of the targeted confusion metrics. Resulting the following objective function:

$$min \ \ \delta_{FN} \sum_{i \in P} \xi_i + \delta_{FP} \sum_{i \in Z} \sum_{k \in K_i} \omega_k \tag{5.8}$$

where δ_{FN} penalises the occurrence of false negatives in the retrieved ruleset, whereas δ_{FP} contributes to the loss if a false positive appears there.

That allows for the implementation of the Nested Cavities in this environment, utilising a slightly modified version of the BRCG solver as the convex-hull approximator (Section A.3.2). Therefore the resulted ruleset after each part of the iteration is defining an wrapped subspace associated with $\varphi(S_j)$. It is supported by the fact that if $\delta_{FN} >> delta_{FP}$, then a homogeny group of negative samples can be separated and excluded from the collection of observed samples, therefore the above defined S_i sets can be constructed.

The algorithm also consists of a loop of iterations, which conform with the above mentioned process. Each iteration has two consecutive repetitive phases. After each iteration the complexity parameter are set to retrieve results of gradually increasing complexity, while the search space shrinks, because a lot of true positive and true negative values are separated already.

In each iteration the MIP (Equation 5.8) are solved globally, over the entire input space. The matrices, as the inputs of the solver, are constructed so to contain all possible combination of features as conjunctions, rather the only the ones occurring in the input dataset. This can cause performance issues, since the size of the matrices are exponentially large in terms of the dimension of the input. Solving the MIP over a large set should not cause performance issues as the algorithm is designed to compute rough and simple cuts in early stages. However as of now the algorithm cannot avoid listing all the possible combinations of the features, since they are inputted to the solver at once, which can have massive storage demands. Constraining the initial search space is definitely a subject of future research.

During the phases of the iteration AIX algorithm parameterised in an alternating fashion, in each even phase all the positives and some false positives are enclosed, whereas in each odd phase all negatives and some false negatives are retrieved. In the next iteration the separated samples are examined.

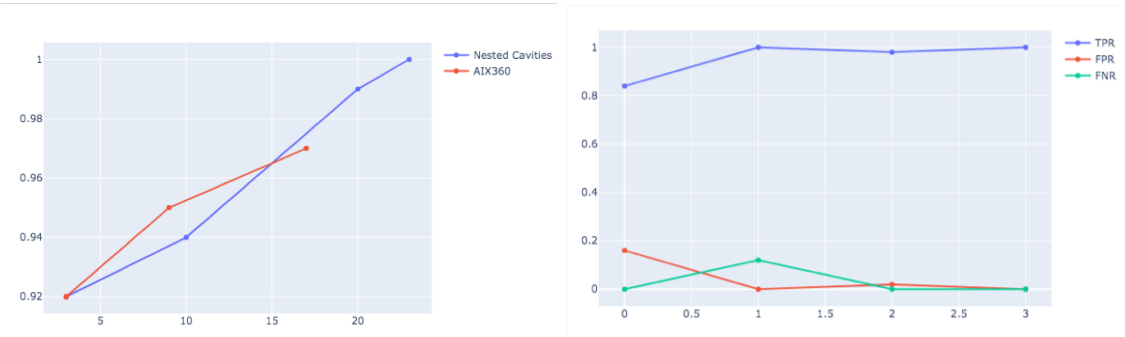
The retrieved ruleset is conform with the one described in Equation 5.7. Here each subset associated with the first and the second halves of the iteration can be combined with a logical XOR connection, due to the fact that, if A and B are disjunct $\implies A \cup B \iff A \oplus B$.

5.2.4 Evaluation

To demonstrate the use of the algorithm the Iris dataset [2] was used. Due to the global solving approach, the algorithm cannot manage high dimensional datasets yet. Therefore, the problem has been narrowed down to a binary classification - as it initially contained three types of target labels. Also the Feature Binarizer preprocessor was set to divide the

[2] `https://archive.ics.uci.edu/ml/datasets/iris`

feature domains to only 3 distinct parts in order to keep the dimensions narrow, and the global solver manageable.



(a) Accuracy - complexity plot for the models.

(b) Confusion metrics of NC.

Figure 5.2. Comparing the performance of the NC and the BRCG.

Figure 5.2b illustrates the inner mechanics of the algorithm as the confusion metrics alternate. It is also clearly visible that after the third phase (the first part of the second iteration) all samples in each class are separated, thus the true positive rate of the model converges to 100% meantime the true negative rate declines to 0.

Figure 5.2a plots the accuracy of each competing algorithm (Nested cavities and BRCG) in the function of their complexity. Meanwhile the AIX algorithm provides better performance with simpler rules, it struggles give more accurately performing model even with almost unlimited complexity. (It cannot provide a better one than it did with $C = 17$). However the algorithm proposed in this dissertation was able to find the perfect solution (global optimum).

Chapter 6

Identifying operational regimes

The primary subject of this thesis is to showcase how modern AI explainers can play an essential role in both the extraction and the validation of qualitative models related to cyber-physical systems. In previous chapters, foundational topics were introduced and covered in detail, revealing insightful relationships with the target domain. The reader was able to follow through numerous illustrative demonstrations to acquire experience regarding the capabilities of state-of-the-art XAI solutions. It is high time to place the gathered knowledge in a complex real-life application through the conducted research.

Therefore, this chapter focuses on interpretable and explainable discrete qualitative modelling of observations. The core abstraction problem in qualitative model creation is the clustering of continuous observed data belonging to the same operational domain exposing a similar behaviour into a single qualitative state. The structure of qualitative models well-reflect the dynamic architecture of the target system. Thus, they are natural choices to build a digital twin serving as the core for supervisory control of complex cyber-physical systems.

The quality of clustering crucially influences the faithfulness of the qualitative model and consequently the efficiency of the control. Moreover, proper detection and management of outliers is a primary objective in critical applications.

The current research proposes a technique that combines dimensionality reduction and clustering methods that accurately separate the operational domains while also recognising outliers using well-fitting cluster borders. This solution also features various interpretability methods that can showcase the cohesive factors amongst the operating regions and guide the understanding of the functionality as well.

6.1 Outline of the research work

6.1.1 Overview

Implementations use a wide variety of sensors both for the cyber and the physical parts of the CPS to assure proper observability of the control and controlled systems, thus resulting in a many-dimensional stream of big data. The extraction of a manageable and interpretable qualitative model and supervisory control demands selecting a limited subset of core variables, ascertained by careful feature selection. Ensuring the quality of the extraction begins here since incautious feature selection can induce a set of correlated variables as redundancy and noise to the engineering model. As a consequence, it would also

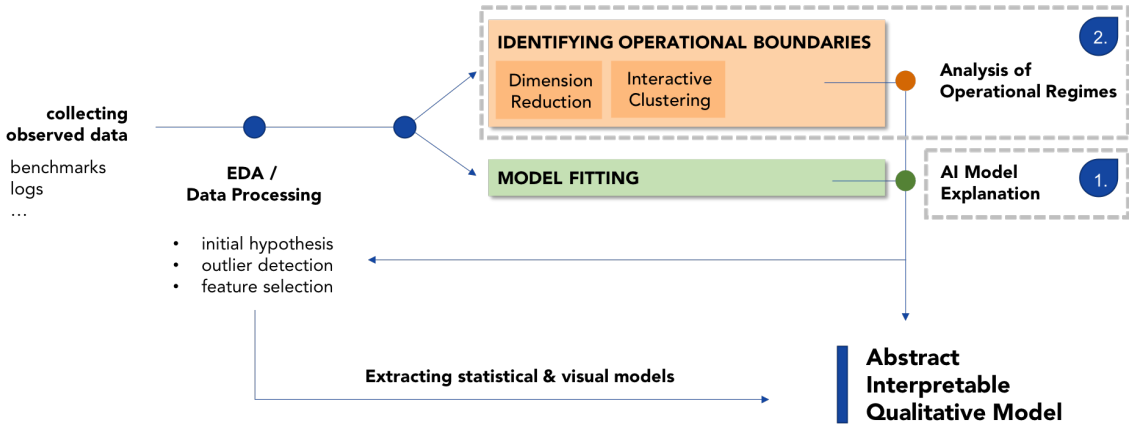


Figure 6.1. Iterative qualitative model building workflow.

go against the fundamental principles of a qualitative model, generalising the properties of complex physical processes and then inferring from minimal information. However, in addition to these concepts, various preliminary data processing tasks establish the deeper analysis of the system, they are presented below as part of the entire process.

The proposed approach for the work in the framework of the research adopts the following iterative model building process as depicted in Figure 6.1.

The concept of empirical system identification allows creating compound scalable and hybrid models that provide a guaranteed level of service under the typically fluctuating workload. As its initial step, the observed data is *collected* from different sources and *transformed* into a format that is suitable for further visual and statistical analysis. The subsequent *EDA stage* is an essential part of every data analysis workflow. *Data cleansing* as an initial phase of the process consolidates the dataset by handling missing or undoubtedly faulty instances. In order to construct complex models revealing deep inherent relationships between events or system components, an initial hypothesis should be established as a preliminary step. It consists of statistical and visual analysis methods that serve as a means of a high-level overview of the modelled system by highlighting its focal points that are subjects of subsequent analysis. Each task in the data processing phase have been assisted by state-of-the-art data analytics libraries as Python frameworks, just like: Pandas, Numpy, Plotly, etc.

Outlier detection identifies anomalies from which unexpected but crucial events can be deduced that require careful consideration and handling and also facilitate data cleansing. Thorough *feature selection* only a minimal relevant subset of information will be available for the data scientist. In this workflow, both parts of the initial hypothesis were refined based on obtaining additional domain knowledge in the V&V phase. That allows for a semiautomated model building scenario, which eventually transforms the EDA process into a confirmatory data analysis that validates the correctness of the hypothesis.

The dynamic architecture of the continuous CPS is decomposed into several qualitative model fragments to separate individual qualitative states, describing similar behaviour. The recognition of these domain boundaries is supported with the following tools.

The first step requires the transformation of the entire dataset to a compact and understandable 2D visual representation (Fig. 6.2b). It exposes the potential operation domains, which are impossible to reveal in the original view (Fig. 6.2c). This technique allows the domain expert to get an intuition for the distribution of the data while retaining as much structure from the initial high dimensional space as possible. Statistical and

inference methods audit the correspondence of the indicated candidate clustering and the distinct operational domains of the system. Statistics evaluate the quality of separation and any noise introduced, forming a solid foundation for feature engineering.

Eventually, reasoning explores the cohesive factors amongst each qualitative operating state. It provides insight into their inner functionality. If complemented with variable importance metrics, it highlights the contribution of different technical factors in the individual domains.

Embedded AI subcomponents extend the hybrid model, as they are utilised to solve immensely complex tasks that regular modelling tools cannot conceptualise (i.e. image processing). They are assisted with AI Explainers validate their operation and justify their use.

6.1.2 Pilot example

The pilot example originates in an experiment evaluating remote web-service by measuring the latencies in different phases in a client-server task invocation (communication: Response Time [RT], data processing: Request Processing Time [RPT], total: Round Trip Time [RTT]). Experiments were concluded homogeneously across several spatial and temporal locations. The timestamped [*start.time*] logs capture geographic properties, such as *location*, *country*, time zones [*time*]; client attributes: [*client.type*], IP address (*IP*), data centre [*DC*]. The dependence of the total service time (RTT) on individual factors outlines the central issue of remote data-processing involved in the given typical CPS application.

(a) Class distribution of the initial data

(b) Clustering (10 clusters)

(c) Back-annotation of the clusters to the data

Figure 6.2. Clustering process

An initial partitioning by visual exploratory data analysis identified cases (*normal, border class, faulty*) (Fig. 6.2a) by the qualitative discretisation of the RT values (*low, medium, high*); and the next step is the estimation of dependencies on input factors by partitioning the clusters.

6.2 Identifying the operational boundaries

6.2.1 Dimension reduction

Dimension reduction (DR) projects a set of observations into a low-dimensional space while preserving most of the knowledge on the input features, like the cluster structure in the high-dimensional space. It deals with the spatial disposition of the samples, i.e. whether other points of data should be close or far away from the selected one (local & global structure) - as illustrated by Figure 6.3. [2]

Figure 6.3. Dimension reduction methods retain information regarding spatial properties in the lower-dimensional space.

The process generally happens either by omitting features that have a low variance across the whole distribution or deriving a new feature from several initial ones.

Traditional DR methods, like PCA, account for only the linear separation of the data, which is an excellent first approach for rough modelling but inadequate for generating complex models. Advanced techniques (like t-SNE, UMap) tend only to affirm the primary model's local or global structure. A new algorithm, PaCMAP [28] combining mid-near pairs of observations and dynamic graph optimisation, performed well in comparision to other DR tools on both local and global structure (cf. Fig. 6.2b).

Modelling physical objects raises additional requirements of representational invariance for modelling physical systems: (i) results must be independent of the unit of measurement and (ii) substituting a set of features with another one having the same information content (like transformed metrics) should not affect the results.

Since none of the algorithms mentioned above fulfils these in their basic form, we propose additional solutions. As only the relative magnitude of a particular feature is essential in qualitative modelling, feature scaling helps: e.g., domains of each variable should be normalised according to the mean of their distribution. The second problem can be solved by careful feature selection, discussed in a later chapter.

Another known insufficiency of DR methods is their inability to deal with categorical data. It is suggested to resolve this issue by 1-of-N coding similarly as machine learning algorithms do it.

6.2.2 Interactive clustering

Any commonly used algorithm can cluster the transformed data. In this research, DBScan [13] (a fast, density-based clustering) was used, as it performs the best with PaCMAP's output. A further significant advantage of DBScan is the recognition of sample points that lie alone in low-density regions - resulting in labelling them as outliers while also being parametrised practically. In operation-critical systems, the noise only after the second layer of manual examination or with the support of additional outlier detection methods can be disregarded.

Figure 6.2b shows the outcome of the clustering process, while Figure 6.2c shows how the clustering process refined the initial qualitative boundaries of the target variable (RT). According to the newly emerged hypothesis, some of the `border class` samples (instances of the red cluster) should be considered as normal ones, whereas a few `faulty` samples (instances of the green cluster) should have been in the `border class` set. This hypothesis is based on the assumption, that the majority of the samples was correctly classified initially. As a consequence, one could say that the RT's threshold, below which samples

are classified as normal higher, than the initial, and the other threshold above which faulty samples are found should be also higher. The yellow cluster has been divided evenly between the normal and the border class set, implying there is no clear separating factor between the two classes.

The derived clusters will be regarded as individual operating region of the given system, and they will be the subject of further analysis.

6.3 Model analysis, validation, and verification

6.3.1 Statistical evaluation and feature selection

High dimensionality generally introduces redundant or highly correlated variables to the model, which will impair the process of extracting qualitative models and could lead to unwanted behaviour that should be avoided in critical systems at all costs. The quality of the clustering outcome heavily depends on selecting relevant descriptor variables to avoid the underfitting of the model or redundancy-induced surplus complexity. In this scenario, interactive partitioning-based clustering is preferred since the usual/automatic methods are less helpful. Interactive clustering tailors the process to specific application domains validated by several evaluation measures. [4] In a purely unsupervised learning case, correlation metrics (e.g., phi(K) [3] - which can process categorical features as well) can be used to select the relevant features.

A target variable can be constructed for the given problem using prior knowledge about the system extracted from the engineering model. The proposed iterative feature selection process is similar to a concept called Minimum Redundancy Maximum Relevance - mRMR [29]. This idea seeks to identify a small subset of features that collectively have the maximum possible predictive power while minimising the size of the collection, omitting the redundant variables. The field of xAI offers a Feature Importance that showcases the contribution of each individual feature to the target metrics. Therefore an ordered list of variables ranked by their respective importance guides the selection of the most relevant features. Fundamental means of cluster evaluation, such as noise level, the ratio of homogeneity, and completeness, support the V&V of clusters.

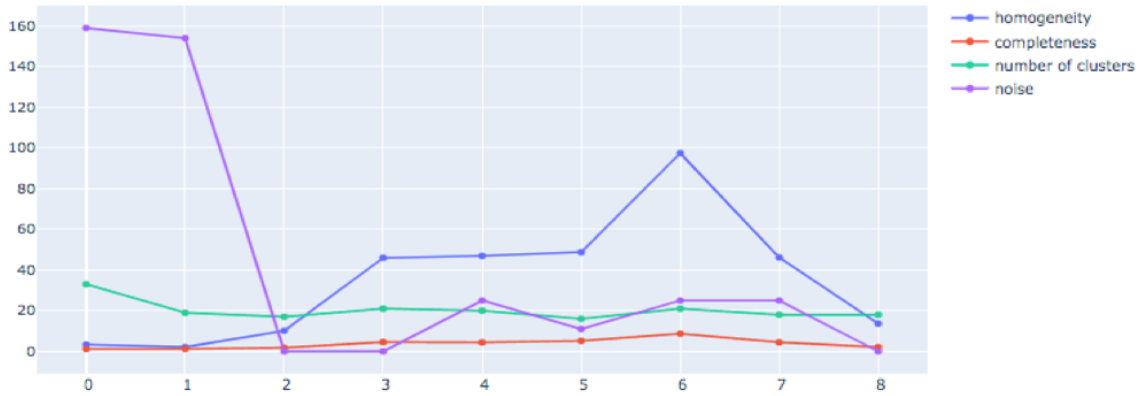
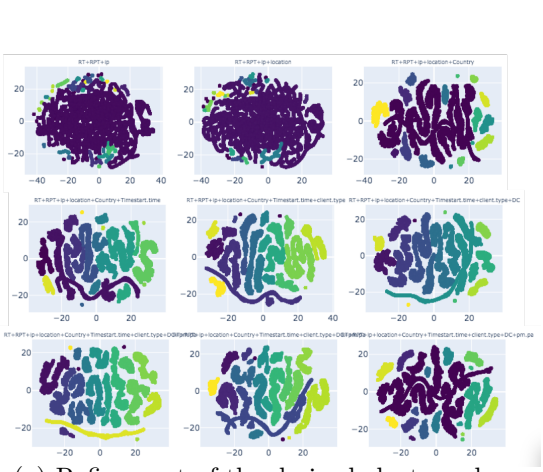


Figure 6.4. Noise level, homogeneity, completeness of the derived clusters along the feature selection process.

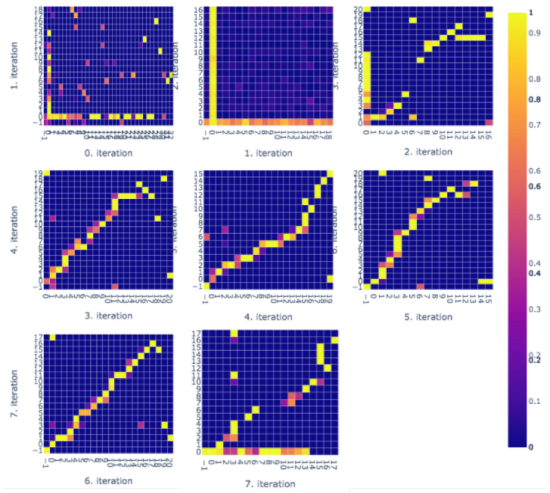
Figure 6.4 illustrates how these properties of the derived clusters changes overtime, as the feature selection refines. Some degree of redundancy is introduced to the model if the

noise level starts to rise. In this case the feature selection should stop, as the given feature set already covers relevant dimensions of the model.

Firstly the input attributes are sorted based on relevance metrics, such as the Feature Importance value delivered by an xAI algorithm. Then, a feature is selected with high relevance in each iteration, starting from the most significant feature. But it is discarded in case of high correlation with other variables that have been already picked as a member of the set. The selection must stop when the noise level rises, indicating the irrelevance of the additional variables regarding the initial problem.



(a) Refinement of the derived clusters along the feature selection process.



(b) How the containment of samples change along the feature selection process.

Figure 6.5. Newly emerging clusters as the feature selection process becomes more refined.

Figure 6.5 illustrates the distinct set of imposed features to the cluster separation. During this process features of the pilot example were sorted based on their importance to the model and then iteratively selected into the set of currently observed features. When there is not enough information to infer structure of the model (iteration 1, 2, 3 with less than 6 selected features), clusters cannot be formed, only noise appears on the model. Figure 6.5b shows on the other hand how members of each cluster are reassigned to the newly emerged clusters. If the reassignment is not homogeneous, meaning that samples originating from the same clusters will end up in different ones, than the currently introduced feature still bears additional information to the model as it forms the separation of the operational domains.

Homogeneous reassignment of the samples can be seen as a result of the 5. iteration (4. picture in Figure 6.5b, 5. in Figure 6.5a). At this point the selection of further features should have stopped - in accordance with what the statistical evaluation of the clusters suggested Figure 6.4.

In the last iteration the newly introduced feature added a lot of noise to the system, which worsened the quality of the separation.

6.3.2 Analysing qualitative regions

Operational domain identification aims to identify homogeneous domains composed of data points of similar system behaviour. The clustering process already determined these domains phenomenologically; however, discovering their boundaries and interpreting their inner functionality is still a primary intention.

This section proposes two distinct methods to encapsulate each operating region's cohesive factors and realise the system boundaries. Both previously review techniques capitalise on tools from the realm of XAI.

- The first one creates to each cluster an approximate **symbolic membership function**. Here BRCG can be directly applied, as it specialises in constructing a directly interpretable model based on data, either the initial dataset or ML-generated model of it.
- The second one uses DALEX to estimate **explanatory metrics and visualisations** highlighting the main characteristics of the domain after fitting an arbitrary ML model to it.

However, it is essential to realize that these tools are not just playing a role in the validation phase. Instead, they can be incorporated into every aspect of the Exploratory data analysis process, just as this dissertation showcases.

Two particular clusters will be the primary subject of the analysis in the pilot example. Let **Cluster A** be the well-separated set of data at the top on Fig. 6.2b marked with dark blue. The significant curve-shaped green-colored cluster at the bottom of the figure is referred to as cluster **Cluster B**.

6.3.3 Membership function of the operation domain via BRCG

Having no prior knowledge of the characteristics of **Cluster A & B**, these regions are analysed with the BRCG.

Ruleset A	Ruleset B	
RT > 2598.20 RPT > 828.00 ip = 208.87.25.162 region 1	RTT > 1933.00 RPT > 828.00 client = Microsoft region 1	RTT > 1933.00 RPT > 656.00 ip = 209.188.85.60 start.time <= 1359777 region 3
RT > 2598.20 RPT > 719.00 start.time <= 1359769 location = Durham region 2	RTT > 1933.00 RPT > 781.00 ip = 64.20.37.202 start.time > 1359769 region 2	ip = 67.227.216.114 region 4

Figure 6.6. Result of the rule generation for the respective clusters

Results (Fig. 6.6) indicate that samples with only high latency values are present in the observed domains. These samples originate from specific locations and timeframes. From the rulesets, it is deductible that both clusters represent substandard data - due to the RT value being high - indicating the presence of classes: *border class*, and *faulty*. Cluster A shows higher RT base values than B, however, overlaps may be possible, therefore no

further distinction could be made. Additional details (like spatial and temporal properties) will be the subject of root cause analysis.

The membership function approximation approach may provide accurate results, but it only returns a single solution without transparency of the process. For instance, it hides the reason for inclusion or omission of the individual factors into the rules. Moreover, a moderate change in the weights in the cost function may lead to drastically different rule sets.

6.3.4 Operating domain characterisation via DALEX

Operating domain characterisation aims at the well-interpretable exposition of the cohesion and separation factors and the boundaries of a cluster. It relies on the concept of **prototype generation**, where a single point representing the characteristics of all points in the entire cluster, if it is sufficiently homogeneous (similarly to an equivalence class leader in automata theory). The ProtoDash [17] implementation suffers from performance issues upon processing a high-dimensional large-scale dataset. However, its core idea can be adapted by using the central element in a density-based cluster as its representative prototype. Note that clusters mapped to qualitative states have to be homogeneous by their definition.

Using a prototype reduces the characterisation of a cluster to the local analysis of a single data instance. modelStudio [7] provides local explainability techniques supporting:

- **Extraction of relevant features**: Break Down / Shapley values. These methods determine the features' contribution to the symbolic membership function, using its partial derivative (the sensitivity) to the individual input features. Variables of a marginal influence on the model are subject to the following sensitivity analysis.
- **Sensitivity analysis**: The already thoroughly reviewed Ceteris paribus profiles support the identification of the cluster cohesion variables and their respective domains.

The above-mentioned analytical steps on Clusters A and B, result in a better insight to their composition.

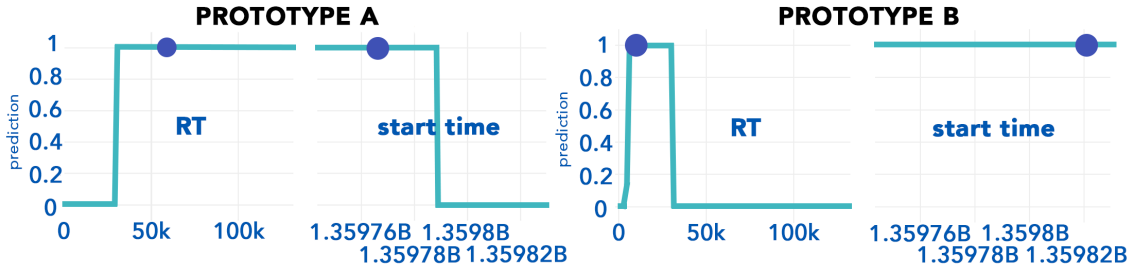


Figure 6.7. CP profiles representing Cluster A, B
RT's background signals initial class separation (Fig. 6.2a)

The highlighted points on the CP profiles are the prototypes of the particular clusters (Fig. 6.7). Both *RT* and *start.time* contribute to the symbolic membership function in Cluster A, which is sensitive only to the higher values of *RT*'s and the low values of *start.time*. Cluster B is constrained to a specific subset of *RT*, to which the feature demonstrates high sensitivity while being insensitive to the *start.time*.

Comparing the identified domains of sensitivity with the qualitative regions, Cluster A contains only samples that qualify as *faulty data*, this way it fulfils the criterion of homogeneity. At the same time, the samples in this set originate **only** from specific time period *start.time*. Accordingly, Cluster A lies in the cut set of the qualitative domains *RT=faulty* and *start.time=low*. In contrary, Cluster B seems to contain the **entire** class of *border class* samples, as it depends only on RT and the clusters are disjoint. However, V&V indicates overlap with different qualitative domains of *RT* violating the requirement of homogeneity, thus the initial, somewhat arbitrary visual clustering of *RT* was slightly shifted. Hence, the intrinsic capability of checking the assumptions helps to detect even minor mistakes (where after correction the result becomes consistent).

6.3.5 Evaluating the results

The clustering method approximately separated the samples according to the initial engineering viewpoint - by the qualitative magnitude of transfer time *RT*.

It is also essential to understand that the clusters formed by the DR method constructed their own qualitative domain itself, thereby eliminating the need for various visual or statistical analyses to construct them. Reprojecting (Fig. 6.2c) the clusters association with their contained samples refines the initial visual exploratory analysis-based partitioning, and it readjusts the operational region boundaries slightly, assuring a coherent structure of the model.

Additional information - like specific IP addresses or timeframes - may suggest a causal relationship for the unusual delay of the transfer to occur.

Chapter 7

Conclusion

This thesis has tried to provide a general overview of some current, widely used state-of-the-art model interpretability techniques. It has clearly shown that not only opaque but also transparent models can be suitable targets for (further) interpretation. Even though these approaches can present immediate and clear explanations for a user, a selection of these tools can only be used for V&V purposes in regards to CPS systems if in-depth knowledge of these mechanisms is acquired. Therefore a complete workflow showcasing various tools and their use and influence on the system analysis has been proposed for this process.

The concluded examination reinforces and justifies the use of a qualitative model. Moreover, with the mentioned tools, the proposed technique showcased its ability to define a **qualitative membership** function for each operational region and form the boundaries. With the use of XAI methods, cohesive factors and their respective importance can be thoroughly understood. The quality of the resulting separation can be observed by statistics, such as *homogeneity level* and the *separation quality*, and *residual diagnosis* built on top of that knowledge. That allows for interpretable, verifiable, and validatable qualitative model construction. As the research demonstrated, the preliminary manual separation between clusters has been refined as the outcome of the process. Alternatively, for this purpose, any other concept learning method (BRCG or an improved version of LEN) can also be used.

Since each cluster defines states showing similar behaviour, their characteristics can also be extracted and examined with XAI techniques (CP, PDP). That gives further insights regarding each operational state, reveals qualitative relationships between instances, and exposes hazardous states.

The introduction of a solid XAI repertoire to the analysis process has shifted the focus towards the V&V phase from the domain knowledge by forming the qualitative regions as a byproduct of the analysis process.

The analysis results related to the boundaries of the different operation domains are directly relevant in setting up the monitoring part of a hybrid supervisory control scheme. Here the detection of entering a new operation domain is detected by comparators watching the crossing of the boundaries and triggering the control actions, referred to as scenario identification in the technical literature.

CPS can utilise embedded AI components to capture complex behavioural patterns accurately. The thesis has shown that the modelled decision-making process can be interpreted in a very tangible form (either as simple rulesets or some sort of visual representation).

If strict compliance with system requirements are mandatory hard constraints can be incorporated into the model.

The other part of the research has been concerned with AI explainers relying on a concept-based rule learning approach, which greatly impacted the field domain (BRCG, LEN). These algorithms were compared and then dissected to allow their implementation-level analysis. Leveraging its individual problem formulation, a directly explainable classifier is proposed, which was able to surpass its counterpart. However, for efficient use, there are still improvements to be made.

In conclusion, one can say that interpretability techniques have evolved immensely in the last couple of years, and they still manage to showcase new forms of applications in other scientific fields as well. One of the primary intentions of this thesis was to show the diverse purposes requiring some form of interpretability and the solution for them. AI development in real-life applications are far from being responsive, and avoiding confusion, however, the gap between the developers and the models tends to narrow.

Chapter 8

Future work

The most promising approaches reviewed in this thesis to precisely approximate the decision-making process of an arbitrary classifier was all incorporating a concept-based learning approach. If the problem of concept learning would have been solved by a model that is (i) able to tailor the discretisation of the domains according to a specific problem formulation, (ii) and it can be validated by formal methods, then these reviewed XAI tools would have a much larger impact on this scientific field. Furthermore, concept learners would be essential tools for directly quantising a continuous domain into qualitative states. Therefore, investigating these approaches is definitely a primary target of subsequent academic work.

In regards to the proposed Nested cavities algorithm, it currently lacks scalability, due to the global solver. The resource demanding traversal of the exponentially large search space can be improved in two ways. Firstly the input data should be pipelined into the algorithm, and cuts leading the suboptimal solutions could be discarded therefore right away, thereby saving memory. The pipelined data should be generated, outputting instances only, that can improve the cost function according to some heuristics. This idea could prune the search tree. The second approach would be finding a Prolog-like environment that could speed up the traversal marginally. An ideal extension to the problem that could leverage the logical knowledge representation of the explainer would be the field of Inductive Learning of Answer Set Programs (ASP). ASP is a declarative programming paradigm that supplements the proposed algorithm's shortcomings while solving NP-hard search problems as it reduces the search space. This concept should be further investigated in the future to verify its suitability for the given problem formulation.

Further research should examine the question of exhaustive outlier detection, as well as the causal relationships between explained factors and the target variable. Formally proving the correctness of our hypothesis formulated by interpretability metrics is also a key subject of interest.

Moreover the model robustness is unknown if noise is introduced to the system. It should be observed in regards to this aspect, and in case of erroneous behaviour, the model extraction workflow should be refined to be able to handle the noise, to avoid obfuscated model fragments.

The research suggested that the proposed methods can be individually used, or even theoretically jointly as a pipeline for qualitative model extraction. However, there are certain decision points in the process, which requires human interaction, such as determining whether the homogeneity of the derived clusters are sufficient, or causal inference. The

long term goal of the research is to examine the suitability of these techniques and models for automatic qualitative model derivation.

Bibliography

[1] Oludare Isaac Abiodun, Aman Jantan, Abiodun Esther Omolara, Kemi Victoria Dada, Nachaat AbdElatif Mohamed, and Humaira Arshad. State-of-the-art in artificial neural network applications: A survey. *Heliyon*, 4(11):e00938, 2018. ISSN 2405-8440.

[2] Shaeela Ayesha, Muhammad Kashif Hanif, and Ramzan Talib. Overview and comparative study of dimensionality reduction techniques for high dimensional data. *Information Fusion*, 59:44–58, 2020. ISSN 1566-2535.

[3] M. Baak, R. Koopman, H. Snoek, and S. Klous. A new correlation coefficient between categorical, ordinal and interval variables with Pearson characteristics. *Computational Statistics & Data Analysis*, 152:107043, 2020. ISSN 0167-9473.

[4] Juhee Bae and et al. Interactive clustering: A comprehensive review. *ACM Comput. Surv.*, 53(1), Feb. 2020. ISSN 0360-0300.

[5] Alejandro Barredo Arrieta, Natalia Díaz-Rodríguez, Javier Del Ser, Adrien Bennetot, Siham Tabik, Alberto Barbado, Salvador Garcia, Sergio Gil-Lopez, Daniel Molina, Richard Benjamins, Raja Chatila, and Francisco Herrera. Explainable artificial intelligence (xai): Concepts, taxonomies, opportunities and challenges toward responsible ai. *Information Fusion*, 58:82–115, 2020. ISSN 1566-2535.

[6] Tom Beucler, Michael Pritchard, Stephan Rasp, Jordan Ott, Pierre Baldi, and Pierre Gentine. Enforcing analytic constraints in neural-networks emulating physical systems, 2021.

[7] Przemyslaw Biecek and Tomasz Burzykowski. *Explanatory Model Analysis*. Chapman and Hall/CRC, New York, 2021. ISBN 9780367135591. URL `https://pbiecek.github.io/ema/`.

[8] Azzedine Boukerche, Lining Zheng, and Omar Alfandi. Outlier detection: Methods, models, and classification. *ACM Comput. Surv.*, 53(3), jun 2020. ISSN 0360-0300.

[9] Bert Bredeweg, Floris Linnebank, Anders Bouwer, and Jochem Liem. Garp3—workbench for qualitative modelling and simulation. *Ecological informatics*, 4(5-6): 263–281, 2009.

[10] Gabriele Ciravegna, Pietro Barbiero, Francesco Giannini, Marco Gori, Pietro Lió, Marco Maggini, and Stefano Melacci. Logic explained networks, 2021.

[11] Michele Conforti, Gérard Cornuéjols, Giacomo Zambelli, et al. *Integer programming*, volume 271. Springer, 2014.

[12] Sanjeeb Dash, Oktay Günlük, and Dennis Wei. Boolean decision rules via column generation. *arXiv*, 1805.09901, 2020.

[13] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. KDD'96, page 226–231. AAAI Press, 1996.

[14] Kenneth D Forbus. Qualitative modeling. In Frank Van Harmelen, Vladimir Lifschitz, and Bruce Porter, editors, *Handbook of knowledge representation*, volume 3, pages 361–393. Elsevier, 2008.

[15] Stephen D Grantham and Lyle H Ungar. Qualitative physics, 1990.

[16] David Gunning and David Aha. Darpa's explainable artificial intelligence (xai) program. *AI Magazine*, 40(2):44–58, 2019.

[17] Karthik S. Gurumoorthy, Amit Dhurandhar, Guillermo Cecchi, and Charu Aggarwal. Efficient data representation by selecting prototypes with importance weights. *arXiv*, 1707.01212, 2019.

[18] Douglas M Hawkins. *Identification of outliers*, volume 11. Springer, 1980.

[19] Alfred Inselberg. Parallel coordinates: visualization, exploration and classification of high-dimensional data. In *Handbook of data visualization*, pages 643–680. Springer, 2008.

[20] Imre Kocsis, Ágnes Salánki, and András Pataricza. Measurement-based identification of infrastructures for trustworthy cyber-physical systems. In Alexander Romanovsky and Fuyuki Ishikawa, editors, *Trustworthy Cyber-Physical Systems Engineering*, pages 395–420. Chapman and Hall/CRC, 2016.

[21] Benjamin Kuipers. *Qualitative Reasoning: Modeling and Simulation with Incomplete Knowledge*. MIT Press, Cambridge, MA, USA, 1994. ISBN 026211190X.

[22] Vladimir Lifschitz, Leora Morgenstern, and David Plaisted. Knowledge representation and classical logic. In Frank van Harmelen, Vladimir Lifschitz, and Bruce Porter, editors, *Handbook of Knowledge Representation*, volume 3 of *Foundations of Artificial Intelligence*, pages 3–88. Elsevier, 2008.

[23] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of Machine Learning*. The MIT Press, 2nd edition, 2018. ISBN 0262039400.

[24] Emmanuel Müller, Stephan Günnemann, Ira Assent, and Thomas Seidl. Evaluating clustering in subspace projections of high dimensional data. *Proceedings of the VLDB Endowment*, 2(1):1270–1281, 2009.

[25] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach (2nd Edition)*. Prentice Hall, December 2002. ISBN 0137903952.

[26] Peter Struss. Model-based problem solving. In Frank van Harmelen, Vladimir Lifschitz, and Bruce Porter, editors, *Handbook of Knowledge Representation*, volume 3 of *Foundations of Artificial Intelligence*, pages 395–465. Elsevier, 2008.

[27] Fei Tao, Qinglin Qi, Lihui Wang, and A.Y.C. Nee. Digital twins and cyber–physical systems toward smart manufacturing and Industry 4.0, correlation and comparison. *Engineering*, 5(4):653–661, 2019. ISSN 2095-8099.

[28] Yingfan Wang, Haiyang Huang, Cynthia Rudin, and Yaron Shaposhnik. Understanding how dimension reduction tools work: An empirical approach to deciphering t-SNE, UMAP, TriMAP, and PaCMAP for data visualization. *arXiv*, 2012.04456, 2021.

[29] Zhenyu Zhao, Radhika Anand, and Mallory Wang. Maximum relevance and minimum redundancy feature selection methods for a marketing machine learning platform. In *2019 IEEE Intl. Conf. on Data Science and Advanced Analytics (DSAA)*, pages 442–452, 2019.

Appendix

A.1 Machine Learning

A.1.1 Phone Price Classification dataset

The dataset is a collection of samples describing mobile phones on a hypothetical market with various specifications. The problem requires the assignment of price ranges (extra high, high, medium, low) to each phone, revealing features that justify a phone's price on the market. Usually the problem is shrunk into a binary classification problem with $Y \rightarrow \{\{extrahigh, high\} \rightarrow high, \{medium, low\} \rightarrow low\}$.

The dataset consists of 2000 instances, and it is completely balanced with 1000 positive and 1000 negative samples. It has 20 features describing the following aspects of a phone: `battery_power`: Total energy capacity of the battery measured in mAh; `blue`: Possession of a bluetooth module (binary); `clock_speed`: speed at which microprocessor executes instructions measured in GHz; `dual_sim`: Possession of dual sim support (binary); `fc`: Resolution of the front camera in MP; `pc`: Resolution of the rear/primary camera in MP; `four_g`: Possession of 4G support (binary); `three_g`: Possession of 3G support (binary); `int_memory`: Internal Memory in GB; `m_dep`: Physical depth of the mobile phone measured in cm; `mobile_wt`: Weight of mobile phone measured in grams; `n_cores`: Number of cores of the processor; `px_height`: Height dimension of the resolution in pixels; `px_width`: Width dimension of the resolution in pixels; `ram`: Random Access Memory capacity in Megabytes; `sc_h`: Height of the mobile's screen in cm; `sc_w`: Width of the mobile's screen in cm; `talk_time`: Longest time that a single battery charge will last when the user is constantly talking on the phone; `wifi`: Possession of a wifi module (binary); creen (binary).

The dataset originates from kaggle.[1]

A.1.2 Decision & Linear trees comparision

[81]:
```
import pandas as pd
from sklearn.linear_model import *
from lineartree import LinearTreeClassifier, LinearTreeRegressor
from sklearn.model_selection import GridSearchCV
from sklearn import tree
import numpy as np
import matplotlib.pyplot as plt
```

[1] https://www.kaggle.com/iabhishekofficial/mobile-price-classification

```
from sklearn.metrics import f1_score
from sklearn.metrics import accuracy_score
```

Loading the data

[82]:
```
data = pd.read_csv("data/phone/train.csv")
```

merging the categories: {extra high, high, medium, low} -> {high + low}

[83]:
```
data.loc[data["price_range"] == 1, "price_range"] = 0
data.loc[data["price_range"] == 3, "price_range"] = 1
data.loc[data["price_range"] == 2, "price_range"] = 1
```

[84]:
```
data.price_range.unique()
```

[84]:
```
array([0, 1])
```

splitting the dataset into train + test

[85]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(data.iloc[:, :20],␣
↪data.iloc[:, 20], test_size=0.25, random_state=42)
```

[86]:
```
import time

metrics = {
    "lin_tree": {"accuracy": 0, "cpu": 0, "complexity": 0, "fscore": 0},
    "dec_tree": {"accuracy": 0, "cpu": 0, "complexity": 0, "fscore": 0}
}
```

Fitting the decision tree with max_depth = 4

[87]:
```
dTree = tree.DecisionTreeClassifier(max_depth=3)

start = time.time()
dTree.fit(X_train, y_train)
end = time.time()

plt.figure(figsize=(15,10))  # set plot size (denoted in inches)
tree.plot_tree(dTree, fontsize=11)
plt.show()
```

Evaluation

[88]:
```
metrics["dec_tree"]["cpu"] = end - start
metrics["dec_tree"]["accuracy"] = accuracy_score(y_test, dTree.
 ↪predict(X_test))
metrics["dec_tree"]["fscore"] = f1_score(y_test, dTree.predict(X_test),␣
 ↪average='binary')
```

[89]:
```
print("Original Decision tree F1-Score")
print(f1_score(y_test, dTree.predict(X_test), average=None))

print("Accuracy")
print(accuracy_score(y_test, dTree.predict(X_test)))
```

```
Original Decision tree F1-Score
[0.91313131 0.91485149]
Accuracy
0.914
```

Rule extraction

[90]:
```
text_representation = tree.export_text(dTree)
print(text_representation)
```

```
|--- feature_13 <= 2217.50
|   |--- feature_13 <= 1509.50
|   |   |--- feature_11 <= 1917.00
|   |   |   |--- class: 0
|   |   |--- feature_11 >  1917.00
|   |   |   |--- class: 1
|   |--- feature_13 >  1509.50
|   |   |--- feature_0 <= 1461.50
|   |   |   |--- class: 0
|   |   |--- feature_0 >  1461.50
|   |   |   |--- class: 1
|--- feature_13 >  2217.50
|   |--- feature_13 <= 2653.00
|   |   |--- feature_0 <= 933.00
|   |   |   |--- class: 0
|   |   |--- feature_0 >  933.00
|   |   |   |--- class: 1
|   |--- feature_13 >  2653.00
|   |   |--- feature_0 <= 569.50
|   |   |   |--- class: 1
|   |   |--- feature_0 >  569.50
|   |   |   |--- class: 1
```

Complexity

[91]:
```
def tree_complexity(rules):
    complexity = 0
    sub_rules = rules.split("\n")
```

```
    for ssr in sub_rules:
        if 'class' in ssr:
            complexity += ssr.count('|') - 1

    return complexity
```

[92]:
```
tree_complexity(text_representation)
```

[92]: 24

[93]:
```
metrics["dec_tree"]["complexity"] = tree_complexity(text_representation)
```

Fitting the linear tree

[94]:
```
lTree = LinearTreeClassifier(base_estimator=RidgeClassifier())

start = time.time()
lTree.fit(X_train, y_train)
end = time.time()

lTree.plot_model()
```

[94]:

Evaluation

[95]:
```
metrics["lin_tree"]["cpu"] = end - start
metrics["lin_tree"]["accuracy"] = accuracy_score(y_test, lTree.
 ↪predict(X_test))
metrics["lin_tree"]["fscore"] = f1_score(y_test, lTree.predict(X_test),␣
 ↪average='binary')
```

[96]:
```
print("Original Decision tree F1-Score")
print(f1_score(y_test, lTree.predict(X_test), average=None))

print("Accuracy")
print(accuracy_score(y_test, lTree.predict(X_test)))
```

```
Original Decision tree F1-Score
[0.97465887 0.97330595]
Accuracy
0.974
```

Rule extraction For rule-extraction we can use basic algorithms that are used in regular decision-trees until we reach the leaf node. To interpret the leaves heuristics can be used, just like how it was proposed in this paper: *Generating Rule Sets from Model Trees, G. Holmes* (https://link.springer.com/chapter/10.1007/3-540-46695-9_1) Where the main idea is: Model trees are built repeatedly and the best rule is selected at each iteration.

This method produces rule sets that are as accurate but smaller than the model tree constructed from the entire dataset.

In this notebook rules were only mined until the leaf.

[97]:
```
def traverse_tree(model, node, direction, rules, curr_rule):
    children = node.get("children")
    if not children:
        if direction == "right" and curr_rule:
            rules.append(curr_rule)

        return

    c_rule_l = curr_rule.copy()
    l_node = model[children[0]]
    c_rule_l.append(f"X[{node['col']}] <= {node['th']}")
    traverse_tree(model, l_node, "left", rules, c_rule_l)
    c_rule_r = curr_rule.copy()
    r_node = model[children[1]]
    c_rule_r.append(f"X[{node['col']}] > {node['th']}")
    traverse_tree(model, r_node, "right", rules, c_rule_r)


def extract_rules(tree):
    ruleset = []

    root = tree[0]
    traverse_tree(tree, root, "mid", ruleset, [])
    print(ruleset)
```

[98]:
```
def get_rules(model, feature_names=None):

    summary = model.summary(feature_names=feature_names)
    leaves = model.summary(only_leaves=True).keys()
    rules = {}

    for leaf in leaves:
        final_leaf = leaf
        conditions = []
        while leaf != 0:
            for l,info in summary.items():
                if 'children' in info:
                    if leaf in info['children']:
                        th = info['th']
                        col = (info['col'] if feature_names is not None␣
↪else
                               "X[{}]".format(info['col']))
                        direct = '<=' if leaf == info['children'][0]␣
↪else '>'
                        cond = '({} {} {})'.format(col, direct, th)
                        conditions.append(cond)
```

```
                        leaf = l

        conditions = list(reversed(conditions))
        rules['leaf {}'.format(final_leaf)] = ' AND '.join(conditions)

    return rules
```

[99]:
```
rules = get_rules(lTree, list(X_train.columns))
rules
```

[99]:
```
{'leaf 3': '(ram <= 2717.0) AND (ram <= 1474.5) AND (battery_power <=␣
↪1108.5)',
 'leaf 4': '(ram <= 2717.0) AND (ram <= 1474.5) AND (battery_power >␣
↪1108.5)',
 'leaf 6': '(ram <= 2717.0) AND (ram > 1474.5) AND (ram <= 2129.5)',
 'leaf 7': '(ram <= 2717.0) AND (ram > 1474.5) AND (ram > 2129.5)',
 'leaf 9': '(ram > 2717.0) AND (battery_power <= 1169.0)',
 'leaf 10': '(ram > 2717.0) AND (battery_power > 1169.0)'}
```

L is the complexity of a leaf-level linear model which considered to be |X| as it takes into account all features weighted with according coefficients in the prediction.

[100]:
```
L = len(data.columns)
```

[101]:
```
rule_complexity = sum([len(sr.split("AND")) + L for sr in rules.
↪values()])
rule_complexity
```

[101]:
```
142
```

[102]:
```
metrics["lin_tree"]["complexity"] = rule_complexity
```

Analysing decision making in the leaves

[103]:
```
X = X_test
i = 9
leaf = lTree.summary()[i]

palette = plt.get_cmap('tab20').colors

if hasattr(leaf['models'], 'coef_'):
    coef_ = leaf['models'].coef_[0]
    order = np.argsort(np.abs(coef_))
else:
    coef_ = np.zeros((X.shape[1],))
    order = np.arange(X.shape[1])

text = rules['leaf {}'.format(i)].split(' AND ')

plt.figure(figsize=(16,5))
```

```
plt.subplot(121)
plt.axvline(0, linestyle='--', c='black', alpha=0.4)
print(coef_[order])
plt.barh(list(range(X.shape[1])), coef_[order],␣
 ↪color=[palette[1%len(palette)]])
plt.yticks(range(X.shape[1]), X.columns[order], size=13)
plt.title('leaf {}'.format(i), size=15); plt.xlabel('coefficients',␣
 ↪size=13)

plt.subplot(122)
__ = 0.1
for rule in text:
    rule = rule if rule == text[-1] else rule + ' AND'
    plt.text(0.01, 1-__, rule, size=15)
    __ += 0.1
plt.title('rules leaf {}'.format(i), size=15); plt.axis('off')

plt.show()
```

```
[-4.51961588e-07  6.07239890e-05 -6.20882429e-05  6.58821419e-05
  1.03897162e-04 -2.23605694e-04  6.84806828e-04 -9.09234574e-04
 -1.16945998e-03 -1.90741283e-03 -3.83574376e-03 -5.51620577e-03
  6.01182025e-03 -8.37656843e-03  9.18720894e-03  1.69416231e-02
  3.06403208e-02 -3.43939921e-02  3.50377601e-02 -3.73754101e-02]
```

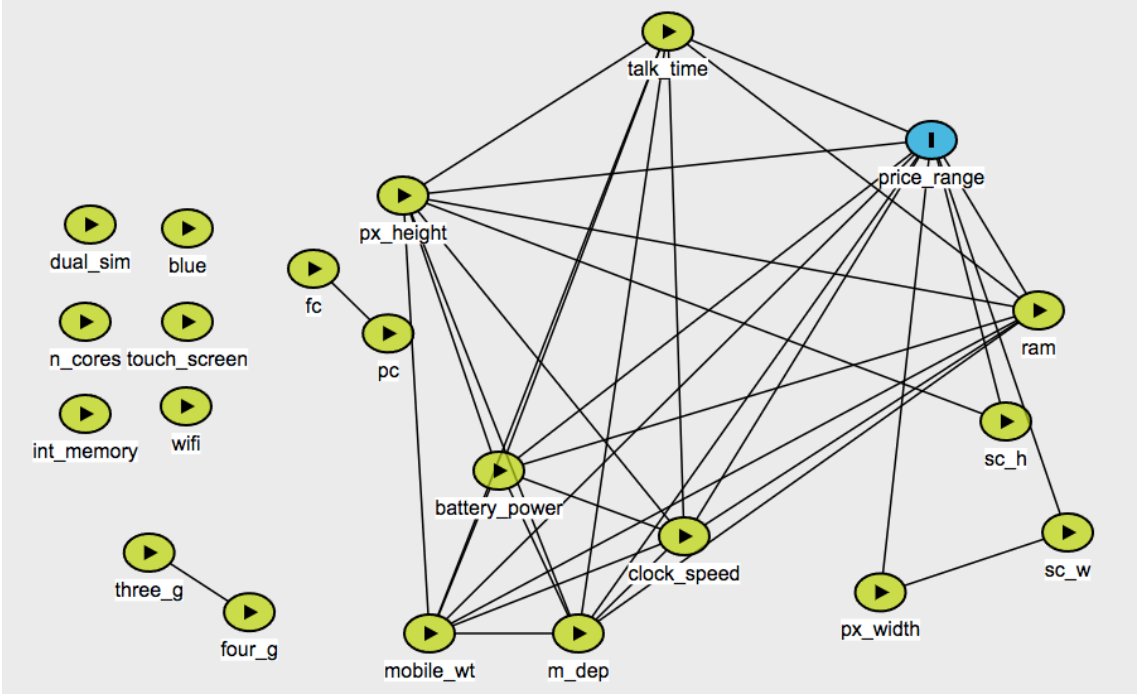
Comparision

[107]:
```
pd.DataFrame.from_dict(metrics, orient='index')
```

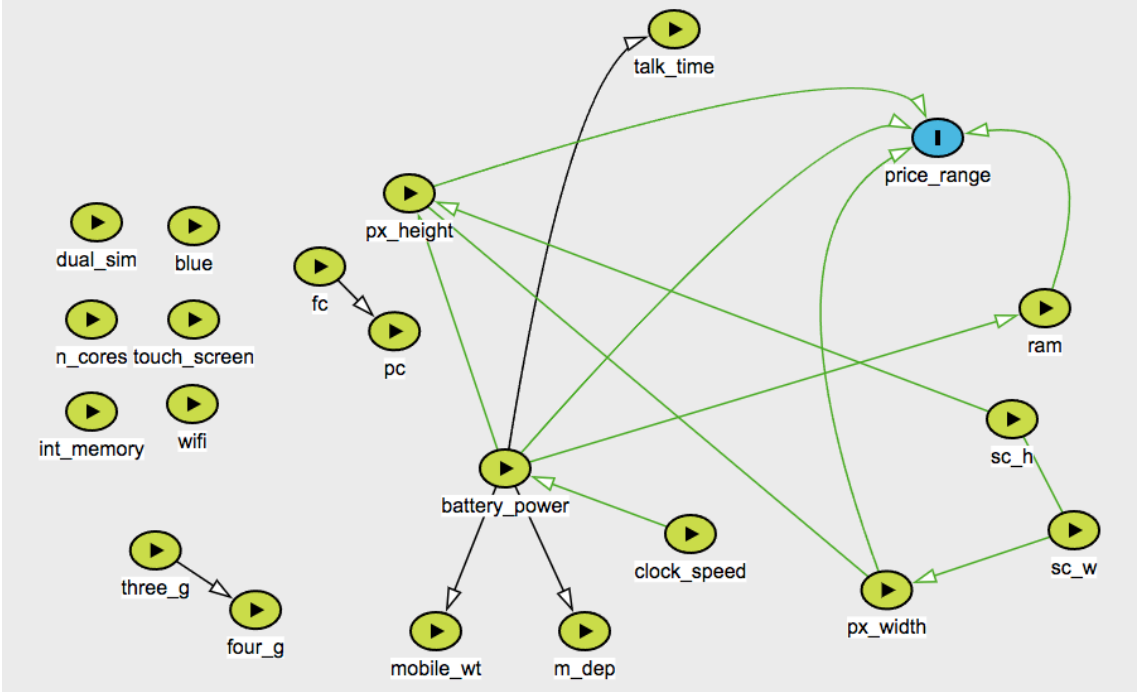
[107]:

	accuracy	cpu	complexity	fscore
lin_tree	0.974	2.626100	142	0.973306
dec_tree	0.914	0.006164	24	0.914851

A.2 Explainable Artificial Intelligence



(a) Occurring correlation associations in the Phone Price Classification problem



(b) Occurring causal relationships in the Phone Price Classification problem

Figure A.2.1. Showing the difference between correlation and causality on the PPC dataset.

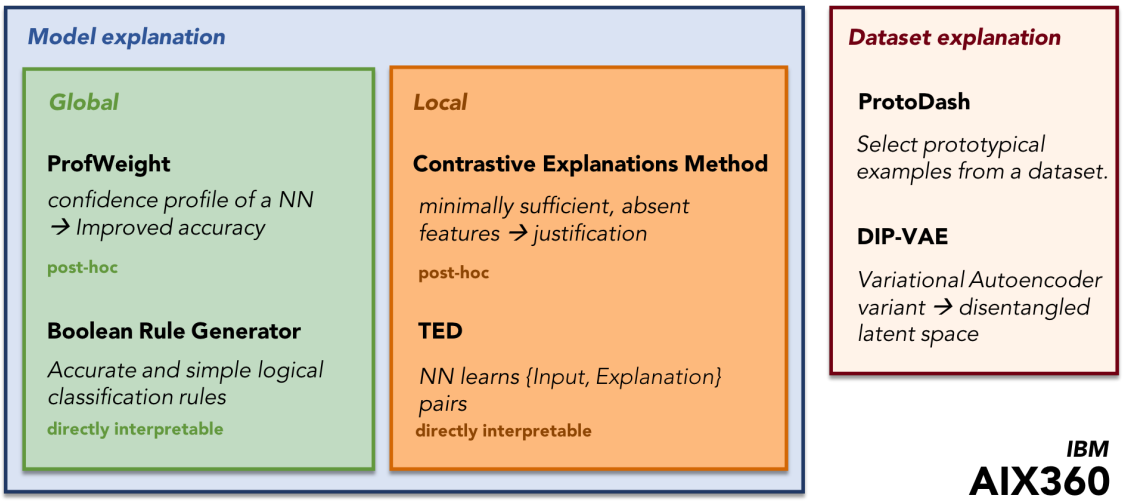


Figure A.2.2. Overview of the algorithms available in IBM AIX360 framework.

A.2.1 Training a Neural Network on the PPC dataset

Importing modules

[1]:
```
from sklearn.linear_model import *
from lineartree import LinearTreeClassifier, LinearTreeRegressor
from sklearn.model_selection import GridSearchCV
from sklearn import tree
import matplotlib.pyplot as plt

import itertools
import numpy as np
import pandas as pd
```

[2]:
```
from sklearn.metrics import f1_score
from sklearn.metrics import accuracy_score
```

[3]:
```
import torch
import torch.nn as nn
from torch.autograd import Variable
from torch_explain.logic.nn import entropy
from torch.nn.functional import one_hot
```

[4]:
```
import torch_explain as te
from torch_explain.logic.nn import psi
```

[79]:
```
import os
from IPython.display import Image

if not os.path.exists("temp-images"):
    os.mkdir("temp-images")
```

Loading the data

[5]:
```
data = pd.read_csv("data/phone/train.csv")
```

[6]:
```
data.loc[data["price_range"] == 1, "price_range"] = 0
data.loc[data["price_range"] == 3, "price_range"] = 1
data.loc[data["price_range"] == 2, "price_range"] = 1
data.price_range.unique()
```

[6]:
```
array([0, 1])
```

[7]:
```
from sklearn.model_selection import train_test_split
Xi_train, Xi_test, yi_train, yi_test = train_test_split(data.iloc[:, :
↪20], data.iloc[:, 20], test_size=0.25, random_state=42)
```

[8]:
```
from torch.utils.data import DataLoader

featuresTrain = torch.tensor(Xi_train.values)
targetsTrain = torch.tensor(yi_train.values).type(torch.LongTensor)

featuresTest = torch.tensor(Xi_test.values)
targetsTest = torch.tensor(yi_test.values).type(torch.LongTensor)

batch_size = 100

train = torch.utils.data.TensorDataset(featuresTrain,targetsTrain)
test = torch.utils.data.TensorDataset(featuresTest,targetsTest)

train_loader = DataLoader(train, batch_size = batch_size, shuffle =␣
↪True)
test_loader = DataLoader(test, batch_size = batch_size, shuffle = True)
```

Defining the NN

[9]:
```
class BlackBoxModel(nn.Module):

    def __init__(self, input_d, hidden_d, output_d):
        super(BlackBoxModel, self).__init__()

        self.expL = te.nn.EntropyLinear(input_d, hidden_d, n_classes=2)
        self.expRelu = torch.nn.LeakyReLU()

        self.fc1 = nn.Linear(hidden_d, hidden_d)
        self.relu = nn.ReLU()
        #self.sigmoid1 = nn.Sigmoid()

        self.fc2 = nn.Linear(hidden_d, hidden_d)
        self.sigmoid2 = nn.Sigmoid()
```

```
        self.fc3 = nn.Linear(hidden_d, hidden_d)
        self.elu = nn.ELU()
        #self.sigmoid3 = nn.Sigmoid()

        self.fc4 = nn.Linear(hidden_d, output_d)

    def forward(self, x):

        out = self.expL(x)
        out = self.expRelu(out)

        out = self.fc1(out)
        out = self.relu(out)

        #out = self.sigmoid1(out)

        out = self.fc2(out)
        out = self.sigmoid2(out)

        out = self.fc3(out)
        out = self.elu(out)

        #out = self.sigmoid3(out)

        out = self.fc4(out)

        return out


input_d = len(Xi_train.columns)
hidden_d = 120
output_d = 1


model = BlackBoxModel(input_d, hidden_d, output_d)

# Cross Entropy Loss
error = nn.BCEWithLogitsLoss()

# SGD Optimizer
learning_rate = 0.0001
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
```

Train

[10]:
```
import time

def epoch_time(start_time, end_time):
    elapsed_time = end_time - start_time
```

```
    elapsed_mins = int(elapsed_time / 60)
    elapsed_secs = elapsed_time - (elapsed_mins * 60)
    return elapsed_mins, elapsed_secs
```

[11]:
```
def calculate_performance(preds, y):
    preds = torch.squeeze(preds)
    y = torch.squeeze(y)
    y_pred_tag = torch.round(torch.sigmoid(preds))
    #correct_results_sum = (y_pred_tag == y).sum().float()
    sum = 0
    for i in range(len(y_pred_tag)):
        if torch.eq(y_pred_tag[i], y[i]).all():
            sum += 1

    correct_results_sum = torch.tensor(float(sum))
    acc = correct_results_sum/y.shape[0]

    acc = torch.round(acc * 100)
    if acc > 100 or acc < 0:
        print(preds)
        print(y)
        print(y_pred_tag)
        print("CORR: ")
        print(correct_results_sum)
        print(acc)
        assert()
    return acc
```

[12]:
```
def evaluate(model, loader):
    accuracy = 0
    total = 0
    count = 0
    val_loss = 0
    model.eval()
    with torch.no_grad():
        for features, labels in loader:
            test = Variable(features)
            v_labels = Variable(labels)
            outputs = model(test.float()).squeeze(-1)
            o_labels = one_hot(v_labels).to(float)
            loss = error(outputs, o_labels) + 0.0001 * te.nn.functional.
 ↪entropy_logic_loss(model)
            val_loss += loss.data
            accuracy += calculate_performance(outputs.squeeze(-1),␣
 ↪o_labels)
            count += 1
            optimizer.step()

    accuracy = accuracy / float(count)
    val_loss = val_loss / float(count)
```

```
    return accuracy, val_loss
```

[13]:
```
num_epochs = 100
best_valid_losses = [float('inf'), float('inf')]

train_loss_list = []
val_loss_list = []
train_acc_list = []
val_acc_list = []

for epoch in range(num_epochs):

    curr_loss = 0
    curr_acc = 0
    count = 0

    start_time = time.time()
    model.train()
    for i, (features, labels) in enumerate(train_loader):

        train = Variable(features)
        labels = Variable(labels)

        optimizer.zero_grad()
        outputs = model(train.float()).squeeze(-1)

        o_labels = one_hot(labels).to(float)

        loss = error(outputs, o_labels) + 0.1 * te.nn.functional.
↪entropy_logic_loss(model)
        loss.backward()
        optimizer.step()
        count += 1
        curr_loss += loss.data
        curr_acc += calculate_performance(outputs.squeeze(-1), o_labels)


    end_time = time.time()
    epoch_mins, epoch_secs = epoch_time(start_time, end_time)

    train_accuracy = curr_acc / float(count)
    train_loss = curr_loss / float(count)
    train_loss_list.append(train_loss)
    train_acc_list.append(train_accuracy)
    val_accuracy, val_loss = evaluate(model, test_loader)
    val_loss_list.append(val_loss)
    val_acc_list.append(val_accuracy)
    """
```

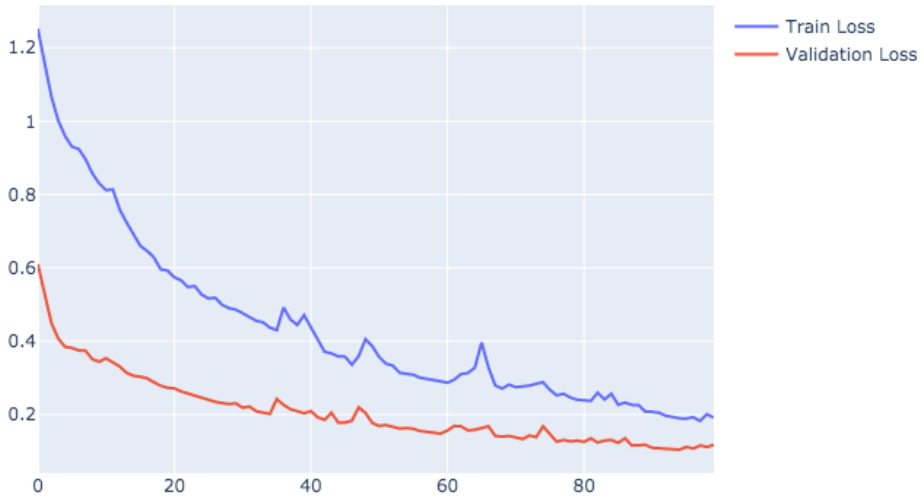
```
    if best_valid_losses[0] < float('inf') and best_valid_losses[0] <␣
↪float('inf') and \
            best_valid_losses[0] > best_valid_losses[1] and val_loss  >␣
↪best_valid_losses[1]:
                break

    best_valid_losses[0] = best_valid_losses[1]
    best_valid_losses[1] = val_loss


    print(f'Epoch: {epoch+1:02} | Epoch Time: {epoch_mins}m␣
↪{epoch_secs}s')
    print(f'\tTrain Loss: {train_loss:.3f} | Train Accuracy:␣
↪{train_accuracy:.2f}%')
    print(f'\t Val. Loss: {val_loss:.3f} |  Val. Accuracy:␣
↪{val_accuracy:.2f}%')
     """
```

[14]:
```
max(val_acc_list)
```

[14]:
```
tensor(97.)
```

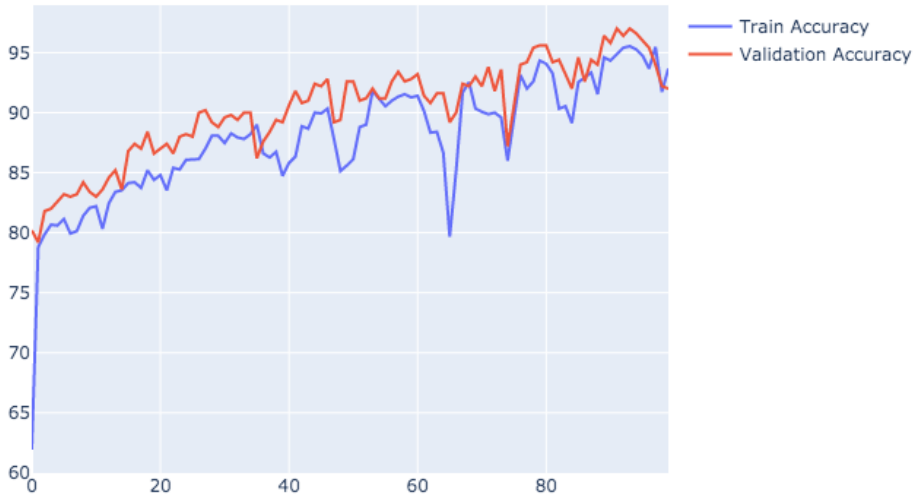
[82]:
```
import plotly.graph_objects as go
fig1 = go.Figure()
fig1.add_trace(go.Scatter(y=train_loss_list,
                    mode='lines',
                    name='Train Loss'))
fig1.add_trace(go.Scatter(y=val_loss_list,
                    mode='lines',
                    name='Validation Loss'))


fig1.write_image("temp-images/fig1.png")
Image("temp-images/fig1.png")
```

[82]:



[83]:
```
fig2 = go.Figure()
fig2.add_trace(go.Scatter(y=train_acc_list,
                    mode='lines', name='Train Accuracy'))
fig2.add_trace(go.Scatter(y=val_acc_list,
                    mode='lines', name='Validation Accuracy'))
fig2.write_image("temp-images/fig2.png")
Image("temp-images/fig2.png")
```

[83]:



Relabelling

[17]:
```
preds = model(torch.tensor(data.drop(columns=['price_range']).values).
↪float())
```

[19]:
```
data["NN"] = torch.round(torch.sigmoid(preds)).view(-1)[1::2].detach().
↪numpy()
```

AIX Rulegenerator

[20]:
```
import aix360.algorithms.rbm
from aix360.algorithms.rbm import FeatureBinarizer
from aix360.algorithms.rbm import BooleanRuleCG

from sklearn.metrics import accuracy_score
```

[ ]:
```
from sklearn.model_selection import train_test_split
Xi_train, Xi_test, yi_train, yi_test = train_test_split(data.iloc[:, :
↪20], data.iloc[:, 21], test_size=0.25, random_state=42)

fb = FeatureBinarizer(negations=True, returnOrd=True)

X_bin, X_std = fb.fit_transform(Xi_train)
X_test_bin, _ = fb.fit_transform(Xi_test)
Y = yi_train
```

```
br_model = BooleanRuleCG(1e-2, 1e-3, CNF=False)
br_model.fit(X_bin, Y)

print("Rules")
print(br_model.explain()["rules"])
print(f'Accuracy: {accuracy_score(Y, br_model.predict(X_bin))}')
print()
```

cleared cell output to spare space

[24]:
```
rules = br_model.explain()["rules"]
```

[32]:
```
rules
```

[32]:
```
['ram > 2479.80',
 'px_height > 223.00 AND px_width > 1244.00 AND ram > 2129.50',
 'battery_power > 1403.00 AND px_width > 669.90 AND ram > 1751.20']
```

[28]:
```
br_model.score(X_bin, Y)
```

[28]:
```
0.942
```

AIX ProtoDash

[33]:
```
pos = data[data["NN"] == 1]
neg = data[data["NN"] == 0]
```

[34]:
```
from aix360.algorithms.protodash import ProtodashExplainer
```

[35]:
```
explainer = ProtodashExplainer()
weights_neg, slices_neg, _ =explainer.explain(neg.values, data.values,␣
↪m=1)
weights_pos, slices_pos, _ =explainer.explain(pos.values, data.values,␣
↪m=1)
```

Negative prototype

[54]:
```
data.loc[[slices_neg]]
```

[54]:
```
      battery_power  blue  clock_speed  dual_sim  fc  four_g  int_memory␣
↪ \
1153           1526     0          1.1         1   2       1          38

      m_dep  mobile_wt  n_cores  …  px_width   ram  sc_h  sc_w ␣
↪talk_time  \
1153    0.3        116        5  …      1418  3317     7     1          8

      three_g  touch_screen  wifi  price_range   NN
1153        1             1     1            0  0.0
```

```
[1 rows x 22 columns]
```

Positive prototype

[55]:
```
data.loc[[slices_pos]]
```

[55]:
```
   battery_power  blue  clock_speed  dual_sim  fc  four_g  int_memory  m_dep  \
7           1954     0          0.5         1   0       0          24    0.8

   mobile_wt  n_cores  …  px_width  ram  sc_h  sc_w  talk_time  three_g  \
7        187        4  …      1149  700    16     3          5        1

   touch_screen  wifi  price_range   NN
7             1     1            0  0.0

[1 rows x 22 columns]
```

Dalex

[57]:
```
import dalex as dx
```

[58]:
```
def predict_function(model, data):

    #data_transformed = preprocessor.transform(data)
    #data_tensor = torch.from_numpy(data_transformed).float()
    preds = model(torch.tensor(data.values).float())
    return (torch.sigmoid(preds).view(-1)[1::2]).detach().numpy()
```

[59]:
```
explainer = dx.Explainer(
    model,
    data=data.drop(columns=['price_range', 'NN']),
    y= data["price_range"],
    predict_function = predict_function,
    model_type = "classification",
    label = "Black-box"
    )
```

```
Preparation of a new explainer is initiated

  -> data              : 2000 rows 20 cols
  -> target variable   : Parameter 'y' was a pandas.Series. Converted to a
numpy.ndarray.
  -> target variable   : 2000 values
  -> model_class       : __main__.BlackBoxModel (default)
  -> label             : Black-box
```

```
  -> predict function  : <function predict_function at 0x1618ec9e0> will␣
↪be used
  -> predict function  : Accepts only pandas.DataFrame, numpy.ndarray␣
↪causes
problems.
  -> predicted values  : min = 4.63e-06, mean = 0.532, max = 1.0
  -> model type        : classification will be used
  -> residual function : difference between y and yhat (default)
  -> residuals         : min = -1.0, mean = -0.0329, max = 0.799
  -> model_info        : package __main__

A new explainer has been created!
```

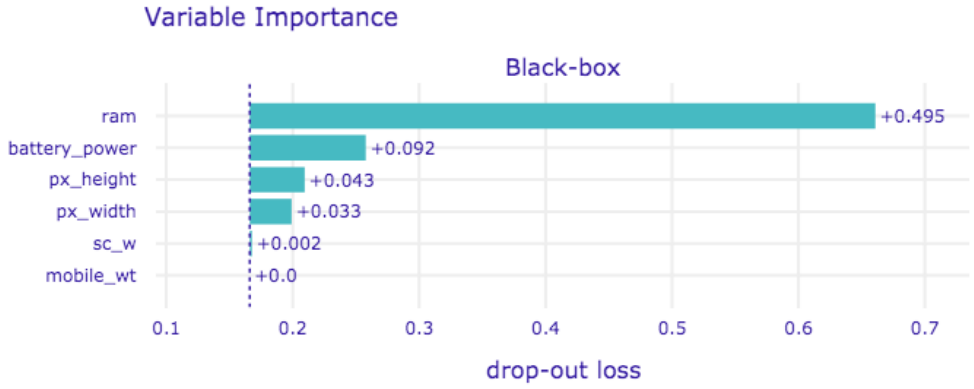
Global explanations

Variable Importance

[60]:
```
mp = explainer.model_parts(loss_function = 'rmse')
```

[90]:
```
mp.plot(max_vars = 6, show=False).write_image("temp-images/vi.png")
Image("temp-images/vi.png")
```

[90]:



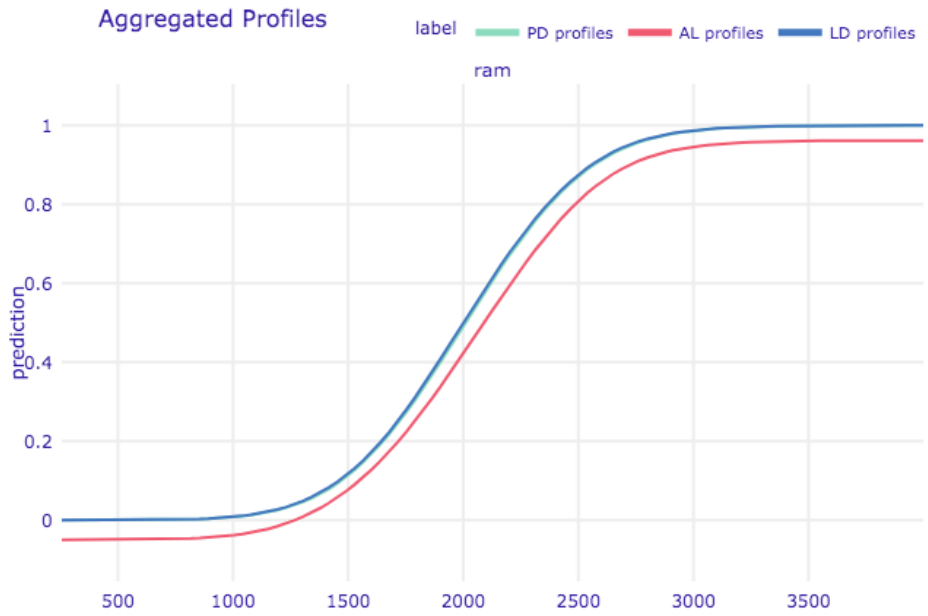
Partial dependence profile

[62]:
```
pd_profile = explainer.model_profile(variables = ["ram"] , type =␣
↪"partial")
pd_profile.result['_label_'] = 'PD profiles'
ld_profile = explainer.model_profile(variables = ["ram"] , type =␣
↪"conditional")
ld_profile.result['_label_'] = 'LD profiles'
ad_profile = explainer.model_profile(variables = ["ram"] , type =␣
↪"accumulated")
ad_profile.result['_label_'] = 'AL profiles'
```

```
Calculating ceteris paribus: 100%|      | 1/1 [00:00<00:00,  3.70it/s]
Calculating ceteris paribus: 100%|      | 1/1 [00:00<00:00,  4.37it/s]
Calculating conditional dependency: 100%|      | 1/1 [00:00<00:00,
13.64it/s]
Calculating ceteris paribus: 100%|      | 1/1 [00:00<00:00,  4.46it/s]
Calculating accumulated dependency: 100%|      | 1/1 [00:00<00:00,
7.36it/s]
```

[91]:
```
pd_profile.plot([ad_profile, ld_profile], show=False).
↪write_image("temp-images/pdp.png")
Image("temp-images/pdp.png")
```

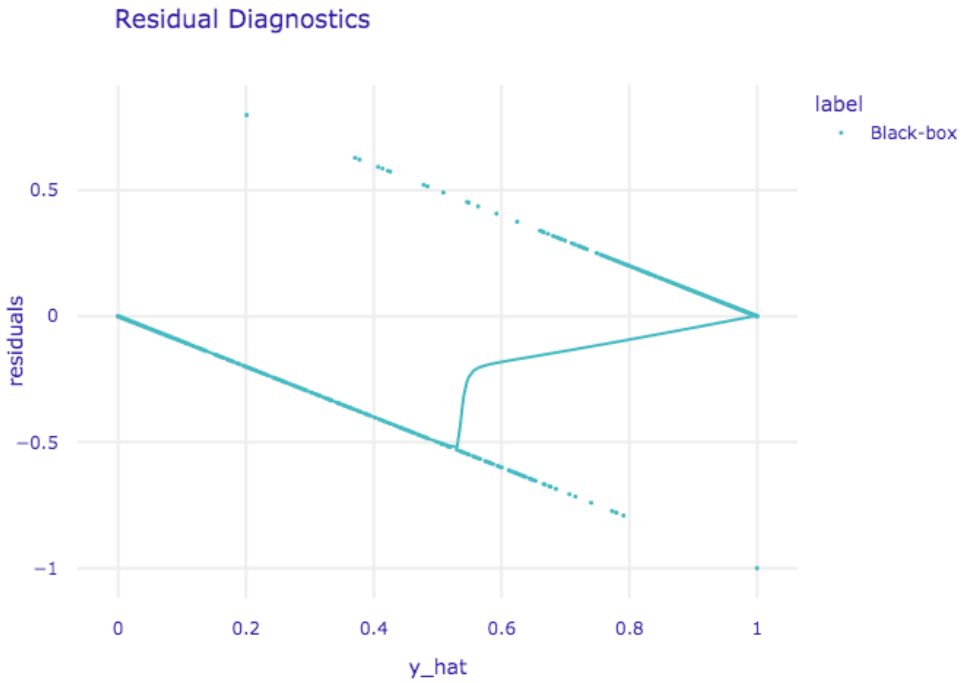
[91]:



Residual Diagnostics

[92]:
```
explainer.model_diagnostics().plot(show=False).write_image("temp-images/
↪rd.png")
Image("temp-images/rd.png")
```

[92]:



Local explanations

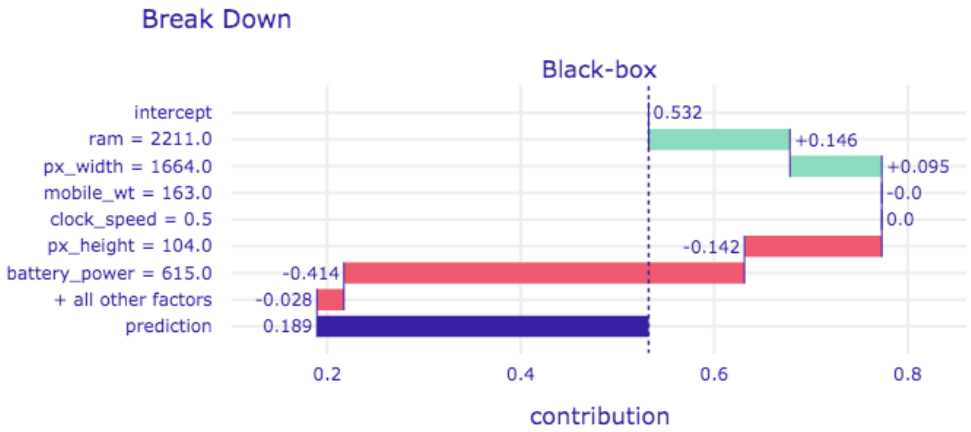
[73]:
```
observation = neg.drop(columns=["price_range", "NN"]).loc[[289]]
observation
```

[73]:
```
     battery_power  blue  clock_speed  dual_sim  fc  four_g  int_memory ␣
↪\
289            615     1          0.5         1   0       0          42

     m_dep  mobile_wt  n_cores  pc  px_height  px_width   ram  sc_h ␣
↪sc_w  \
289    0.6        163        6   4        104      1664  2211     7     4

     talk_time  three_g  touch_screen  wifi
289         18        0             0     1
```

[93]:
```
rf_pparts = explainer.predict_parts(new_observation = observation, type␣
↪= "break_down", order=['ram',
'px_width',

'mobile_wt',
'clock_speed',
'px_height',
'battery_power',
'three_g',
'blue',
'touch_screen',
```

```
 'dual_sim',
 'm_dep',
 'four_g',
 'int_memory',
 'talk_time',
 'sc_h',
 'pc',
 'fc',
 'sc_w',
 'wifi',

 ↪                        ])

# plot Break Down
rf_pparts.plot(max_vars = 6, show=False).write_image("temp-images/bd.
 ↪png")
Image("temp-images/bd.png")
```

[93]:

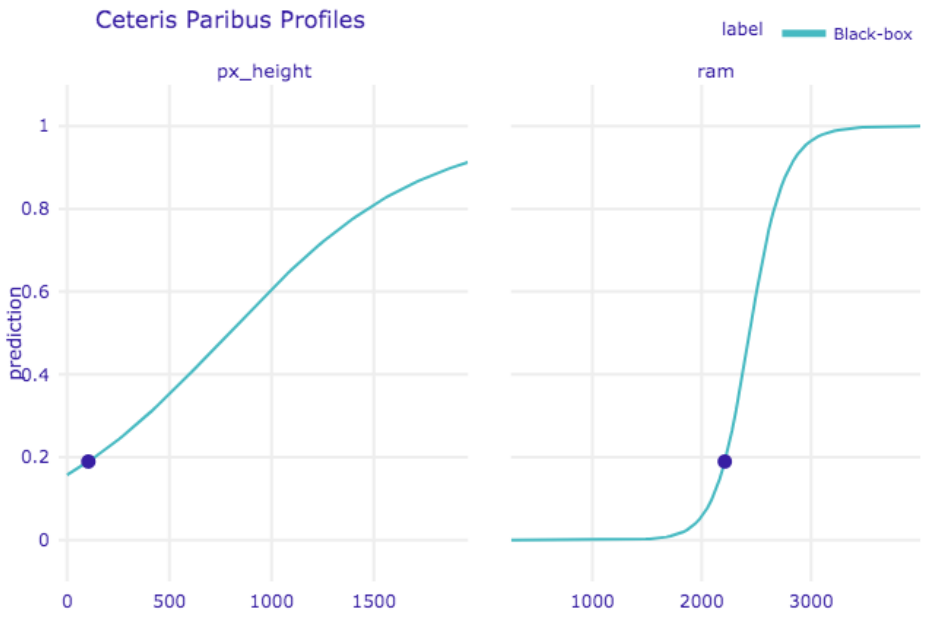


[75]:
```
ceteris_p = explainer.predict_profile(new_observation = observation,␣
 ↪variables=["px_height", "ram"])
```

```
Calculating ceteris paribus: 100%|      | 2/2 [00:00<00:00, 141.16it/s]
```

[94]:
```
ceteris_p.plot(show=False).write_image("temp-images/cp.png")
Image("temp-images/cp.png")
```

[94]:



A.2.2 Logic Explained Networks

The snippet in this section is only an extension to Section A.2.1, to showcase how to make the NN explainable with Logic Explained Networks. Methods that differ from the ones described there: binarising the input (which will be loaded into tensors), extracting explanation from the entropy layer with LEN.

Binarising the input

[8]:
```
categorical_features = ['blue', 'dual_sim', 'four_g','three_g',␣
↪'touch_screen', 'wifi']
fb = FeatureBinarizer(colCateg=categorical_features, negations=False,␣
↪returnOrd=False, threshStr=False)

X_bin = fb.fit_transform(data.iloc[:, :20])
```

Explaining the NN

[38]:
```
from torch_explain.logic.nn import entropy
from torch.nn.functional import one_hot

X = featuresTrain
Y = targetsTrain
```

```
y1h = one_hot(Y)
explanation, _ = entropy.explain_class(model, X.float(), y1h.float(), X.
↪float(), y1h.float(), target_class=1)
```

[39]:
```
from torch_explain.logic.metrics import test_explanation, complexity

accuracy, preds = test_explanation(explanation, X.float(), y1h.float(),␣
↪target_class=1)
explanation_complexity = complexity(explanation)
```

[40]:
```
print(accuracy)
print(preds)
print(explanation_complexity)
```

```
0.9079985279764475
[ True  True  True … False  True False]
1
```

[41]:
```
explanation
```

[41]:
```
'~feature0000000093'
```

[42]:
```
print(X_bin.columns[93])
```

```
('ram', '<=', 2146.5)
```

A.3 Inside the core of a learning algorithm

A.3.1 BRCG analysis

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn import metrics
# ignore warning related to deprecated modules inside packages
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
warnings.simplefilter(action='ignore', category=UserWarning)
# train with fixed seed to eliminate different results
np.random.seed(777)
from IPython.display import Image
```

Rulesets

Below as an example, there is a simple ruleset which aims to approximate a model which can classify the samples correctly, according to their statuses:

[7]:
```
dnf = [
    'ssc_p > 58.00 AND workex  ',
    'hsc_p > 52.00 AND degree_p > 65.00',
    'ssc_p > 64.00 AND hsc_p > 65.00',
    'ssc_p > 52.00 AND hsc_p > 52.00 AND degree_t != Sci&Tech AND mba_p␣
 ↪<= 58.78'
]
```

Therefore some applications of rulesets are required to remain manageable, that is why sometimes it is necessary to formulate constraints on their complexity. This parameter takes 2 factors into account: the number of clauses, and the number of terms. Each factor is amplified by the δ_0, δ_1 hyperparameters.

$c = \sum_{k \in K} \delta_0 + \sum_{i \in k}^{k} \delta_1$

[11]:
```
import aix360.algorithms.rbm
from aix360.algorithms.rbm import FeatureBinarizer
from aix360.algorithms.rbm import BooleanRuleCG
```

Analysis of the Framework

Overview of the API Below a quick glance of the algorithm's interface, and a brief overview of its process will be presented. Details of its inner structure both on algorithmic and implementation level will be explained in a later chapter.

The Boolean Decision Rules Generator Framework (BRCG) expects its input in a form of binarized labels, therefore both the input and the target variables need to be preprocessed.

Binarizing

Predictor labels

Dropping index, and salary labels as they are redundant for the current problem.

[16]:
```
categorical_features = ['gender', 'ssc_b', 'hsc_b', 'hsc_s',␣
 ↪'degree_t', 'workex', 'specialisation']
fb = FeatureBinarizer(colCateg=categorical_features, negations=True,␣
 ↪returnOrd=True)

X = data.drop(columns=["sl_no", "salary", "status"])
X_bin, X_std = fb.fit_transform(X)
```

Target variable

[16]:
```
Y = data["status"].map(lambda x: 1 if x == "Placed" else 0).astype(int)
```

Generating rules

Training the rule generator model, with small assigned δ_0, δ_1 parameters.

[17]:
```
def explain_with_BRCG(X, y, CNF=False, lambda0=1e-1, lambda1=1e-2,␣
 ↪verbose=True):
    # Instantiate BRCG with small complexity penalty
    br_model = BooleanRuleCG(lambda0, lambda1, CNF=False)
```

```
    # Train, print, and evaluate model
    br_model.fit(X, y)
    if verbose:
        print('Training accuracy:', metrics.accuracy_score(y, br_model.
↪predict(X)))
    if br_model.CNF:
        print('Predict Y=0 if ANY of the following rules are satisfied,␣
↪otherwise Y=1:')
    else:
        print('Predict Y=1 if ANY of the following rules are satisfied,␣
↪otherwise Y=0:')
    print(br_model.explain()['rules'])
    return br_model
```

Running the algorithm

[18]:
```
rules = explain_with_BRCG(X_bin, Y)
```

```
Learning DNF rule with complexity parameters lambda0=0.1, lambda1=0.01
Initial LP solved
Iteration: 1, Objective: 0.2960
Training accuracy: 0.8418604651162791
Predict Y=1 if ANY of the following rules are satisfied, otherwise Y=0:
['ssc_p > 58.00 AND hsc_p > 52.00']
```

Performance analysis In order to propose an improvement for the current algorithm, its main behavior should be thoroughly examined. Generally, the AIX BRCG package is an AI explainability tool, therefore its main objective is to provide relatively high accuracy in a low complexity model. It can also be claimed that the algorithm performs poorly with higher complexity parameters. Below, the earlier proposed evaluation metrics are showcased on this algorithm with the given dataset.

Examination of accuracy as a function of complexity:

Counting the number of conjunctions and terms in the ruleset

[19]:
```
def rule_set_transform(rules):
    conjunction = 0
    literal = 0
    for rule in rules:
        conjunction += 1
        literal_list = rule.split("AND")
        for l in literal_list:
            literal += 1

    return conjunction, literal
```

Calculating the complexity value, accoding to the earlier proposed formula

[20]:
```
def calc_dnf_complexity(rules, lambda0=1, lambda1=1):
    if len(rules) == 0:
        return 0
```

```
    if type(rules[0][0]) is str:
        conj, lit = rule_set_transform(rules)

    complexity = conj * lambda0 + lit * lambda1
    return complexity
```

Extracting the confusion features from the model

[21]:
```
def calc_conf_props(y_, y_pred_):

    conf_matrix = confusion_matrix(y_, y_pred_).flatten()
    tpr = conf_matrix[3]
    fpr = conf_matrix[1]
    tnr = conf_matrix[0]
    fnr = conf_matrix[2]

    return tpr, fpr, tnr, fnr
```

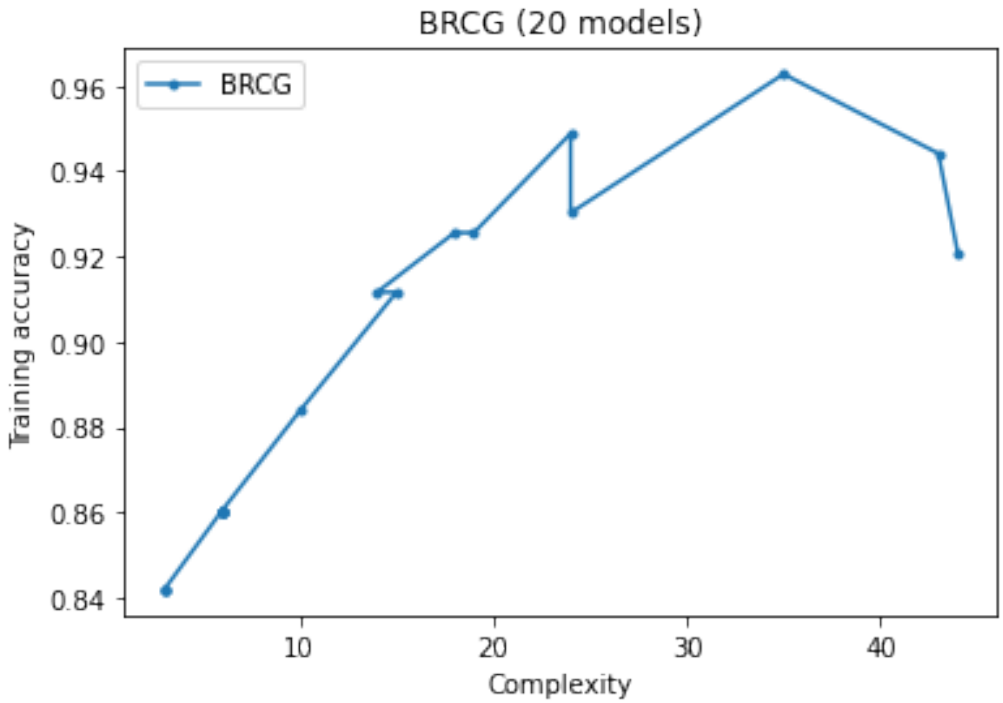
Generating 20 models with gradually increasing lambda parameters, and evaluating each resulted model

[ ]:
```
from sklearn.metrics import confusion_matrix

np.random.seed(777)
brcg_metrics = {'accuracy': [], 'complexity': [], 'dnf': [], 'tp': [],␣
↪'fp': [], 'tn': [], 'fn': [], 'lambda': []}
number_of_models_to_train = 20
lambda_values = np.linspace(0.000001, 0.01, number_of_models_to_train)

for i, l in enumerate(np.flip(lambda_values)):
    print("iter:", i)

    brcg = explain_with_BRCG(X_bin, Y, lambda0=l, lambda1=l,␣
↪verbose=False)
    y_pred = brcg.predict(X_bin)
    rules = brcg.explain()['rules']
    complexity = calc_dnf_complexity(rules)
    accuracy = metrics.accuracy_score(Y, y_pred)
    true_pos, false_pos, true_neg, false_neg = calc_conf_props(Y,␣
↪y_pred)

    brcg_metrics['accuracy'].append(accuracy)
    brcg_metrics['complexity'].append(complexity)
    brcg_metrics['dnf'].append(rules)
    brcg_metrics['tp'].append(true_pos)
    brcg_metrics['fp'].append(false_pos)
    brcg_metrics['tn'].append(true_neg)
    brcg_metrics['fn'].append(false_neg)
    brcg_metrics['lambda'].append(f"l0, l1 = {l}")
```

Visualizing the Accuracy - Complexity diagram:

[23]:
```
import os

if not os.path.exists("plts"):
    os.mkdir("plts")
```

[26]:
```
import plotly.graph_objects as go

plt.plot(np.array(brcg_metrics['complexity']),np.
 ↪array(brcg_metrics['accuracy']), marker='.', label='BRCG')
plt.xlabel('Complexity')
plt.ylabel('Training accuracy')
plt.title('BRCG (20 models)')
plt.legend()
plt.savefig('plts/comp-acc.png')
```



Due to the high imbalance in the dataset, the algorithm starts with a pretty high accuracy even with the lowest complexity, and it peaks at 96% accuracy, which later declines as the complexity increases.

Plotting the ROC curve

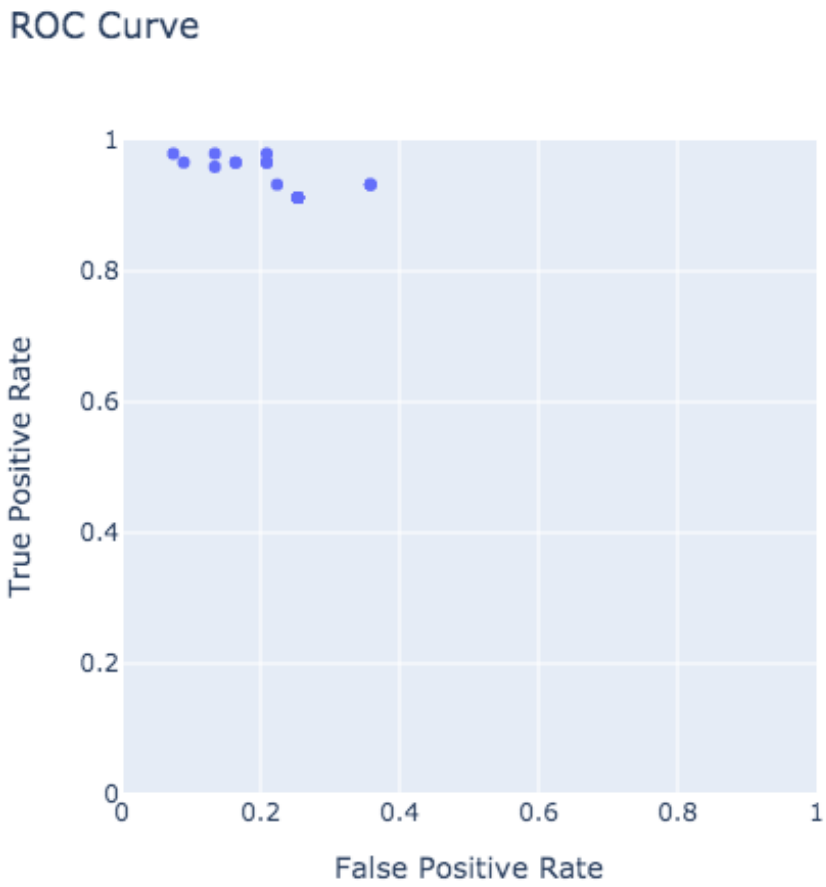
[29]:
```
n_samples = Y.count()
pos_samples = Y.sum()
neg_samples = n_samples - pos_samples

tpr = [(TP / pos_samples) for TP in brcg_metrics['tp']]
```

```
fpr = [(FP / neg_samples) for FP in brcg_metrics['fp']]
precision = [(brcg_metrics['tp'][i] / (brcg_metrics['tp'][i] +␣
↪brcg_metrics['fp'][i])) for i in range(len(brcg_metrics['tp']))]
```

[33]:
```
import plotly.graph_objects as go


fig = go.Figure(data=go.Scatter(x=fpr, y=tpr))
fig.update_traces(hovertext=l, selector=dict(type='scatter'),␣
↪mode='markers')
fig.update_layout(title="ROC Curve",
                  xaxis_title="False Positive Rate",
                  yaxis_title="True Positive Rate",
                  width=500,
                  height=500)
fig.update_xaxes(range=[0, 1])
fig.update_yaxes(range=[0, 1])
fig.write_image("plts/roc.png")
```

[34]:
```
Image(filename='plts/roc.png')
```

[34]:

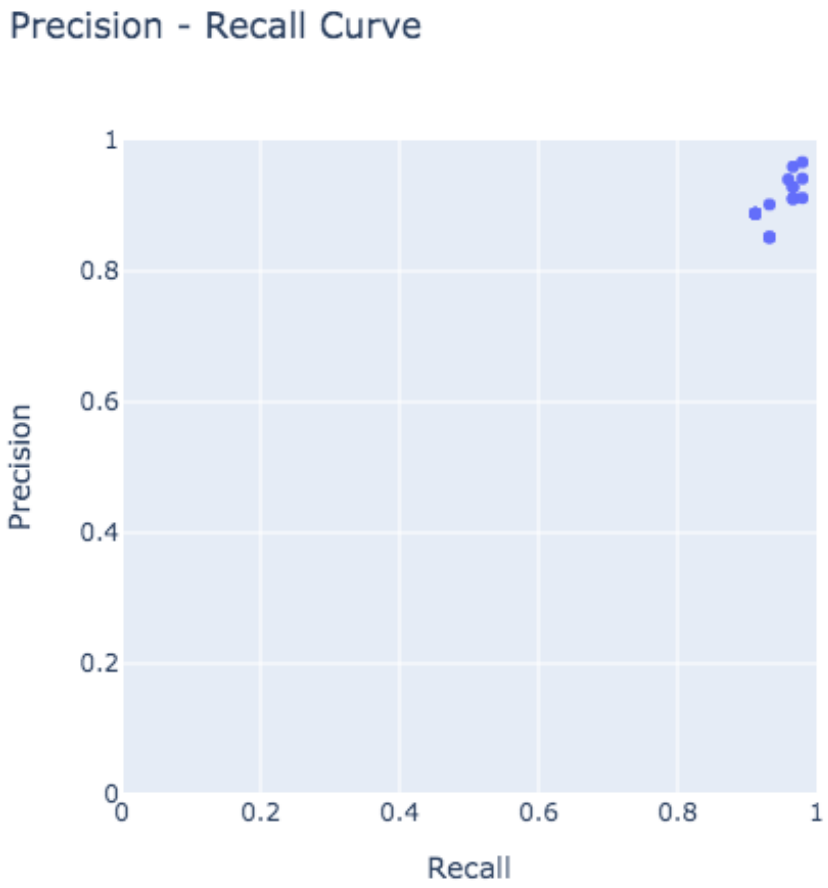


Plotting the Precision - Recall curve

[35]:
```
fig = go.Figure(data=go.Scatter(x=tpr, y=precision))
fig.update_traces(hovertext=l, selector=dict(type='scatter'),␣
↪mode='markers')
fig.update_layout(title="Precision - Recall Curve",
                  xaxis_title="Recall",
                  yaxis_title="Precision",
                  width=500,
                  height=500)
fig.update_xaxes(range=[0, 1])
fig.update_yaxes(range=[0, 1])
fig.write_image("plts/prec-rec.png")
```

[36]:
```
Image(filename='plts/prec-rec.png')
```

[36]:



Semantical analysis As it was previously mentioned, one of the major strengths of rule-based models, is that they can enclose complex mathematical connections into logic functions. This provides an opportunity to analyze the resulting models with tools that support formal logical reasoning.

Binary Decision Diagrams (BDD)

While there could be many different decision trees built up from a given ruleset, BDDs remain unique as long as the order of the variables in the rules are fixed. Since BDDs are the canonical representation of boolean functions. BDDs are represented as a rooted, directed, acyclic graph, which consists of several nodes (here: terms) and two terminal nodes, which indicates the result of the binary classification label (originally: the truth value of the boolean function).

In practical and popular usage an optimized version of BDDS are used, called: Reduced Ordered Binary Decision Diagram (ROBDD), which cuts the initial graph and leaves out redundancies.

Transformation of the terms into logic variables

Currently terms are a data set dependent representation of the engineering model. To visualize the in the above mentioned representations additional knowledge (like threshold value, operator type) is not required to be present at the first glance. The most important factor is the encapsulation of its truth value (whether the term are satisfied or not), while maintaining the orderability property of the set of terms.

Assuming that given a feature x, which set of interpretation ranges between 0 and 10. Many constraints could be defined for x over this set, take `x <= 3`, `x > 2` and `x <= 1` as an example. Since the set of interpretations can be sorted, an order over these terms can be also defined. A term $a > b$ if $d(b) \subseteq d(a)$ $d(x)$ is a funtion which returns e collection of samples which satisfies the term x, and $e \in D$ which is the initial dataset.

A problem arises when to terms which uses different comparing operators are need to be compared since for instance

$d(x <= 1) \cap d(x > 2) = \emptyset$

and

$d(x <= 3) \cap d(x > 2) \neq \emptyset$ but $(d(x <= 3) \setminus d(x > 2) \neq \emptyset) \cap (d(x > 2) \setminus d(x <= 3) \neq \emptyset)$

Therefore these terms cannot be compared by this definition. Let us describe the motivation behind the definition of this ordering. The implication ($\implies$) operator needs to be introduced in order to optimize the boolean functions. Because if a set of samples satisfies the term `x <= 1` it should also satisfy `x <= 3`. There are 3 cases: - term $a > b : a \implies b$ - term $d(a) \cap d(b) = \emptyset : a \iff \neg b$ - term $d(a) \cap d(b) \neq \emptyset$: we cannot optimize other terms must split the samples

Also when using categorical values in terms we can take advantage of that optimization option (with OneHot Encoding), that:

term: $c = c_1 \implies c \neq c_i \in C : i \neq 1$ where $C = c_1, c_2, ..., c_n$

The below presented transformation algorithm takes these cases into account.

Transforms a term with an ordinal value to a boolen expression. (Also saves the original term)

[5]:
```python
import string

class LogicalOrdinalValue:
    class Term:
        def __init__(self, expr):
            parts = expr.split()
            if len(parts) != 3:
                raise ValueError

            self.value = float(parts[2])
            self.op = parts[1]
            self.attr = parts[0]

        def get_expr(self):
            return f"{self.attr} {self.op} {self.value}"

    def __init__(self, expr, landmarks):
        print(expr.split())
        if expr.split()[0] not in landmarks.keys():
            self.term = expr
            self.type = "categorical"
            self.logical_v = None
            self.le = self.name_cat_value(expr)
        else:
            self.term = self.Term(expr)
            self.type = "ordinal"
            self.logical_v = self.v(self.term, landmarks[self.term.
↪attr]).astype(int)
            self.le = f"{self.term.attr}_{self.v2bin(self.logical_v)}"

    def v(self, term, landmark):
        value = term.value
        op = term.op
        result = np.zeros(19)
        for i, thr in enumerate(landmark):
            if op == ">":
                if value < landmark[i]:
                    result[i] = 1
            elif op == "<=":
                if value >= landmark[i]:
                    result[i] = 1
            else:  # =
                if value < landmark[i] and i != 0:
                    result[i - 1] = 1
                    break

        return result

    def v2bin(self, arr):
        return np.sum((2 ** np.arange(arr.shape[0])) * arr).astype(int)
```

```
    def name_cat_value(self, expr):
        # naming: attr_c_[in]v
        parts = expr.split()
        op = "i" if len(parts) > 1 and (parts[1] == "==" or parts[1] ==␣
↪"not") else "n"
        if len(parts) == 3:
            c = parts[2].translate(str.maketrans('', '', string.
↪punctuation))
        else:
            c = "0"
        return f"{parts[0]}_c_{op}{c}"
```

Transforms the whole ruleset into a set of boolean expressions, while applying the above mentioned optimisation techniques.

[6]:
```
from pyeda.inter import *

class TransformLogical:
    def __init__(self, ruleset, landmark):
        self.landmark = landmark
        self.transformed_rules = []
        for rule in ruleset:
            terms = rule.split(" AND ")
            self.transformed_rules.append([LogicalOrdinalValue(term,␣
↪landmark) for term in terms])

    def get_expr(self):
        result = ""
        for rule in self.transformed_rules:
            for terms in rule:
                result += terms.le
                result += " & "
            result = result.rstrip(" & ")
            result += " | "
        return result.rstrip(" | ")

    def add_ord_impl(self, x, y, const_list):
        dim = len(x.logical_v)
        x_and_y = np.bitwise_and(x.logical_v, y.logical_v)
        mask = np.zeros(dim).astype(int)
        mask[0] = 1
        mask[-1] = 1
        if np.count_nonzero(x_and_y) == 0:
            const_list.append(expr(f"{x.le} <=> ~ {y.le}"))
        elif np.count_nonzero(np.bitwise_and(x_and_y, mask)) != 0:
            if x.v2bin(x.logical_v) < y.v2bin(y.logical_v):
                const_list.append(expr(f"{x.le} => {y.le}"))

    def make_constraints(self):
```

```
        cat_constraints = dict()
        ord_constraints = []
        flat_rules = list(np.concatenate(self.transformed_rules).flat)
        # could be n*logn
        for x in flat_rules:
            if x.type == "categorical":
                attr = x.term.split()[0]
                op = x.le.split('_')[-1].startswith("i")
                if op:
                    if attr not in cat_constraints:
                        cat_constraints[attr] = list()
                    cat_constraints[attr].append(x.le)
            else:
                for y in flat_rules:
                    if y.type == "ordinal" and x.term.attr == y.term.
↪attr and x != y:
                        self.add_ord_impl(x, y, ord_constraints)

        onehot_list = []
        for onehot in cat_constraints.values():
            if len(onehot) > 1:
                onehot_list.append(OneHot0(*onehot))

        onehot_expr = And(*onehot_list)
        impl_expr = And(*ord_constraints)
        return And(onehot_expr, impl_expr)

    def get_expr_w_constraints(self):
        main_expr = expr(self.get_expr())
        constraints = self.make_constraints()
        return And(main_expr, constraints)
```

Presenting a ruleset in a form of BDD:

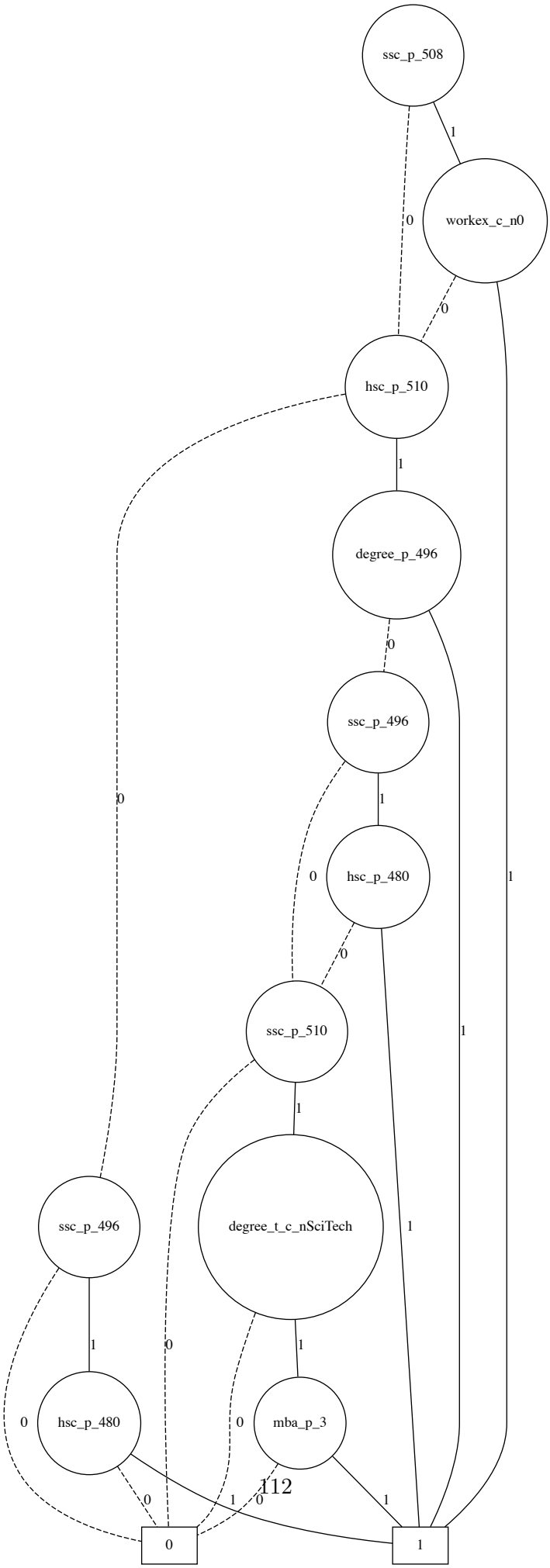
[17]:
```
c1 = TransformLogical(dnf, fb.thresh)
```

```
['ssc_p', '>', '58.00']
['workex']
['hsc_p', '>', '52.00']
['degree_p', '>', '65.00']
['ssc_p', '>', '64.00']
['hsc_p', '>', '65.00']
['ssc_p', '>', '52.00']
['hsc_p', '>', '52.00']
['degree_t', '!=', 'Sci&Tech']
['mba_p', '<=', '58.78']
```

[19]:
```
import gvmagic
%load_ext gvmagic
```

```
bdd1 = expr2bdd(c1.get_expr_w_constraints())
%dotobj bdd1
```

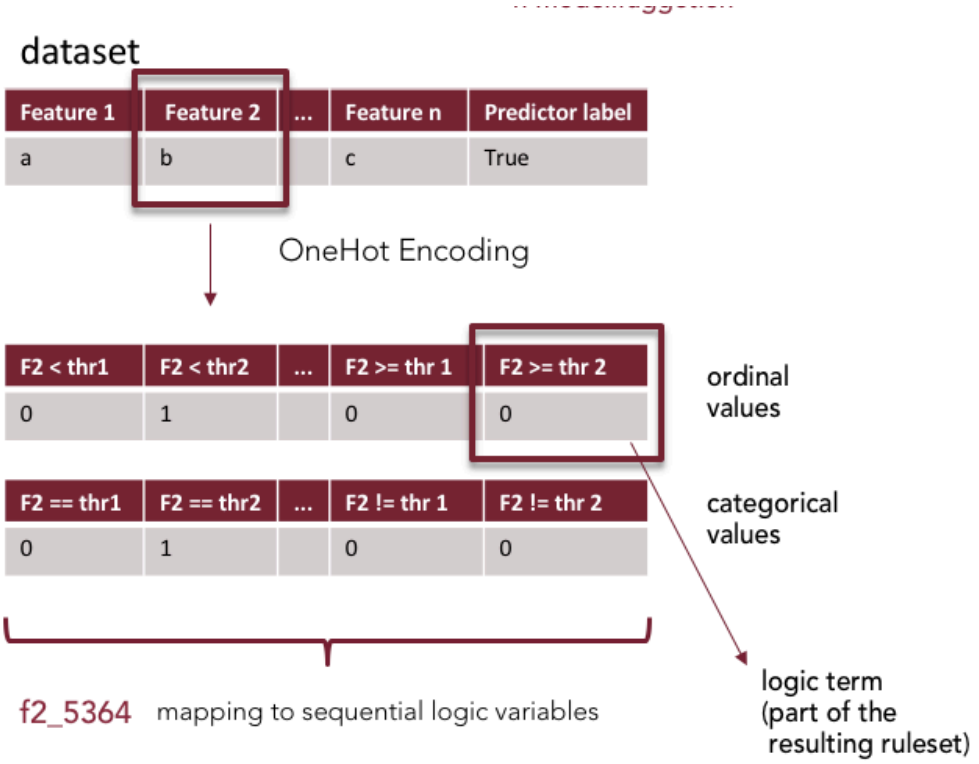
```
The gvmagic extension is already loaded. To reload it, use:
  %reload_ext gvmagic
```


ssc_p_508
workex_c_n0
hsc_p_510
degree_p_496
ssc_p_496
hsc_p_480
ssc_p_510
ssc_p_496
degree_t_c_nSciTech
hsc_p_480
mba_p_3
0
1

Preprocessing the data

[21]:
```
Image(filename='project-img/preprocessing.png')
```

[21]:



A.3.2 Implementing the Nested Cavities algorithm with a Global Solver

Here I present my additions to the original codebase and the major parts of the algorithm, however the whole code will be imported, as it contains many unrelated but neccesary snippets as well.

The algorithm uses the CBS Integer Program solver by Coin-Or.

Modified AIX code snippet:

[23]:
```
    def fit(self, X, y):
        if not self.silent:
            print('Learning {} rule with complexity parameters␣
↪lambda0={}, lambda1={}'\
                  .format('CNF' if self.CNF else 'DNF', self.lambda0,␣
↪self.lambda1))
        if self.CNF:
            # Flip labels for CNF
            y = 1 - y
```

```
        # Positive (y = 1) and negative (y = 0) samples
        P = np.where(y > 0.5)[0]
        Z = np.where(y < 0.5)[0]
        nP = len(P)
        n = len(y)

        # Initialize with all possible conjunctions
        # Feature indicator and conjunction matrices
        z = pd.DataFrame(np.array(list(product([0, 1], repeat=X.
↪shape[1]))).T, index=X.columns)
        A = 1 - (np.dot(1 - X, z) > 0)

        # Iteration counter
        self.it = 0

        # Formulate master LP
        # Variables
        w = cvx.Variable(A.shape[1], integer=True)
        xi = cvx.Variable(nP, integer=True)

        # Cost function, to bound to model's complexity
        cs = self.lambda0 + self.lambda1 * z.sum().values
        cs[0] = 0
        # Objective function
        # Here are the weighted sums, to identify the confusion metrics
        obj = cvx.Minimize(cvx.sum(xi * self.omegaFN) / n + cvx.
↪sum(A[Z,:] * w * self.omegaFP) / n + cs * w)
        # Constraints
        constraints = [xi + A[P,:] * w >= 1, xi >= 0, xi <= 1, w >= 0,␣
↪w <= 1]

        # Solve problem
        prob = cvx.Problem(obj, constraints)
        prob.solve(solver=self.solver, verbose=self.verbose)
        if not self.silent:
            print('Initial LP solved')

        # Extract variables
        sum1 = np.sum(xi.value * self.omegaFN) / n
        sum2 = np.sum(A[Z, :] * w.value * self.omegaFP) / n
        sum3 = np.sum(cs * w.value)

        # Save generated conjunctions and LP solution
        self.z = z
        self.wLP = w.value
        print(prob.value)
        print(self.wLP)

        self.w = self.wLP
```

```
        print(f"self.w \n {self.w} \n")
        if len(self.w) == 0:
            self.w = np.zeros_like(self.wLP, dtype=int)

        return sum1, sum2, sum3
```

Nested cavities algorithm:

[25]:
```
class NestedCavities:
    def __init__(self, dataset, cat_list, drop_list, target_n,␣
↪target_v, FPcost, FNcost, complexity, complexity_rate,
                 max_iter):
        self.df = dataset
        self.categories = cat_list
        self.to_drop = drop_list
        self.target_n = target_n
        self.target_v = target_v
        self.c_fp = FPcost
        self.c_fn = FNcost
        self.c = complexity
        self.c_rate = complexity_rate
        self.max_iter = max_iter

    def process(self, data, cat_list, drop_list, target_n, target_v,␣
↪pos):
        """
        FeatureBinarizer throws an exception if the dataset which it␣
↪needs to binarize consist of only 2 or less rows.
        The workaround is to inject a dummy sample to the dataset, then␣
↪remove it after binarization.
        """
        trunkAtEnd = 0
        if len(data) < 3:
            dummy_dims = 3 - len(data)
            trunkAtEnd = dummy_dims
            df_padded = pd.DataFrame(columns=data.columns)
            for i in range(dummy_dims):
                df_padded.loc[i] = np.zeros(len(data.columns))
            data = pd.concat([df_padded, data])

        self.fb = FeatureBinarizer(colCateg=cat_list, numThresh=2,␣
↪negations=True, returnOrd=True)
        X = data.drop(columns=drop_list)
        X_bin, X_std = self.fb.fit_transform(X)

        if pos:
            Y = data[target_n].map(lambda x: 1 if x == target_v else 0).
↪astype(int)
        else:
```

```
            Y = data[target_n].map(lambda x: 1 if x != target_v else 0).
↪astype(int)

        if trunkAtEnd > 0:
            return X_bin[trunkAtEnd:], Y[trunkAtEnd:]

        return X_bin, Y

    def fit_cavity(self, bin_v, target_v, complexity, FPcost, FNcost):

        br_model = BooleanRuleCG(complexity, complexity,␣
↪omegaFP=FPcost, omegaFN=FNcost, CNF=False)
        # Train, print, and evaluate model
        sum1, sum2, sum3 = br_model.fit(bin_v, target_v)
        with open("sum_logger.txt", "a+") as sum_logger:
            sum_logger.write("New iteration \n")
            sum_logger.write(str(sum1) + ', ' + str(sum2) + ', ' +␣
↪str(sum3) + '\n')

        return br_model

    def filter_data(self, dataset, bin_v, rules, pos):

        target = 1 if pos else 0
        filtered_df = pd.DataFrame(columns=dataset.columns)

        for i, sample in bin_v.iterrows():
            if rules.predict(sample) == target:
                filtered_df.loc[i] = dataset.loc[i]

        return filtered_df

    def subtract_rules(self, ruleset, neg_rules):
        sub_rules = TransformLogical(neg_rules, self.fb.thresh)
        if ruleset is None:
            return sub_rules.get_expr()
        else:
            return Xor(ruleset, sub_rules.get_expr(), simplify=False)

    def good_enough(self, prev_set, current_set, n_iter, end):
        if n_iter == self.max_iter or current_set.equals(prev_set) or␣
↪end:
            return True
        else:
            return False

    def evaluate(self, filtered_data, stats_holder, dnf_rules):
        # processing the Y values [dataset - predicted]
        Y_pred = []
        for i, row in self.df.iterrows():
```

```
            if i in filtered_data.index:
                Y_pred.append(1)
            else:
                Y_pred.append(0)

        Y_org = self.df[self.target_n].map(lambda x: 1 if x == self.
↪target_v else 0).astype(int)

        # evaluating confusion values
        conf_matrix = confusion_matrix(Y_org, Y_pred)
        conf = []
        for i, count in enumerate(conf_matrix.flatten()):
            conf.append(count)
        stats_holder["confusion"].append(conf)

        # evaluating accuracy
        stats_holder["accuracy"].append(accuracy_score(Y_org, Y_pred))

        # evaluating complexity
        c = 0
        for phase in dnf_rules:
            for conj in phase:
                c += 1
                terms = conj.split(" AND ")
                for term in terms:
                    c += 1
        stats_holder["complexity"].append(c)

        return stats_holder

    def learn(self):
        dnf_ruleset = []
        tr_ruleset = None
        model_stats = dict()
        model_stats["accuracy"] = []
        model_stats["complexity"] = []
        model_stats["confusion"] = []
        prev_samples = pd.DataFrame()
        pos_class_samples = pd.DataFrame(columns=self.df.columns)
        dataset = self.df
        n_iter = 1
        end = False
        while not self.good_enough(prev_samples, pos_class_samples,␣
↪n_iter, end):
            # getting all the positives + some false positives
            tr_X, tr_Y = self.process(dataset, self.categories, self.
↪to_drop, self.target_n, self.target_v, True)

            print(f"Phase {n_iter} / I.")
```

```
            rules = self.fit_cavity(tr_X, tr_Y, complexity=self.c,␣
↪FNcost=self.c_fn, FPcost=self.c_fp)
            filtered_data = self.filter_data(dataset, tr_X, rules, True)
            removed_data1 = self.filter_data(dataset, tr_X, rules,␣
↪False)

            dnf_ruleset.append(rules.explain()['rules'])
            tr_ruleset = self.subtract_rules(tr_ruleset, rules.
↪explain()['rules'])
            self.evaluate(pd.concat([pos_class_samples,␣
↪filtered_data]), model_stats, dnf_ruleset)

            if filtered_data.groupby(by=self.target_n).size()[self.
↪target_v] == len(filtered_data):
                print("Filtered out all the negative values")
                end = True

            self.c *= 0.8
            if not end:
                # removing all the negatives + some false negatives
                tr_X, tr_Y = self.process(filtered_data, self.
↪categories, self.to_drop, self.target_n,
                                          self.target_v, False)

                print(f"Phase {n_iter} / II.")

                rules = self.fit_cavity(tr_X, tr_Y, complexity=self.c,␣
↪FNcost=self.c_fn, FPcost=self.c_fp)
                filtered_data2 = self.filter_data(filtered_data, tr_X,␣
↪rules, False)
                removed_data2 = self.filter_data(filtered_data, tr_X,␣
↪rules, True)

                dnf_ruleset.append(rules.explain()['rules'])
                tr_ruleset = self.subtract_rules(tr_ruleset, rules.
↪explain()['rules'])

                # end of the iteration
                prev_samples = pos_class_samples
                pos_class_samples = pd.concat([pos_class_samples,␣
↪filtered_data2])
                self.c *= 0.5
                dataset = removed_data2
                self.evaluate(pos_class_samples, model_stats,␣
↪dnf_ruleset)
                n_iter += 1

                if self.target_v not in removed_data2[self.target_n].
↪unique():
```

```
                    print("Filtered out all the negative values,␣
↪finishing.")
                    end = True

            else:
                pos_class_samples = pd.concat([pos_class_samples,␣
↪filtered_data])

        return pos_class_samples, dnf_ruleset, tr_ruleset, model_stats
```

For demonstration I used the Iris dataset (https://archive.ics.uci.edu/ml/datasets/iris), from which I have removed a class to shrink the problem into a binary classification. Also FeatureBinarizer was set to divide the set of interpretability to 3 parts in order to keep the dimensions narrow, and the global solver manageable.

[2]:
```
data = pd.read_csv("../datasets/Iris.csv")

data = data.drop(data[data["Species"] == "Iris-setosa"].index)
```

[3]:
```
from boolean_rule_cg_global import *
from nestedCavities_iris import *
```

[ ]:
```
cat_list = []
drop_list = ["Id", "Species"]
target_n = "Species"
target_v = "Iris-versicolor"
FPcost = 0.5
FNcost = 2
complexity = 0.01
complexity_rate = 0.1
max_iter = 10
with open("sum_logger.txt", "w") as sum_logger:
    sum_logger.write("NESTED CAVITIES RESULTS: \n")
nc = NestedCavities(data, cat_list, drop_list, target_n, target_v,␣
 ↪FPcost, FNcost, complexity, complexity_rate,
                    max_iter)

p_samples, dnf, logic, stats = nc.learn()
print(stats)
print(dnf)
print(logic)
print(f"Accuracy: {p_samples.groupby(by=target_n).size()[target_v] /␣
 ↪data.groupby(by=target_n).size()[target_v]}")

with open("sum_logger.txt", "a+") as sum_logger:
    sum_logger.write("\n\nAIX360 results: \n")
aix_stats = compare_aix(data, cat_list, drop_list, target_n, target_v)

# plotting the confusion values
import plotly.graph_objects as go
```

```
nPos = data[data[target_n] == target_v].count()[0]
nNeg = data[data[target_n] != target_v].count()[0]
conf_stats = np.array(stats["confusion"]) / np.array([nPos, nNeg, nPos,␣
 ↪nNeg])
fig_conf = go.Figure()
param_name = ["TPR", "FPR", "FNR", "TNR"]

for i, _ in enumerate(conf_stats):
    fig_conf.add_trace(go.Scatter(y=conf_stats[:, i],
                                  mode='lines+markers',
                                  name=param_name[i]))
fig_conf.write_image("confusion.png")

fig_acc_compl = go.Figure()
acc_stats = np.array(stats["accuracy"])
comp_stats = np.array(stats["complexity"])
fig_acc_compl.add_trace(go.Scatter(x=comp_stats, y=acc_stats,
                                   mode='lines+markers',
                                   name="Nested Cavities"))
fig_acc_compl.add_trace(go.Scatter(x=aix_stats["complexity"],␣
 ↪y=aix_stats["accuracy"],
                                   mode='lines+markers',
                                   name="AIX360"))
fig_acc_compl.write_image("comp_acc.png")
```

A.4 Identifying operational domains

The notebook consisting the entire analysis is attached to this document as a separate file (`operational-domain-analysis.pdf`), due to the magnitude of the source code.