1 2 9 0

UNIVERSIDADE Đ
COIMBRA

João Pedro Damas Martins

# VERIFIABLE ARTIFICIAL INTELLIGENCE
## A CASE STUDY IN EMERGENCY HOSPITAL PATIENTS RISK ASSESSMENT

**Dissertation in the context of the Master in Informatics Engineering, Specialization in Intelligent Systems supervised by Prof. Raul Barbosa and co-supervised by Prof. Nuno Lourenço and presented to the Faculty of Sciences and Technology / Department of Informatics Engineering.**

June 2020

## Abstract

Intelligent software systems are increasingly being used in critical domains like the medical health care. Artificial Intelligence in general, and Machine Learning in particular, pose new challenges to Verification, a crucial step of the critical systems development process. Formal Methods, such as Model Checking, are well known techniques that allow for proving properties in critical systems. Current work assesses the usage of Model Checking to perform verification in an emergency hospital patients risk assessment use case. The proposed approach is a framework that contemplates verification steps during both design and run time. In concrete, at design-time, it is able to check a model for invalid end states, non-determinism and accordance with *a priori* knowledge. Online verification focus on verifying the confidence of a classification (forecasting) for a specific instance, based on a tailored distance measure that checks the closeness to the model decision boundaries. This last phase of verification is also considered as ensemble strategy in a scenario of combining more than one classifier. Experimentation was done on three available risk assessment models (the Risk Scores GRACE, PURSUIT and TIMI) with real data of 460 hospital patients. Verification at design-time for the three models (a) confirmed the inexistence of invalid end states for the whole operation input space nor (b) non-determinism for the available test set, and (c) provided confirmation of compliance with *a priori* knowledge statements. Online verification (performed for GRACE) successfully divided the available instances (patients) into two groups, *Confident* and *Not-confident* about the risk assessment, where (a) the performance in comparison to the baseline improved for the *Confident* group and degraded for the *Not-confident* one, and (b) the execution statistics of the model checker proved its efficiency to perform verification at run time. The ensemble strategy was evaluated in two scenarios that considered different overall usage ratios for the verified GRACE model (based on the online verification parametrisation), along with several complementary classifiers. PURSUIT, one of the domain dependent Risk Scores, and a trained Decision Tree Classifier provided the best match to complement GRACE in the classification of instances not-confidently assessed. Based on the results, the proposed framework succeeds in using Model Checking for verification to increase trust on intelligent systems decisions, made in critical domains.

## Keywords

critical systems, intelligent systems, verifiable artificial intelligence, trustworthy artificial intelligence, verification, formal methods, model checking, risk assessment tools, medical risk scores

This page is intentionally left blank.

## Resumo

Sistemas de software inteligentes são cada vez mais usados em domínios críticos como o setor da saúde. A Inteligência Artificial em geral e Aprendizagem Máquina em particular, colocam novos desafios à Verificação, um passo crucial no processo de desenvolvimento de software crítico. Métodos Formais, como *Model Checking*, são técnicas bem conhecidas que permitem provar propriedades de sistemas críticos. O presente trabalho avalia a utilização de *Model Checking* para realizar verificação num caso de estudo de avaliação de risco em pacientes de emergência hospitalar. A abordagem proposta é uma estrutura que contempla verificação quer na fase de desenho, quer na fase de execução (em linha) do sistema. Em concreto, durante a fase de desenho, é capaz de verificar um modelo para a existência de estados finais inválidos, não-determinismo e a conformidade com conhecimento *a priori*. A verificação em linha foca-se em avaliar a confiança da classificação (ou previsão) para uma dada instância, baseada numa medida de distância adaptada que indica a proximidade às fronteiras de decisão do modelo. Esta última fase de verificação é ainda considerada como estratégia de *ensemble* para um cenário de combinação de mais do que um classificador. A experimentação foi realizada em três modelos de avaliação de risco disponíveis (Escalas de Risco GRACE, PURSUIT e TIMI) com dados reais de 460 pacientes hospitalares. A verificação em fase de desenho para os três modelos (a) confirmou a inexistência de estados finais inválidos, nem (b) de não-determinismo para os dados testados, (c) confirmando também concordância com as declarações de conhecimento *a priori*. A verificação em linha (realizada para o GRACE) dividiu com sucesso as instâncias disponíveis (pacientes) em dois grupos, *Confiante* e *Não Confiante* em relação à avaliação de risco, onde (a) o desempenho em relação à execução de controlo melhorou para o grupo *Confiante* e degradou para o *Não Confiante*, e (b) as estatísticas de execução do *Model Checker* provam a sua eficiência para realizar verificação em linha. A estratégia *ensemble* foi avaliada em dois cenários, considerando rácios de utilização diferentes para o modelo verificado GRACE (baseados na parametrização da verificação em linha), combinados com vários classificadores complementares. PURSUIT, um dos modelos de escalas de risco restritos ao domínio, e um Classificador de Árvore de Decisão treinado nos dados foram os que melhor complementaram o modelo GRACE na classificação de instâncias sem confiança na avaliação de risco. Com base nos resultados, a abordagem proposta tem sucesso em usar *Model Checking* na verificação para aumentar a confiança nas decisões de sistemas inteligentes, tomadas em ambientes críticos.

## Palavras-Chave

Sistemas críticos, sistemas inteligentes, inteligência artificial verificável, inteligência artificial confiável, verificação, métodos formais, ferramentas de avaliação de risco, escalas de risco médicas

This page is intentionally left blank.

## Acknowledgements

Here, I provide some kind words to those who contributed to and throughout my dissertation process and results.

I want to start by thanking *Professor* Henrique Madeira, who challenged me to address this important topic, setting high expectations for the work I ended up doing as my dissertation and within the AI4EU Project.

Of course concluding this work would have been an impossible task without the advisory of *Professor* Raul Barbosa and *Professor* Nuno Lourenço. To them I share my gratitude for guidance and continuous feedback, which I extend to *Professor* Jacques Robin, from University Paris-Sorbonne 1, also involved since the beginning in the planning and discussion of the research I developed.

I want to thank *Professor* Paulo Carvalho, *Professor* Jorge Henriques and *Professor* Simão Paredes for sharing with me the details and fruitful insights of the Use Case I studied, as well as the available data repository.

I express my regards to the assessment provided by *Professor* Vasco Pereira and *Professor* Joel Arrais, members of the Jury for this dissertation, during the intermediate defense.

A word of appreciation goes to the anonymous reviewers of the Workshop on Dependable and Secure Machine Learning (DSML) of the 50th IEEE/IFIP International Conference on Dependable Systems and Networks, who accepted my contribution and provided relevant comments.

A warm thanks goes to my friends and colleagues for the fruitful discussions and for making this research period a lot easier.

Lastly, a special tribute goes to my family for their continuous backing, motivation and caring, and to my girlfriend, Margarida Penacho, who was a constant source of emotional support (and many times also domain expertise) throughout all the steps of this dissertation. To them I owe several dinners!

This work is partially funded by Portuguese Foundation for Science and Technology, I.P., within the project CISUC - UID/CEC/00326/2020, by European Social Fund, through the Regional Operational Program Centro 2020 and by the project AI4EU – "A European AI On Demand Platform and Ecosystem", project funded by the European Commission, H2020-EU.2.1.1., Grant agreement ID: 825619.

This page is intentionally left blank.

# Contents

This page is intentionally left blank.

# Acronyms

**AI** Artificial Intelligence. 1, 2, 4, 7–9, 13–16, 28

**CVD** Cardiovascular disease. 2, 3, 9, 12, 15, 19, 31, 33, 34, 39, 46

**GRACE** Global Registry of Acute Coronary Events. 11, 20, 36, 38, 41–44

**MC** Model Checking. 3, 14, 15, 30

**ML** Machine Learning. 2, 15, 19, 34

**PURSUIT** Platelet in Unstable angina: Receptor Suppression Using Integrilin Therapy. 11, 36, 41, 44

**RS** Risk Score. 3, 9–12, 15, 18–21, 28, 31, 33–36, 38, 41–44

**TIMI** Thrombolysis In Myocardial Infarction. 11, 12, 36, 41, 44

# List of Figures

This page is intentionally left blank.

# List of Tables

This page is intentionally left blank.

# Chapter 1

# Introduction

## 1.1 Motivation

Nowadays, digital systems are present in a variety of fields, many of them understood as critical. A few examples appear in the health and transportation sectors, ranging from diagnosis tools [43] to self-driving vehicles [22]. If in previous years there were significant improvements in the software development process to address the requirements inherent to the critical characteristic of certain fields [21], the same might not apply when it comes to intelligent systems evolving their own notion of solutions and procedures [48]. This stresses the importance of revising the way intelligent systems are being built, as well as how and in what fields they are being deployed.

One could focus on the regulatory perspective and enforce best practices through legal binding processes. Unfortunately, when it comes to technology, the legislation cycle is usually inadequate to the tech disruptive environment. As an example, there are already guidelines to ensure that intelligent systems comply with ethical requirements[1], but what they represent in practice still falls to the field of interpretation [53]. Therefore, it is still pertinent to look at this challenge from an engineering point of view.

There are two main phases to assess the quality of a software system: *verification* and *validation* [39]. Put it in simple terms, *verification* ensures that the system is built right according to a set of requirements, whereas *validation* understands if it is the right system (i.e. if it solves the problem, which indicates the requirements were right). If both steps are applied correctly, the right system has been built in the right way. However, this is usually easier said than done.

*Validation* can take different methods according to the application field [20, 30]. This task focus on the outcome of the system and is usually the one most valued from a business perspective. Let us, then, turn the attention to the *verification* process. First, thought should be put on what kind of system is being verified, in order to understand the tools available and suitable to perform verification. Then, the development process itself, as mentioned above, is also in some cases part of the verification phase as well.

Critical software in general, and medical software in particular, must comply with certain standards like life cycle management (IEC 62304). Regardless, these standards were drafted upon assumptions that are normally not verified by Artificial Intelligence (AI). Examples include considering software is manually implemented by human programmers, specifying

[1]https://ec.europa.eu/digital-single-market/en/news/ethics-guidelines-trustworthy-ai

at design-time the whole control and data flows of the software, or that this implementation can be tested extensively by running test sets that verify properties of control and data flows (also manually programmed).

An AI system or component, on the other hand, is generally implemented by a persistent declarative knowledge base, that is specific to the software application domain, but it is interpreted by a generic inference completely independent of the field in question (learning algorithm) [36]. It is the run-time relation between (a) the volatile input data given to the AI to reason out, (b) its declarative persistent knowledge base and (c) its application-independent inference engine that defines control and data flow. Also, the inference engine can perform heuristic and/or non-deterministic approximate search of large combinatorial spaces, which makes extensive design-time and manual writing of tests for such AI impractical [36].

This difficulty is made worse by the fact that nowadays part if not all the knowledge base is acquired from data by Machine Learning (ML) instead of manually declared. This learning process includes steps of data selection [10] and transformation [7], choice of learning algorithms [8], parameter tuning of these algorithms [28], learning hypothesis, search space *a priori* pruning [31], all of which introduce biases that can completely change the resulting knowledge base and hence the reasoning of the AI component. All these questions are only starting to be explored by the ISO/IEC task force JTC-1/SC-42[2].

Formal methods in general and Model Checking in particular are suitable techniques to prove properties in critical systems [13]. They are mathematical based and hence model agnostic, if certain constrains like being code reproducible and deterministic are met. There are challenges in applying formal methods to AI, which have been identified along with possible research strategies to solve them [49]. These techniques have been widely adopted by the software development community, and for that reason well documented and available as off-the-shelf tools.

## 1.2 Research Questions

This dissertation aims to study ways to ensure verification and guarantee properties such as reliability and safety of ML models.

Binary classification is the simplest kind of ML task, where one has to separate a pool of instances in two different groups. Several approaches, presented in Section 3.2, exist to address that problem. Forecasting is the task of predicting whether or not a given event will occur in a future time window of a given length, and it can be treated as a binary classification problem.

Given the already acceptance and use of Formal Methods like Model Checking, present work starts from the following general research question:

**Q1.** Can Model Checking be used to verify the confidence of a critical-domain forecast for a *specific* instance made by a machine learnable model?

To narrow the research scope, a case study in emergency hospital patients risk assessment is considered, given the relevance that verification steps might have in this field for medical/ethical reasons and implications. Work builds on a Cardiovascular disease (CVD) Risk

[2]https://www.iso.org/committee/6794475.html

Assessment Tool and the challenge of verifying each of the forecasting models available to assess a patient risk. Risk assessment is a forecasting task, predicting if a certain negative medical relevant event might happen in a future time frame. This use case poses as a straight away research challenge to verify the instance-based (patient) confidence of forecasting (risk assessment).

If it is true that each of the forecasting models, in specific called Risk Score (RS) models, have already been validated to some extent [14], defining a systematic approach to verify properties about them looks like the next interesting step. Ideally, such approach should provide insights on how to either combine or select the RS models when more than one is available and suitable to the forecasting task.

Model Checking (MC) is an algorithmic method that has been initially developed for hardware designs and communication protocols inspection, but it can also perform exhaustive state space analysis of software. Such technique allows for proving specific properties, like consistency and absence of deadlocks, returning counterexamples if the software design fails to verify those properties.

From both the use case and suitable technique identified, more specific questions can be addressed:

**Q2.** Can model checking be used to verify invalid end states, non-determinism and *a priori* knowledge of RS models at design time?

**Q3.** Can model checking locally verify and explain at run time (online) the decision made by a RS for a given patient?

**Q4.** Are there advantages of considering an ensemble approach based on online verification information?

**Q5.** Is a verifiable approach competitive in terms of performance and execution efficiency?

So far, no guarantees are provided about the consistency (absence of invalid end states nor non-determinism and compliance with *a priori* knowledge) of the RS models and their decisions in the CVD Risk Assessment Tool. Up to some extent, this has been identified as a drawback to the acceptance of the tool in real life. One can look at it as a downside for two main reasons: (a) medical staff are not intuitively ready to trust on a tool without assurances and (b) in order to be incorporated in/as a certified medical device it would imply a process where questions of verification would be raised.

Assuming the possible verification scopes defined in Chapter 2, one is interested in verifying the property of dependability, on one or more of its dimensions. Opportunities to integrate the verification checks arise both at design and run time.

Considering the CVD Risk Assessment Tool as critical, one is interested in verifying its safety to avoid (or at least minimize) catastrophic failure events of the system. In specific for this tool, this would be failing on the risk assessment and cause a mistreatment of a patient. Further analysis would also include prioritising false positives over false negatives.

Verifying safety can be done through structural checks. This means looking for inconsistencies, contradictions, tautologies, non-determinism in the models or in the system.

One can assume that there are two main sources for errors: limitation of training data and limitation of the learning strategies. In the dataset there can be contradicting results for close sets of symptoms, which the system will have to adjust, making a discretionary choice. Depending on the classifier properties, one can look for n-dimensional volumes where the classification is less accurate and therefore provides less confidence to medical teams. Model checking could be used to find those volumes.

In addition to identifying these border zones, one could explain to medical staff what factors are causing the uncertainty. This can be seen as online model checking, a monitor that can explain why the decision is unclear.

A high level strategy to achieve these research objectives includes: preparing a development environment to train and evaluate the results of machine learnable models in the selected case study; designing an approach in which example-based learning algorithms can be used and combined with a verification plug-and-play component which will monitor the system (before deployment and/or during execution); planning and executing experiments to assess whether the learning outcome (with the verification component) ensures safety and reliability in new instances that are not part of the training sample set.

## 1.3 Contributions

Main contributions derived from this work are the following:

- Development of a conceptual approach for verifying machine learnable models using the formal method of model checking;
- Experimentation of the approach on a use case of emergency hospital patients risk assessment;
- Study of the state of the art on Verifiable AI that was input to the version 3.0 of the survey of Task 7.2 of AI4EU Project;
- Evaluation of instance based online verification to qualify confidence in classification tasks that resulted in an accepted scientific paper at DSML 2020;
- Deployment of a Docker container with the developed approach applied to the case study as part of Delivery 7.2 of the AI4EU Project.

## 1.4 Publications

Main findings of the work done along the online verification phase resulted in a submitted and accepted communication to the Dependable and Secure Machine Learning Workshop of the 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks. Proceedings are in printing, so a pre-print is included as appendix of this dissertation. Preliminary reference is as follows:

> **João Martins**, Raul Barbosa, Nuno Lourenço, Jacques Robin, Henrique Madeira. Online Verification through Model Checking of Medical Critical Intelligent Systems. In *DSN Proceedings, DSML@DSN 2020 - 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, Dependable and Secure Machine Learning Workshop 2020, (in printing).*

## 1.5 Document outline

Current Chapter sets the theme, the research questions of this dissertation and the main contributions. The rest of the document elaborates on the work performed and it is organised as follows: Chapter 2 describes the background context and scope of medical critical systems, including a description of the considered use case on emergency hospital risk assessment; Chapter 3 provides an overview of the state of the art in intelligent systems verification, namely on formal methods in general and model checking in particular, a comparison of common machine learning algorithms and the relevance of ensemble strategies and explanations as means to verification; Chapter 4 accounts for the considered and proposed approach to a verification framework on its three main phases, explaining verification done (a) globally and at design-time versus (b) online and instance oriented, and how this latter can be used as ensemble strategy; Chapter 5 provides detailed results and discussion of the experimentation and validation performed to the framework using real patients data; Chapter 6 wraps up the work summarising the key findings and identifying limitations and future avenues to continue this research.

This page is intentionally left blank.

# Chapter 2

# Background

## 2.1 AI4EU

This dissertation is part of the AI4EU Project. AI4EU is a consortium established in January 2019, under the H2020 programme of the European Commission, to build the first *European Artificial Intelligence On-Demand Platform and Ecosystem*[1]. The project is run by a consortium of 81 institutions (AI academic, business, industry related) and comprises several activities, including:

- creating a multi-stakeholder ecosystem across European nations for collaboration to approximate industry and academia goals;
- designing a platform hub for sharing of expertise and knowledge, datasets, computing resources and funding, essential to any research project;
- inducing industry-led research in real applications, bridging and pushing forward the development of new AI products;
- aggregating research on 5 AI scientific areas – *Explainable AI, Physical AI, Verifiable AI, Collaborative AI and Integrative AI*, – that pose different challenges but should be thought in an interconnected way;
- creating an *European Ethical Observatory* and a *Strategic Research Innovation Agenda for Europe, sustainably handing over the platforms created.*

Within the AI4EU project, present work is included under the scientific area of Verifiable-AI, Work Page 7: Filling AI Technology Gap, on Task 7.2 and already included in Delivery 7.2.

## 2.2 Verifiable-AI

Some examples of domains where Verifiable AI is relevant are autonomous vehicles and intelligent medical devices. Verification is a consequence of the need for certification and assurance of dependability and/or other requirements [39].

[1]https://www.ai4eu.eu/about-project

According to the WP 7/Task 7.2 of AI4EU, Verifiable AI can be defined as all methodologies that are able to check the main properties of an AI system and all of its components.

Verification is, as already briefly mentioned in Section 1.1, assuring the correctness of system operation according to functional and non-functional requirements. A key property to be checked is often dependability [29], that is composed by six dimensions: reliability, safety, availability, confidentiality, integrity and maintainability.

Another way of understanding the scope of Verifiable AI is through three key concepts: *failure, error* and *fault.* Hierarchically speaking, in an increasing severity scale, a *fault* can cause an *error* as a system state that in turn can lead to a service *failure.* When addressing critical systems, one wants to avoid *failure* at all costs and for that should focus on four *fault* activities: *prevention, removal, forecasting* and *tolerance.*

Figure 2.1 depicts the structured thinking put into the broad research topic of Verifiable-AI.

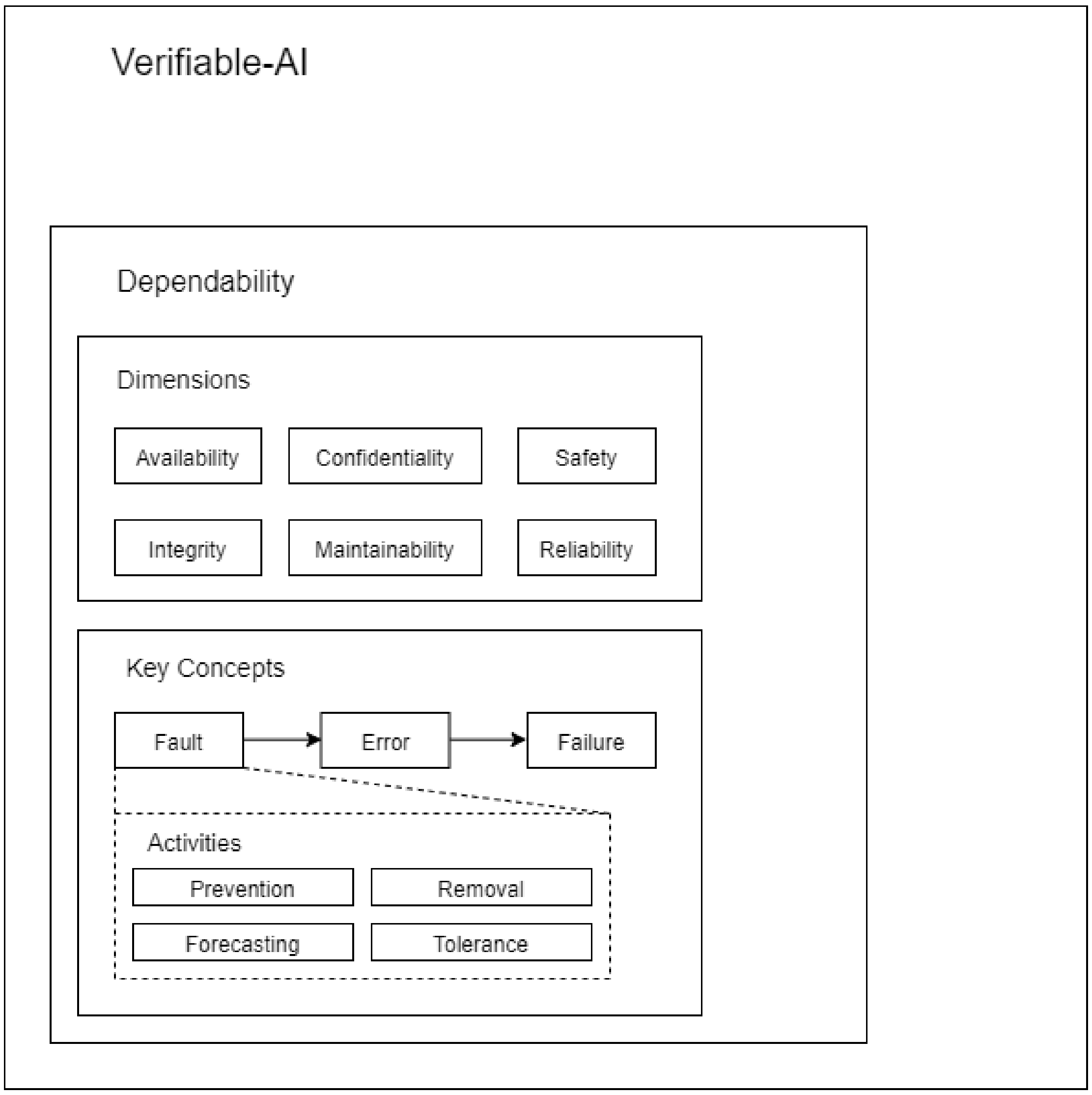


Figure 2.1: From Verifiable-AI to fault activities.

## 2.3 Case study

### 2.3.1 Scope

Intelligent medical devices are promising tools to personalise health care. It is, nonetheless, a shallow research field to certificate medical software with AI components. Although there are standards for medical software, lifecycle management (IEC 62304), risk assessment (ISO 14971) and usability (IEC 62366) are examples that are yet to fully address AI raised issues. Hospital emergencies deal with several different challenges but their mission is to provide the best health care to as many people, as fast as possible. One of the key aspects to succeed in this mission is to constantly correctly assess each patient symptoms and reach the right diagnosis, in order to provide the right treatment. Considering the long list of possible symptoms and the list of diagnoses that non-linearly correlates the set of symptoms, one can categorise hospital triage and admission as, at least, a complex task.

One particular group of patients that arrive at emergencies are patients with Cardiovascular Disease (CVD). Assuming a correct diagnosis, when it comes to treatment it usually falls into two categories, one non-invasive or pharmacological, and another one invasive, for instance surgical [6]. Choosing the right treatment relates to assessing the risk of a future cardiovascular event. In a low/intermediate risk situation one can opt indistinguishably for one of the categories, with same statistical results in decreasing the probability of a fatal event [40]. In this scenario, financial cost of treatment is an important issue. When risk is high, preferably invasive (usually more expensive) measures help decrease the chance of fatal events.

Risk Scores were introduced in medical emergencies as tools to help prioritise and differentiate patients diagnoses, providing the needed balance between saving people's lives while saving on health costs through application of the right treatment.

The RSs are based on large scale longitudinal medical research studies [15]. They are, however, easy to implement as decision-support software since they generally apply simple weighted combinations of indicators such as blood pressure or known history of a condition. The output is the sum of the weighted values which falls under a risk category.

The various limitations of RS where studied to some extent along with attempts to overcome them by the tool selected as use case for this dissertation [42]. In specific, it is relevant to be able to count on several risk scores or other assessment tools and indicators, because they consider different input factors and account for missing patient information while providing the medical staff with a well-founded risk evaluation.

Figure 2.2 depicts the concepts in a structured view.

### 2.3.2 Cardiovascular Disease Risk Assessment Tool

Two general approaches are presented by the authors of a CVD Risk Assessment Tool [35]. Both focus on the use of different Risk Scores (see Section 2.3.3), one by combining them and the other by selecting the RS which *a priori* is better suited to provide a correct risk classification.

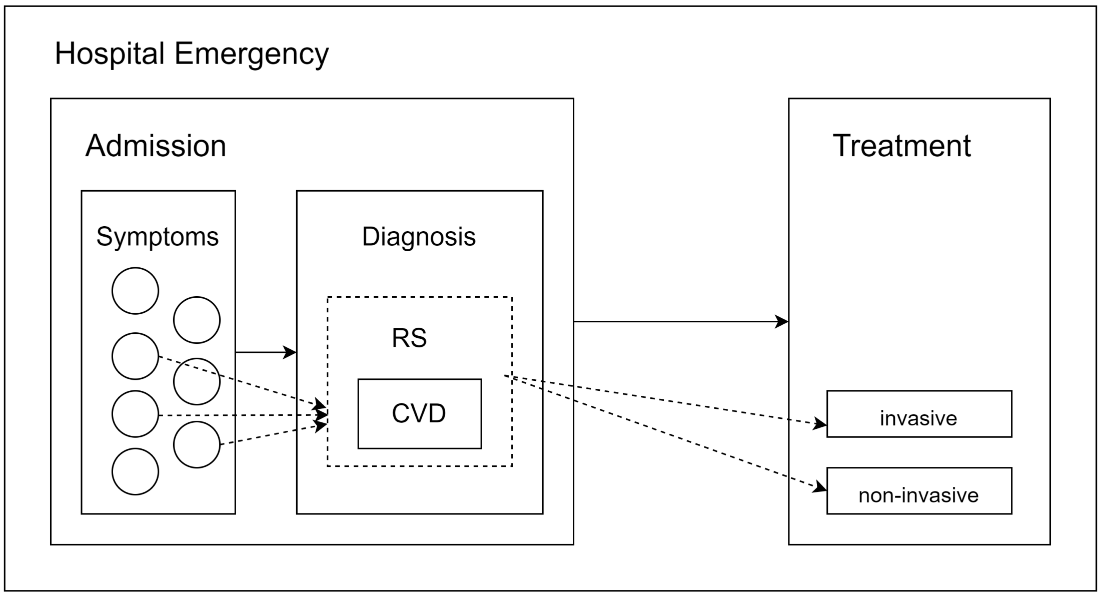


Figure 2.2: Concept diagram of the use case considered.

### Combination Approach

The main idea of this approach is to consider the outputs of every RS model and combine them in a way that can provide the user with a risk classification. From a high level perspective, first step is to transform each of the models into a common representation. An overview of advantages and disadvantages of possible choices for classification representation appears in Chapter 2. The authors of the tool opted for a naive Bayes representation. Second step is a discretisation of the input features space, based on known strategies and domain knowledge. Third step creates a model in the common representation for each RS, after which a weighted parameter combination is performed based on the expressions in [42]. Finally, a global model is obtained. Figure 2.3 provides a brief representation of the strategy.

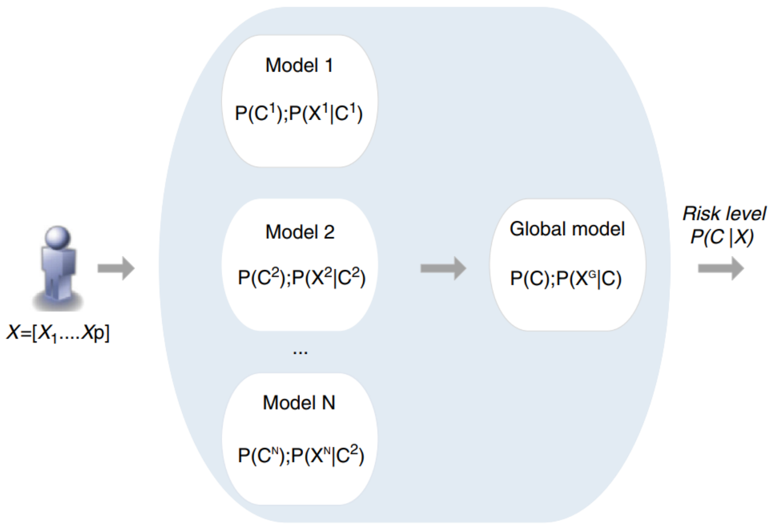


Figure 2.3: Combination approach for the CVD Risk Assessment Tool, obtained from [43]

Results and validation observed that the global model, as is, deprecates its performance in comparison with the best RS within the combined group. A further step using Genetic Algorithms performs optimisation of the parameters, bounded by strict rules in order not to loose the information learnt from the dataset. This final step provides the approach with competitive results [41].

### Selection Approach

This second approach is focused on performing a meta-selection of the best RS to be used for a specific patient risk assessment. In this subsection, only the selection method based on distance metrics is explained. Figure 2.4 provides a brief representation of the strategy.

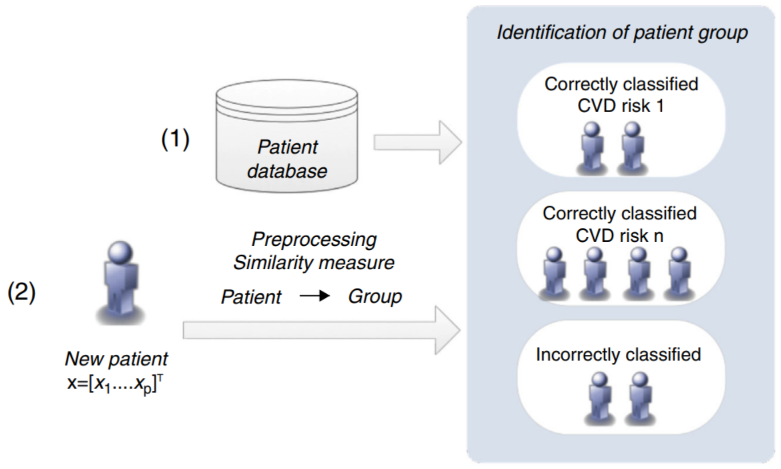


Figure 2.4: Selection approach for the CVD Risk Assessment Tool, obtained from [43]

Patients well categorised by a certain RS are seen as a *cluster*. This creates a scenario with as many groups as RS models available, plus one for the group of patients incorrectly classified by all the models. When a new patient instance is to be risk assessed, the tool calculates the similarity to each cluster, based on a chosen distance metric, and applies the corresponding RS model. For the incorrectly-classified-by-all-models group, the authors of the approach decided that it should be applied the RS model with better performance.

### 2.3.3 Risk Scores

Three Risk Scores (RS) are combined/selected in the CVD Risk Assessment Tool - Global Registry of Acute Coronary Events (GRACE), Platelet in Unstable angina: Receptor Suppression Using Integrilin Therapy (PURSUIT) and Thrombolysis In Myocardial Infarction (TIMI) [2]. The choice for these specific RSs was due to their acceptance and application in real life clinical environment [42].

Each RS operates on the basis of applying weighting factors to the considered input variables, according to their value. The output is the sum of the weighted values.

Table 2.1 characterises each of the RS models used by the CVD Risk Assessment Tool. Important information to retain is that all models were built for short term assessment, based on events of death or myocardial infarction (heart attack) on patients with coronary artery disease, considering different risk factors.

Table 2.1: Risk Score models used in CVD risk assessment, obtained from [42]

| **Model** | **Patients enrolled** | **Event** | **Term (months)** | **Patient's condition** | **Risk Factors** |
|---|---|---|---|---|---|
| GRACE | 1143 | Death/MI | 6 | CAD | Age, SBP, CAA HR, CR, STD, ECE, KIL |
| PURSUIT | 337 | Death | 1 | CAD | Age, Sex, SBP, CCS, HR, STD, ERL, HF |
| TIMI NSTEMI | 3171 | Death/MI/UR | 14 days | CAD | Age, STD, ECE, KCAD, ASP, ANG, RF |
| SBP systolic blood pressure, CR creatinine, HR heart rate, CAA cardiac arrest at admission, KIL Killip class: II-IV, STD ST segment depression, ECE elevated cardiac enzymes, KCAD known coronary artery disease, ERL enrolment (MI/UA), HF heart failure, CCS angina classification, ASP use of aspirin in the previous 7 days, ANG 2 or more angina events in past 24 h, RF 3 or more cardiac risk factors | | | | | |

As identified in the study limitations of [37], the trade off between performance and model complexity of a RS, in specific analysed for TIMI, reveals that simpler (in complexity) models have an important attribute of interpretability, but compromise on risk discrimination power. Indeed, visual inspection of the three considered RS models implementation (see Section 4.2) reveals differences in complexity, which correlate to the performance results referenced in previous research for the CVD Risk Assessment Tool [43].

It is also relevant to mention that presented RSs were drawn out of massive international databases built during clinical trials, that comprised several hospitals and institutions, and that their patients information is proprietary and therefore not available for use in current work. Even if available, since a given RS considers specific risk factors, such databases would be difficult to use for cross-testing and comparison, as only a fraction of the risk factors is common for the three RSs.

# Chapter 3

# Related Work

In this chapter, current state of research of relevant areas to this dissertation are surveyed. Outlined are the basis of formal methods and verification tools, an overview of different machine learning classifiers and the verification closeness to explication tasks.

## 3.1 Formal methods

Formal methods are a complementary approach to software engineering methodologies and development processes, being suitable for describing and reasoning about complex systems [51]. Based on formal logic and mathematical notions, for the purpose of this work one can consider two concepts as key areas or techniques of formal methods: formal proofs [50] and model checking [9].

One can look once again to the concept of verifiable and verified AI. In specific, in [49] we have *'verified AI as the goal of designing AI-based systems that have strong, ideally provable, assurances of correctness with respect to mathematically-specified requirements.'* For the authors, formal methods are plausible candidates to assist on the task, since they can provide evidence on the rightness of what is implemented, although making a note that transposing the techniques *as is* to the AI field is not the way to go.

State of the art in computational proof mechanisms are Boolean Satisfiability (SAT) Solvers [34] and boolean reasoning and manipulation techniques based on Binary Decision Diagrams (BDDs) [9].

Figure 3.1 relates key components and steps of a generic formal verification procedure. Starting by creating a Model $M$, composing representations of the System $S$ and its Environment $E$, one applies a verification tool to verify Property $P$ on $M$. Verification tool either confirms P or provides a counter example that violates P.

When it comes to AI based systems, the challenge is to create proper formalisms of the three inputs, in a way one can apply a formal decision method to provide us with the yes or no output intended, doing it so efficiently and in a reasonable time frame. Five points on specific challenges raised in [49] are:

1. *How to model the Environment E*

   Classical approach to machine learning is considering features that derive from environment characteristics. Nevertheless, one cannot model every aspect of an environment, as doing it so is often too complex or unnecessary to solve the problem.

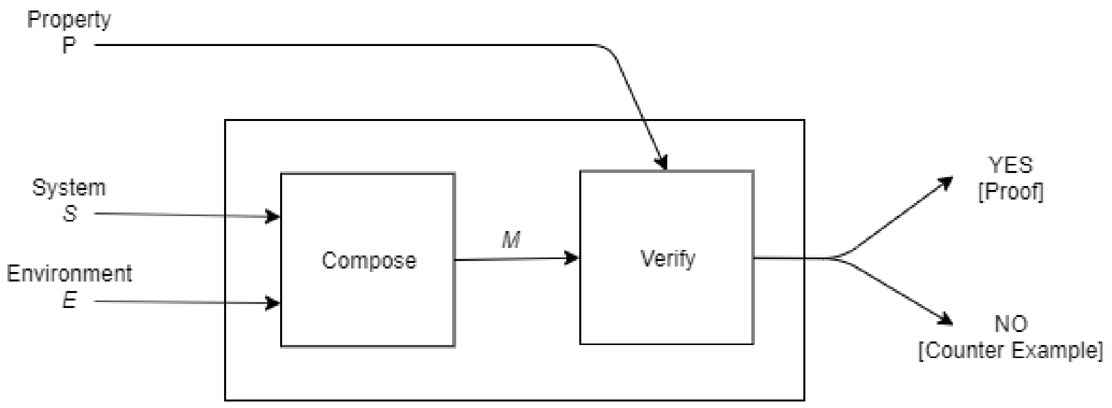


Figure 3.1: Formal verification procedure, obtained from [49]

2. *Specification of the System S*

   The challenge here is to define what to consider as system, and how to combine an AI module specification into a bigger system. A plausible example appears in autonomous driving vehicles, where different components, ones *'intelligent'* and others not, act together to perform the task of driving.

3. *How to model systems that change over time*

   Uncertainty is a major factor in machine learning, since one try to generalise an environment behaviour through a subset of known data. The challenge is to provide a verification that holds through time evolution and for unseen data.

4. *Methods for training, testing and validation*

   Rather than pointing to the hard correctness proof of formal methods application, one can consider only an intermediate level of assurance about a property. However, one must be aware that adversarial perturbations [23] can easily compromise the system with borderline situations, and that is often neglected by formal methods.

5. *New design approaches*

   Currently there is an ever increasing pressure to verify intelligent systems. This will inherently push forward the design of new approaches to machine learning that are verifiable by design. This stresses the challenge of continuing to use current classical methods for verification, while one should perhaps be rethinking the learning process.

Several of these challenges are research topics on their own. For example, there are published proposals that make the most of adversarial learning [22], and considerations to prove safety in reinforcement learning by applying formal methods in runtime monitoring [19] or via policy extraction [5].

### 3.1.1 Model checking

Model Checking (MC) is a formal method used to verify hardware and software systems [13]. The concept of model derives from a specification of what a system should do, or a formal way of requirements. MC is intended to detect any flaws in the model specification.

MC has historical relevance in critical systems development typically following a waterfall process, meaning that it was a verification step before implementation. Nowadays, with new development processes, one can see a program code as a specification itself [13]. The argument is that code is a static representation that has yet to be compiled, converted into binary and executed, so it is actually valid to see it as a requirements specification.

MC is assuring a certain specification property for a certain model. Specifications are usually written in a temporal logic language [12], with primitives like *eventually* and *always*, and can be used to avoid for instance deadlocks and contradictions or confirm correctness of the system output.

A MC tool performs the comprehensive task of exhaustively exploring the state space of a system in order to assess the reachability of error states. Such tool is composed by two main components [52]: (a) an executor that, for a model and entry states, outputs reachable states; (b) a verifier that, for a given specification and state along the state exploration, checks if the specification is satisfied, returning the state and explored path as a counter-example when the specification fails.

Spin[1] is an example of an open-source tool that can be used for the formal verification of software applications. It is an explicit state model checker since it generates a graph where the nodes are all the global system states and the edges indicate valid state transitions. Specification properties are verified against each of the nodes in the expanded graph. Explicit state model checkers are constrained by the state-space explosion problem, meaning that as models are increasingly more complex, computation becomes as well more time and memory expensive [4].

## 3.2 Comparing classifiers

One has reviewed, in the previous section, the contribution that formal methods, in general, and model checking, in particular, can provide to the verification task of individual AI models. On the use case of risk assessment in hospital emergencies, RSs are either combined or selected by a meta-classification approach. The core work of the tool is leveraged by validated RSs, so one can discuss different options provided by machine learning to perform classification at a complementary level, contrasting the advantages and disadvantages between current and alternative/complementary approaches.

Table 3.1 provides a comparison of ML classifiers. This analysis was conducted as an early stage of the CVD Risk Assessment Tool development process and it is also relevant to address it for the purposes of this dissertation, since it provides an overview of the different knowledge representations and learning algorithms for supervised learning. The representation used to model a complementary classifier has to be compatible with the verification methods and to the type of data considered.

Decision trees are easy to understand but do not handle well the lack of input information, like a specific patient information and Sets of rules have the same issue. These logic based algorithms are the closest types of classification to the RSs considered, which perform a weighted sum of input values, according to their class range.

When it comes to perceptron-based techniques, such as artificial neural networks [56], we can achieve high accuracy, at the expense of possible overfitting situations. The biggest disadvantage of these techniques is the fact that they are black box. Other than that, they

[1]http://spinroot.com/spin/whatispin.html

would be suitable for the issue of handling contradicting data. If no non-deterministic step is used within the net they also would allow for formal specification, for instance via graph representations [55].

Probabilistic approaches to classification allow for missing patient data and to deal with contradicting data. Intuitively, one could derive a strategy for formal specification to this type of classifier by the means of an automaton. At one of its simpler forms, naive bayes [27] classifiers disregard input attributes dependencies.

One can also consider instance based learning [1] or support vector machines [47], but usually they do not deal well with lack of input information, a common scenario in the medical field. Nevertheless, the latter could prove itself useful when the training set is considered small comparing to the number of features available for each instance.

Table 3.1: Classifiers comparison, obtained from [41]

| **Classifier** | **Advantages** | **Disadvantages** |
|---|---|---|
| Logic based algorithms<br>Decision trees | Comprehensibility<br>(easy to understand) | Difficulty/incapability to<br>deal with lack of input information<br>There is no common accepted<br>algorithm to build DT (best features<br>selection, e.g. C4.5)<br>Requires pruning |
| Learning set of rules | Comprehensibility<br>(easy to understand) | Difficulty/incapability to<br>deal with lack of input information |
| Perceptron based<br>Techniques<br>Neural networks | Accuracy | Easy to occur overfitting situations.<br>No comprehensibility<br>Incapability to deal with lack of<br>input information |
| Probabilistic<br>Naïve Bayes<br>(Bayesian networks) | Fast<br>Simple<br>Able to cope with<br>lack of input information | Attributes' independence<br>assumption |
| Instance based learning<br>k-Nearest neighbour | - | Incapability to deal with lack of<br>input information<br>Large computational time for<br>classification<br>Sensitive to the choice of similarity<br>function to compare instances |
| Support vector machines<br>Support vector machine | Suitable when number<br>of features is larger<br>than the number of<br>training instances | Difficulty to deal with lack of<br>input information |

## 3.3 Verification through explanations

The concept of Verifiable-AI is intertwined with other research areas of AI4EU. In specific, it might be relevant to state that one might achieve verification through explanation. In line with this thought one can look for hints on what has been done in order to make so called black-boxes not so black. Black-box is a paradigm where the developer cannot control the inner logic of a system, and is only able to model the behaviour of that system through the inputs and outputs. There are AI approaches along the full monochromatic spectrum, which means that some are not readily understandable (black), while others only allow for partial understanding (gray) and, on the opposite end, inner workings of so called white ones can be understood. Many reasons may contribute to the ability or not to understand the system. One general attribute that can lead to black-boxing is model complexity.

Ensemble learning [46] is a technique that consists in combining a set of models, where each individually is not as powerful as the combination of all. Different approaches range from simple averaging or max voting [32] to more complex ones such as boosting [45] or stacking [54]. One can solve a high-dimensional classification problem by an ensemble of submodels [38]. This approach performs two of the wanted and discussed properties: (1) fault reduction, through expert inspection, possible by the interpretability of low-dimensional submodels; (2) fault tolerance through submodel redundancy, for instance when considering a max aggregation function of the submodels, where it is only necessary for one submodel to be activated in order to trigger an intended output. For the different types of classification, Binary and Multiclass, different approaches are considered. In concrete, on the latter type, one can use a hierarchical misclassification approach to ensure specific safety requirements [38]. Another possibility is to consider the one-versus-rest ensemble, well suited for applications that follow a fail silent paradigm (ISO 26262). This paradigm ensures that when there is no assurances on the output, the model does not provide one, allowing for safe integration as a component of a bigger system. Data pre-processing through data filtering and feature construction [33] can help improve the predictive performance of the ensembles, although compromising overall interpretability. Applications of the above apply to autonomous control scenarios (e.g. airbag release system [24]), as well as decision support or diagnosis (e.g. medical screening [41]).

In terms of interpretability, one can focus on different dimensions, namely the ones in [25]: a) local versus global interpretation and whether it is possible to describe the system as a whole or only locally explain the behaviour of the system; b) time limitations, related to how long does a user of the system has to interpret the explanation provided by the system, therefore touching on the issues of complexity and length; c) nature of the user experience and whether (s)he is proficient or possesses background knowledge about the system context. As requirements for an interpretable model, also in [25], they include not only interpretability, accuracy and fidelity, but also other features essential to the broader scope of machine learning and data mining algorithms, such as ethical requirements like fairness, privacy or usability, and others more generic such as reliability, robustness, causality, scalability and generality, all of them well described by the survey in [26].

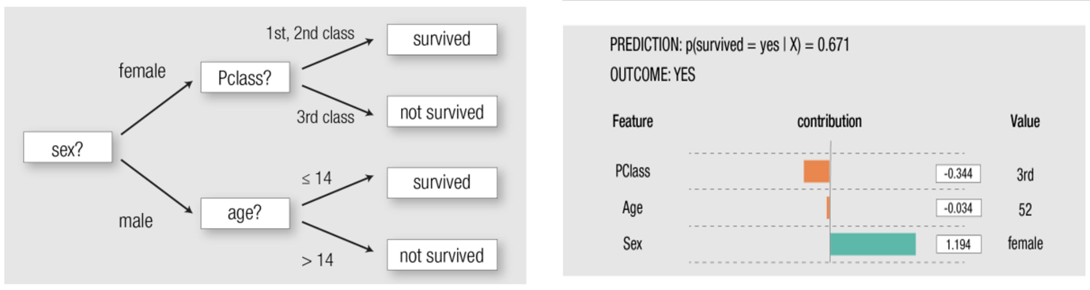


Figure 3.2: Example of a decision tree (left) and feature importance for a linear model (right), obtained from [25]

Also on this work [25] it is identified as easily understandable and interpretable three types of models: decision trees, sets of rules and linear models (as Figure 3.2 shows). The argument is that due to this small set of off-the-shelf interpretable models, one needs to look for frameworks that explain black-boxes. One of such approaches could be used to allow for more complex models in the combination/selection of RSs and afterwards explain the behaviour of the functionality.

# Chapter 4

# Approach

The following chapter describes the steps undergone to create a systematic approach to verify binary classification for forecasting scenarios. Topics will be introduced from a drill-down point of view, also in line with taken research steps. It is worth noting that *classification*, *forecasting* and *risk assessment* will be used throughout the explanations with equivalent meaning, given the use case considered.

## 4.1 High level overview

Figure 4.1 presents a framework for verifying the evaluation made by a RS, along with the key decisions made along its research and development. Although generalisable, the framework will be described as applied to a RS model. RSs are the smallest coding units within the CVD Risk Assessment Tool that can be used individually. The choice of opting for verifying these units allows for a more in-depth revision of the tool, rethinking the way different units can be used together. From a high level perspective, it all starts with an implementation/representation of a RS model, followed by its translation into a specification language, that can be used by a model checker to verify properties both during design time and online. Online verification is particularly relevant for the final step of ensemble classification, where the evaluation performed by the model checker acts as ensemble strategy in the risk assessment task, allowing for integrating with complementary trained classifiers.

To execute this approach Python 3.7 was used as programming language of the RS for fast prototyping. The logic of the Python program was then manually translated into a semantically equivalent Promela modeling language accepted as input by model checker Spin 6.4. Afterwards, assumptions, search ranges and other necessary changes were added to the specification in order to perform verification at design-time and achieve online verification. Output feedback from this last step is then used in a Python classification pipeline as ensemble strategy that also takes into account a complementary ML classification model (previously trained).

An explanation of each of the steps is provided in subsequent sections, providing procedural guidelines to put in practice the verification framework.

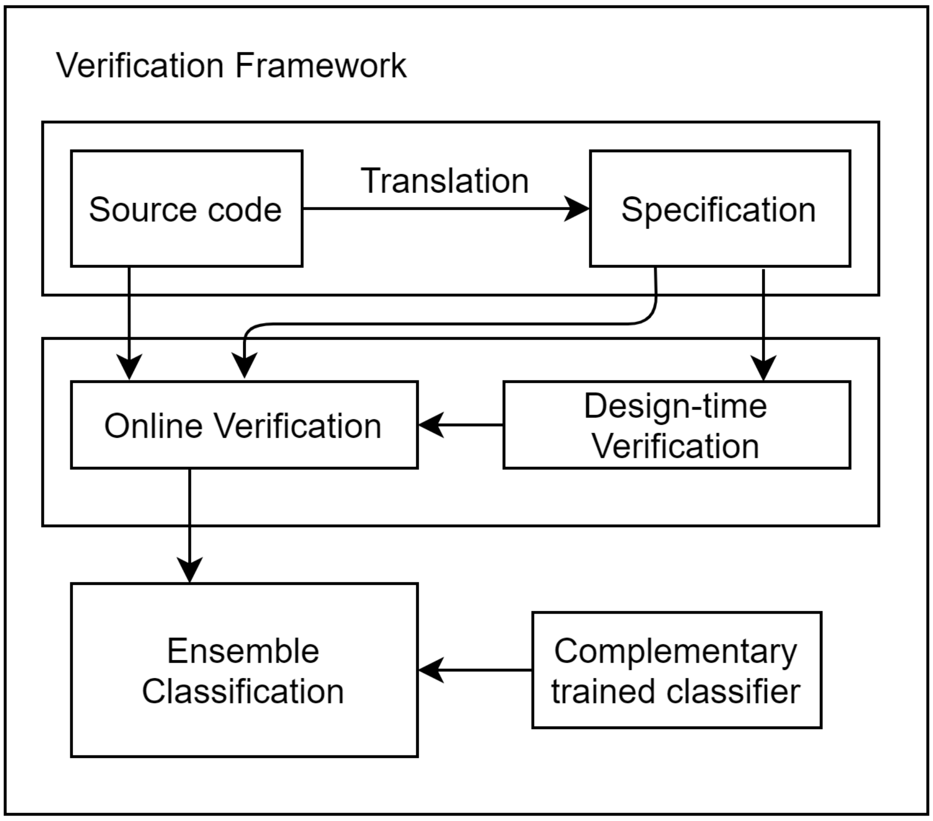


Figure 4.1: High level overview of the verification framework research and development steps

## 4.2 Implementation and Specification

Listing 4.1 shows the implementation of a medical condition RS assessment function. In fact, it is a code snippet of the GRACE source code in Python 3.7. For simplicity, a single risk factor is considered and one can add further risk factors following analogous steps. Risk assessment rules are implemented by conditional expressions and statements. *Age* is a common risk factor among risk scores and often includes several ranges of discrimination of the risk, so it is a good example that shall be used across the explanation of the approach.

A RS receives as input a given patient data, in an iteratable structure, and a pre-determined risk threshold. For each risk factor, a risk value is added to the *risk* accumulator variable according its value. If the accrued risk sum falls above the threshold, the assessment function outputs that the patient is in risk.

At this stage, one has an executable RS assessment function. Unfortunately, one cannot apply directly the selected model checker to the Python source code and one needs to conduct proper translation of the implementation into a specification language. Spin is a popular open-source software verification tool, that analysis Promela verification modelling language.

A few considerations must be taken into account for this step. Since Python has no assigned types, one needs to decide the necessary allocation space for the variables used in the code. Usually, type *int* is suitable for the majority of the risk factors, since it can both store categorical and numerical variables. Variables of interest are declared as *global* in order to

Listing 4.1: Implementation of a risk score in Python

```python
def grace(patient_data, risk_threshold):

    risk=0

    age = patient_data['Age']
    if age >= 40 and age <= 49:
        risk += 15
    elif age >= 50 and age <= 59:
        risk += 29
    #Other age ranges...
    elif age >= 90:
        risk += 80

    #Other risk factors...

    if risk >= risk_threshold:
        patient_in_risk=True
    else:
        patient_in_risk=False

    return patient_in_risk
```

obtain their value state from Spin if a counter example is found for a verification task.

Listing 4.2 presents the specification for the implemented risk score. For convenience, patient information is defined through macros, as Spin has a link to a pre-processor that makes that step transparent to the user, and one can add as many variables as needed to the model specification. This becomes relevant for achieving the verification steps described further on, and allows for passing information from another tool or component during compile time. Additionally to the input risk factor values, it is also provided the risk threshold.

As is, the translation process was straight forward. Each of the risk factor conditionals are defined as guards, like branches in Python. An interesting point to retain is that the order that Spin uses to traverse each guard of a conditional block is not necessarily the specified one, meaning, for instance, that lines 13 and 14 could exchange places and Spin could still visit them in the same order. This is relevant for the verification phase.

Up to this point, the framework considers an implementation of a RS and its translation to a specification language. The next sections explore how one can extract value, in terms of verification, from building on the specification attained.

## 4.3 Design Time Verification

At design time, one has the chance to assess a certain tool before it has been deployed. Besides extensive testing, a verification phase can play an important role in two major areas, namely on verifying *a priori* knowledge about the expected system behaviour and searching for invalid end states or cases of non-determinism, these last two examples of possible consistency checks. Model checking can consequently be used to allow such verification.

It is at design time that one can comprehensively perform global verification, meaning

Listing 4.2: Specification of a risk score using Promela

```
#define AGE
//other risk factor values...
#define RISK_THRESHOLD 145

int age, risk;
bool patient_in_risk;

active proctype Grace() {

    risk=0;

    age= (AGE);
    if
    :: age >= 40 && age <= 49 -> risk = risk + 15
    :: age >= 50 && age <= 59 -> risk = risk + 29
    //other age ranges...
    :: age >= 90 -> risk = risk + 80
    fi;

    //other risk factors...

    if
    :: risk >= (RISK_THRESHOLD) -> patient_in_risk = true
    :: else -> patient_in_risk = false
    fi;
}
```

proving properties about the system as a whole. This is because time constrains are less severe at this stage and one can perform extensive computations that are time consuming, although such approach to verification might prove unfeasible to models with high complexity. An opposite approach, focused on a local perspective of proving properties of the system is provided through online verification and described on the next section. Inherently both approaches are concurrent and complementary.

Starting from the baseline specification already detailed, one will now reason about the modifications necessary to perform consistency checks and verification of compliance with *a priori* knowledge.

### 4.3.1 Consistency checks

Checking for consistency means assessing that the model is in its design correct, providing an output for every possible input scenario and that its output is deterministic, meaning that there is a consistent mapping between input and output, and that this mapping did not occur due to a fortuitous arrangement of the conditions.

Consistency checks are very simple in principle but provide essential assurances in critical scenarios. Also, they allow for reasoning about the model if problems are found, identifying where in the input space would occur a fault in the system.

#### Invalid end states

Checking for invalid end states is assuring that for each input there is a suitable output.

First thing to do is bound the acceptable values for each input feature. In other words, this is defining the ranges of operation values for every input feature. Such information should be available as a requirement, might reflect physical constraints or be estimated based on similar applications. Any input values outside these bounds should be blocked at the interface level.

Once the bounding information is attained, one can formalise it in the specification language and run the model checker. Listing 4.3 shows the modified specification. Now, as input one provides each feature (risk factor) bound, in this case 25 to 100 for the *Age* risk factor. This information is used in line 12, where the *select* command will make the model checker verify the execution of the model for each value within that range. Given more risk factors, Spin will verify every combination possible.

Listing 4.3: Modified specification for invalid end states search

```
 1 #define AGE_MIN 25
 2 #define AGE_MAX 100
 3 //other risk factor range values...
 4 #define RISK_THRESHOLD 145
 5
 6 int age, risk;
 7 bool patient_in_risk;
 8
 9 active proctype Grace() {
10
11     risk=0;
12
13     select(age : (AGE_MIN) .. (AGE_MAX) );
14     if
15     :: age >= 40 && age <= 49 -> risk = risk + 15
16     :: age >= 50 && age <= 59 -> risk = risk + 29
17     //other age ranges...
18     :: age >= 90 -> risk = risk + 80
19     fi;
20
21     //other risk factors...
22
23     if
24     :: risk >= (RISK_THRESHOLD) -> patient_in_risk = true
25     :: else -> patient_in_risk = false
26     fi;
27 }
```

Since there are no other properties specified, the model checker will just try to run through the model and output in a valid state. In order to run it follow the commands in Listing 4.4 on a terminal. You obtain as output general information about the execution that either confirms the correctness or alerts for errors, as well as time and memory statistics.

For the given example, Spin will report finding 1 error and generates a *.trail* file. By default Spin exits once it finds the first counter example that violates the specification. To obtain it one can run Spin again with *-t* flag, as in Listing 4.5.

Listing 4.4: Running the model checker on a terminal for invalid end state search.

```
#!/bin/bash

spin -a specification_invalid_end_states.pml
gcc pan.c -o invalid_end_states -w
./invalid_end_states
```

Listing 4.5: Obtain a counter example.

```
#!/bin/bash

spin -t specification_invalid_end_states.pml
```

Spin will identify each variable value state when the verification ends unsuccessfully. One will find that for this example *age* was 25. This happens since Promela is *blocking* when conditional statements do not contemplate all possible inputs by explicit assignment of a guard to them. In this case, it is a false problem since the wanted behaviour is that *risk* is not increased when patient *age* is less than 40, so this was omitted as condition. In other cases, it would be relevant to identify such issues, as perhaps an above-zero-value could have been missing. Incremental changes and corrections can be made with the help of Spin. For the example one can add an *else* statement guard that unblocks the program like in line 18 of Listing 4.6.

Listing 4.6: Updated specification according the invalid state check

```
 1 #define AGE
 2 //other risk factor values...
 3 #define RISK_THRESHOLD 145
 4
 5 int age, risk;
 6 bool patient_in_risk;
 7
 8 active proctype Grace() {
 9
10     risk=0;
11
12     age= (AGE);
13     if
14     :: age >= 40 && age <= 49 -> risk = risk + 15
15     :: age >= 50 && age <= 59 -> risk = risk + 29
16     //other age ranges...
17     :: age >= 90 -> risk = risk + 80
18     :: else -> skip
19     fi;
20
21     //other risk factors...
22
23     if
24     :: risk >= (RISK_THRESHOLD) -> patient_in_risk = true
25     :: else -> patient_in_risk = false
26     fi;
27 }
```

This task is complete once the model checker returns without errors.

### Non-determinism

Checking for non-determinism is asserting that for a given instance one cannot obtain different classification outputs. This is a verification check that can be executed during testing of specific instances. It also lays the foundations for the online verification procedure, so it will be detailed here. Common issues identified with this step comprise detecting probabilistic behaviour when a deterministic one is wanted and if by chance the order of conditional operations in the model affects the output behaviour.

As in the previous verification task, we need to formalise into the specification what one wants to verify. Listing 4.7 presents the necessary changes. In line 3 it is specified the classification already provided by the Python implemented risk assessment. Then line 13 specifies that verification should occur in specific for the input risk factor value (of the patient considered). Line 29 defines the property to be verified: Spin should make sure that for every possible end state of the specification, the classification is the same as the assessed risk.

Listing 4.7: Specification for non-determinism check

```
 1 #define AGE
 2 //other risk factor values...
 3 #define ASSESSED_RISK
 4 #define RISK_THRESHOLD 145
 5
 6 int age, risk;
 7 bool patient_in_risk;
 8
 9 active proctype Grace() {
10
11     risk=0;
12
13     select(age : (AGE-0) .. (AGE+0) );
14     if
15     :: age >= 40 && age <= 49 -> risk = risk + 15
16     :: age >= 50 && age <= 59 -> risk = risk + 29
17     //other age ranges...
18     :: age >= 90 -> risk = risk + 80
19     :: else -> skip
20     fi;
21
22     //other risk factors...
23
24     if
25     :: risk >= (RISK_THRESHOLD) -> patient_in_risk = true
26     :: else -> patient_in_risk = false
27     fi;
28
29     assert(patient_in_risk == ASSESSED_RISK)
30 }
```

To run the model checker one can use the same set of commands as in Listing 4.4, with the necessary changes to the filenames. If there is nothing to report and the model checker returns without errors, as is the case of the considered model, one can move forward with the development process. If an error is identified, one can use as well the command from Listing 4.5 to obtain a counter example and go back and fix the model specification and implementation.

### 4.3.2 *A priori* knowledge

Ideally, one could rely on expert knowledge or other previously acquired knowledge to bound a system behaviour, at least to some extent. Integrating such knowledge can be done at design-time through the specification and verified by the model checker, as long as it can be specified by temporal logic or assertion structures. In the verification process, one is interested in identifying hard boundaries for the expected behaviour of the system.

Applying this step can be done by building on the updated specification for invalid end state searching, as essentially one wants to verify the whole system model for certain specifications, added at the end of the code, like it has been done for the non-determinism check. Listing 4.8 provides an example of the possible structures and further examples can be found in the corresponding Results section 5.2 or constructed according to the Promela Reference[1]. In specific, Spin will verify the compliance of the specification with the knowledge that *age* above 90 and other risk factor *killip* class above 2 implies that the patient is assessed as high risk.

After running the model checker, it is possible to fall into one of two situations: (a) the specification already verifies the intended base knowledge, or (b) the model checker encountered a counter example that violates it. If (a) happens, design-time verification has finished and one is ready to move towards online verification and deployment. On the other hand, if (b) is to occur, one can deal with the situation on three concurrent ways:

- **going back to the training phase**, refining the data and learning parameters and redoing the verification steps so far, hopefully passing current *a priori* knowledge checks;
- **rewriting the *a priori* knowledge base**, making it less restrictive, which is pertinent when boundaries are set on experimental limits;
- **dealing with it in the application flow**, making sure bad decisions are overridden.

## 4.4 Online Verification

Sometimes it is only possible to have a local understanding of the system behaviour due to its complexity that makes it impracticable to analyse it as a whole. That is why online verification can be seen both as a complement or substitute of design-time verification.

One is concerned with local understanding of the system when considering an instance oriented analysis, where for a given instance one is focused on findings of the instance itself and for small variations of the instance features values. Instance oriented verification is therefore proving that certain properties hold for specific instances, and this makes sense to be performed online because: (a) if it was performed during design time, one would have

[1]http://spinroot.com/spin/Man/ltl.html

Listing 4.8: Specification for verification of *a priori* knowledge.

```
#define AGE_MIN 25
#define AGE_MAX 100
//other risk factor range values...
#define RISK_THRESHOLD 145

int age, risk;
bool patient_in_risk;

active proctype Grace() {

    risk=0;

    select(age : (AGE_MIN) .. (AGE_MAX) );
    if
    :: age >= 40 && age <= 49 -> risk = risk + 15
    :: age >= 50 && age <= 59 -> risk = risk + 29
    //other age ranges...
    :: age >= 90 -> risk = risk + 80
    :: else -> skip
    fi;

    //other risk factors...

    if
    :: risk >= (RISK_THRESHOLD) -> patient_in_risk = true
    :: else -> patient_in_risk = false
    fi;

    ltl rule1 {always (age>=90 && killip> 2) implies (eventually
    patient_in_risk == true)}
    assert(...)
    //other formulations to be verified
}
```

to generate artificial instances that might not be ever actually assessed by the system, (b) generating all the possible instances would be unfeasible for complex models and (c) the output from verification can be further explored by the user and the system in run time. Current research focus on this last premise.

General classifier approaches can provide alongside the classification label some representation of confidence, usually in a probability format. In some cases it is only possible to rely on global notions that might resemble to confidence, like a confusion matrix. Here, one wants to provide a binary confidence indication for a given risk assessment, for a given patient, and in terms to a certain distance measure to a classification boundary. Following classical techniques it would boil down to sensitivity analysis [44], but model checking can add further value to the task.

Confidence is a relevant factor in critical scenarios where we have an augmented AI application, meaning the feedback provided by the intelligent system is used by a human operator to make a decision. In this area, it is pertinent to provide more than a classification output to the user.

Let us explore the online verification as shown in Fig. 4.2. Each time a risk assessment is performed, one uses the implemented RS assessment function to obtain a classification for a given patient. Then, the model checking tool is provided with the specification, previously translated, which is set according to the input data of the patient (the risk factors values) and the risk assessment output of the Risk Score. Through the execution of the tool it checks for counter examples of the expected risk assessment output within the risk factors search space bounded by range parameters. If no counter example is found, the system provides the user with an indication of confidence on the risk assessment.

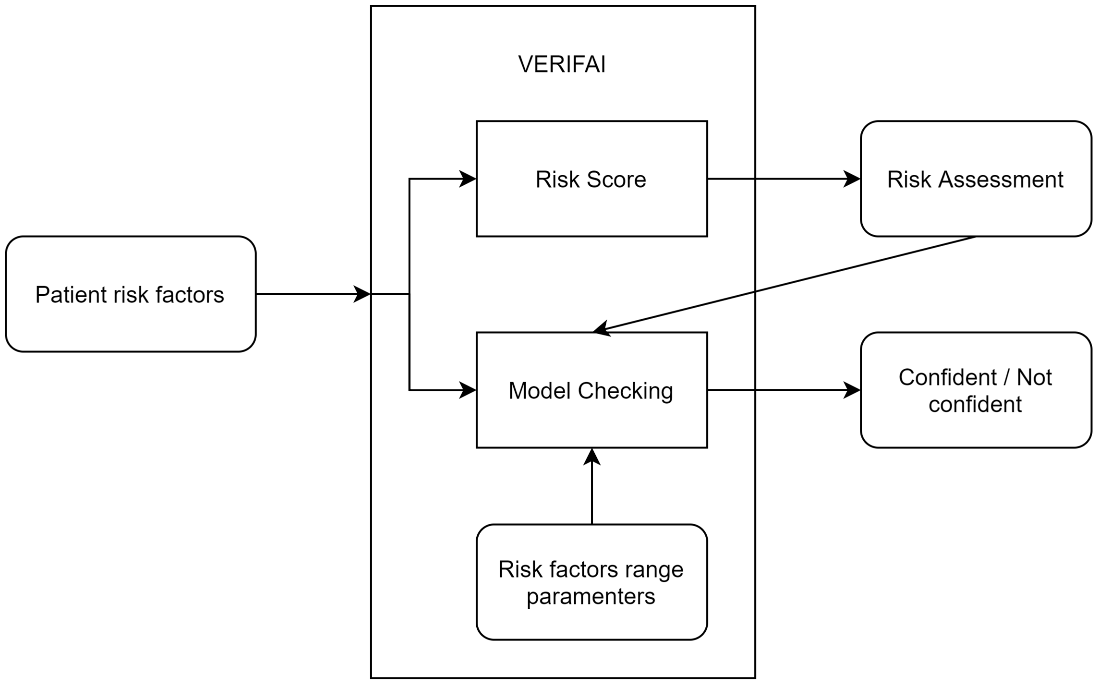


Figure 4.2: High level overview of online verification

Online verification builds on most of the individual changes made to the specification during design-time verification. Besides the straight forward translation, the specification considers some extra elements as presented in Listing 4.9: line 2 and 3 define the search bound parameters for the *Age* risk factor, which are used in line 15 along with the *select* command; line 5 adds the assessed risk for that specific patient that is used in line 31 in the *assert* statement that sets the property we want to verify.

Listing 4.9: Specification for non-determinism check

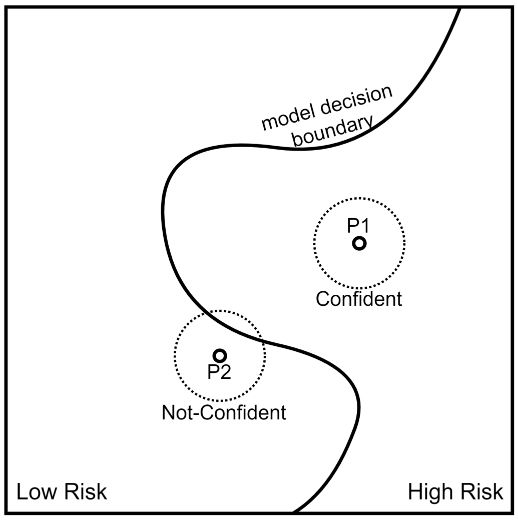
```
#define AGE
#define AGE_MIN_RANGE 5
#define AGE_MAX_RANGE 5
//other risk factor values and ranges...
#define ASSESSED_RISK
#define RISK_THRESHOLD 145

int age, risk;
bool patient_in_risk;

active proctype Grace() {

    risk=0;

    select(age : (AGE- AGE_MIN_RANGE) .. (AGE + AGE_MAX_RANGE) );
    if
    :: age >= 40 && age <= 49 -> risk = risk + 15
    :: age >= 50 && age <= 59 -> risk = risk + 29
    //other age ranges...
    :: age >= 90 -> risk = risk + 80
    :: else -> skip
    fi;

    //other risk factors...

    if
    :: risk >= (RISK_THRESHOLD) -> patient_in_risk = true
    :: else -> patient_in_risk = false
    fi;

    assert(patient_in_risk == ASSESSED_RISK)
}
```

One needs to specify the bounds on which each input variable (risk factor) is going to be verified for the assumption of falling within the same risk assessment. One can limit the verification space through the *select* command, creating an upper and lower interval around the patient risk factor. As is, online verification feasibility is linked with how wide the search is, rather than how complex the model is.

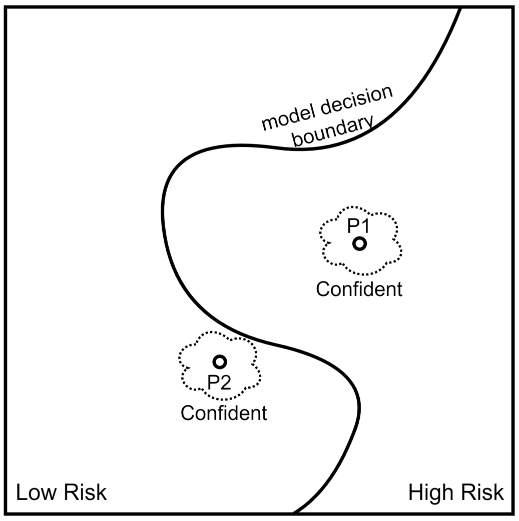
An important consideration is related to which risk factors one wants to vary in our verification process, and that is a significant improvement that this approach can provide. One does not perform ranging for the risk factors that are categorical since different values for those features represent patients that are too far apart, considering the notion of vicinity of medical risk factors search space. In fact, considering them would fall into the regular techniques and perhaps remove the confidence notion from a medical perspective.

Once identified the relevant features (risk factors in this use case), one can perform tuning (during a training phase) of the range parameters in order to obtain better performance.

Figure 4.3 provides a 2D visualisation of the online verification process for two sets of range parameters. The scope is a binary decision space and a model decision boundary that splits



(a) Set of range parameters A



(b) Set of range parameters B

Figure 4.3: Graphical visualisation of the online verification process

the space into low risk (negative) and high risk (positive) assessments. For each instance $P_i$, a vicinity is defined, depicted as closed dotted lines for each $P_i$, that are expressed in terms of ranges of specific features (or risk factors in this use case), and are input of the Model Checker. The MC tool will explore the decision space within this vicinity and check for synthetic examples with a different classification. As it can be seen in 4.3a, vicinity of $\mathrm{P}_2$ partially includes a decision space with different classification from the one of the given instance, being therefore classified as Not Confident. This approach allows for choosing which features one wants to consider and their ranges, which one can see in 4.3b, where a different range of risk factor parameters provides now a Confident indication for $\mathrm{P}_2$. This conceptualisation is valid for multidimensional problems.

### 4.4.1 Explainability Bonus

For a given patient it is verified if the vicinity shall be close the decision boundary. By default, the model checker performs exhaustive search of that space and either returns that no closeness was found or reports a counter example. A small change in this online verification framework can allow for augmenting the confidence binary indication, by providing a counter example (a generated patient) that has a different risk assessment.

One can raise the argument that providing a single counter example might not be so relevant. Luckily, Spin allows generating as many counter examples as existent or up to a pre-determined limit. This is particularly relevant for this use case as the system can provide the user with further information to make an informed decision (to rely or not in the assessment provided by the system).

Such feature circles back the research loop and points towards the explainability efforts in Local Rule-Based Explanations (LORE) [25]. In fact, during current dissertation work, collaboration with an Italian research team was established to build a LORE complementing approach to the selected use case.

## 4.5 Verification for Ensemble Classification

Up to this point, focus has been put on verifying a single unit of what comprises a CVD Risk Assessment Tool. In this subsection, it is described how to connect these units, taking advantage of the verification process.

Online verification identifies zones for a given RS where its risk assessment should be carefully revised by the human user. This creates a gap in the trusted decision space, that can be shortened by the combination of other models.

Since one is not confident on certain risk assessments, the idea is to use another classifier in those cases. Although the use case comprises three RSs, only one ended up having the complexity relevant for online verification, so a decision was made to also test the complementary classifier component as general machine learning models. Moreover, these classifiers can benefit from taking into account further available information that one single given RS does not consider as input, and be trained for a specific pool of patients.

This complementary classifier can be any of the discussed on Related Work 3.2 and is learnt following a classical machine learning pipeline involving: loading the data; performing a correlation analysis, dropping out correlated features (if they are not used by the RS); splitting the data for train and test; using partially the training data to perform parameter tuning, then training with the best configuration.

Having a trained classifier leads to implementation of the ensemble strategy. Figure 4.4 provides in detail the approach to ensemble risk assessment based on the online verification output. For a given patient, a risk assessment A is computed using the Risk Score implementation, which is used along the patient input data to be online verified by the model checker, which provides a confident indication metric. Each time the RS fails the online verification for the given patient, the system outputs instead the risk assessment B, calculated based on the complementary classifier.

A good approach to evaluate and increase the robustness of this setup is to iterate the process several times and for different complementary classifiers, having in mind the advantages and disadvantages of each (as discussed in Section 3.2). While analysing the performance of each strategy and configuration, one will have to consider the percentage of actual use of the complementary model. On normal circumstances, the RS will be making the majority of the classification, which can be further compared against a different parametrisation of the online verification risk factors search bounds.

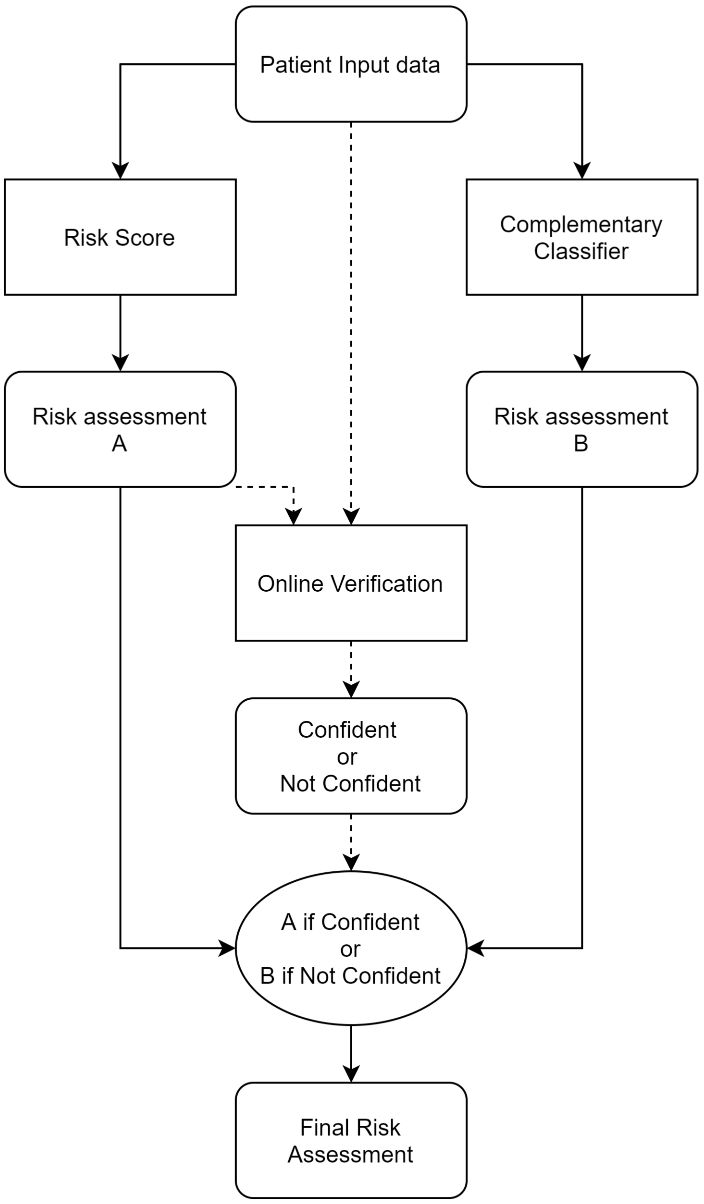


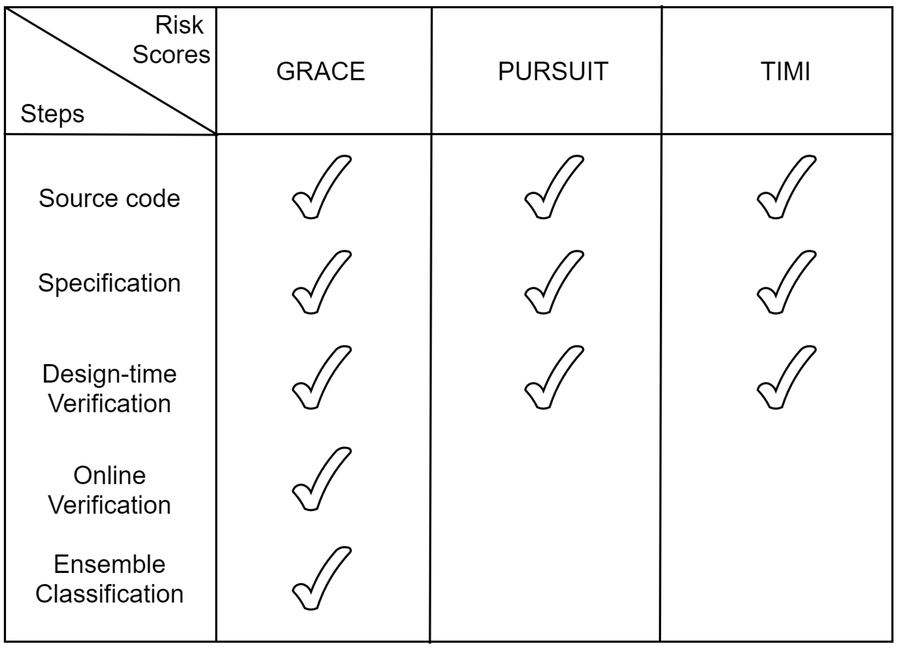
Figure 4.4: Ensemble approach to risk assessment based on the online verification output.

# Chapter 5

# Experimental study and Analysis

In this Chapter, it will be presented the experimentation and validation performed to the proposed Verification Framework. A description of the available data is given, followed by a section for each of the main steps of the approach, reporting on the results and finalising with a discussion on the major outcomes.

Verification steps were performed incrementally and by the presented order. Regarding the available RSs to the risk assessment task, a decision was made to apply the last two steps to a single RS (although the other two are also included for the last step but only as complementary classifiers), as shown in Fig. 5.1. This decision was motivated by the results of the design-time verification, that identified as less relevant the application of online verification to two of the available RSs. This observation was already expected by the presented performance results and discussion of the CVD Risk Assessment Tool [42], which also allowed for a better study of the online verification onto the specific and selected RS, in particular for its potential in the last phase of ensemble classification.



| Steps \ Risk Scores | GRACE | PURSUIT | TIMI |
|---|---|---|---|
| Source code | ✓ | ✓ | ✓ |
| Specification | ✓ | ✓ | ✓ |
| Design-time Verification | ✓ | ✓ | ✓ |
| Online Verification | ✓ | | |
| Ensemble Classification | ✓ | | |

Figure 5.1: Risk Scores evaluated in each verification step.

Experimental evaluation of the framework was done on a Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz machine. An implementation of the framework is made available as Docker container which allows for experimentation, as well as replication of the results that do not consider confidential data. A detailed overview of how to obtain the Docker and run it is provided in Appendix B.

## 5.1 Available Data

Access has been granted to the data used for training, testing and validation of the CVD Risk Assessment Tool, under the premise of being used within the sphere of the Department of Informatics Engineering of University of Coimbra. For that reason, only a global characterisation of the data is therefore presented in Table 5.1.

Table 5.1: Risk factors: baseline characteristics (Santa Cruz data set), obtained from [42]

| **Risk Factor** | **Event** |
|---|---|
| Age (years) | 63.4 ± 10.8 |
| Sex (male/female) | 361 (78.5 %)/99 (21.5 %) |
| Risk factors | |
| Diabetes (0/1) | 352 (76.5 %)/108 (23.5 %) |
| Hypercholesterolemia (0/1) | 180 (39.1 %)/280 (60.9 %) |
| Hypertension (0/1) | 176 (38.3 %)/284 (61.7 %) |
| Smoking (0/1) | 362 (78.7 %)/98 (21.3 %) |
| Previous history/known CAD | |
| Myocardial infarction (0/1) | 249 (54.0 %)/211 (46.0 %) |
| Myocardial revascularization (0/1) | 239 (51.9 %)/221 (48.1 %) |
| PTCA | 146 (31.7 %) |
| CABG | 103 (22.4 %) |
| Sbp (mmHg) | 142.4 ± 26.9 |
| Hr (bpm) | 75.3 ± 18.1 |
| Creatinine (mg/dl) | 1.37 ± 1.26 |
| Enrolment [0 UA, 1 MI] | 180 (39.1 %)/280 (60.9 %) |
| Killip 1/2/3/4 | 395 (85.9 %)/31 (6.8 %)/33 (7.3 %)/0 % |
| CCS [0 I/II; 1 CSS III/IV] | 110 (24.0 %)/350 (76.0 %) |
| ST deviation (0/1) | 216 (47.0 %)/244 (53.0 %) |
| Signs of heart failure (0/1) | 395 (85.9 %)/65 (14.1 %) |
| Tn I &gt; 0.1 ng/ml (0/1) | 313 (68.0 %)/147 (32.0 %) |
| Cardiac arrest admission (0/1) | 460 (100 %)/0 % |
| Aspirin (0/1) | 184 (40.0 %)/276 (60.0 %) |
| Angina (0/1) | 19 (4.0 %)/441 (96.0 %) |

Data represents 460 patients admitted at Santa Cruz Hospital from March 1999 to July 2001, with the specific condition of acute coronary syndrome with non-ST segment elevation - ACS-NSTEMI [42]. This dataset has a positive event rate of 7.2%, comprising 33 events of death or myocardial infarction after 30 days of admission.

Quality medical data is hard to acquire. RSs were obtained from international trials that involved big corporate that do not make their databases public. In fact, in open repositories like UCI ML Repository [17] or Kaggle[1], one can find three types of datasets: (a) raw, unprocessed data sets with several thousands of different types of entries, which makes it hard to evaluate their quality; (b) clean and detailed datasets, with a relevant amount

[1]https://www.kaggle.com

of instances, but that are usually synthetic generated or derived; (c) real data, often well documented, but with far fewer instances available. Opting for one of these three options would leave current work with an added challenge of finding a suitable risk assessment model that considered the available features in those data sets. On top of this, the option of using the already collected, high quality, available data, for the specific purpose of evaluating the use case, was a straight away decision.

## 5.2 Design Time Verification

Experimentation at design-time was a means to move towards online verification and ultimately redefining the aggregation of risk assessment tools. One can consider a previous step as completed, which was already implementing in Python 3.7 each of the risk scores and acquiring a proper translation to Promela, that needed to be ready and available for performing the design checks.

First on the list was the **invalid end states check**. Initial experimentation identified the need for *else* conditional guards, in order to run full verification through the model checking. An *else*-complete version of the RSs specification was then used to report on the feasibility of applying model checking at design time for each of the available models.

The initial task was defining the operation bounds to each of the risk factor features. Table 5.2 presents each of the considered bounds. Also, a common discretisation approach to the feature space was done, in order to optimize the model checking execution. The idea was to consider the minimal viable search step in order to verify each of the possible states. This discretisation is applied only for the numerical variables, having no meaning for the categorical ones, and for that reason those are identified with a *C* in the *Step* column. On the left, one can also see in which RSs a certain risk factor is considered, where a subset of these features are used by more than one.

Table 5.2: Risk factor bounds for invalid end state verification.

| Risk Scores | | | Risk Factor | Lower Bound | Upper Bould | Step |
|---|---|---|---|---|---|---|
| GRACE | PURSUIT | TIMI | | | | |
| Y | Y | Y | Age | 20 | 100 | 10 |
| | Y | | Sex | 0 | 1 | C |
| | | Y | 3 Known CAD | 0 | 1 | C |
| Y | | | Sbp | 50 | 250 | 10 |
| Y | | | Hr | 30 | 250 | 10 |
| Y | | | Creatinine | 0.0 | 4.0 | 0.1 |
| Y | | | Killip | 1 | 4 | C |
| | Y | | CCS | 0 | 1 | C |
| Y | Y | Y | ST Deviation | 0 | 1 | C |
| | Y | Y | Signs of heart failure | 0 | 1 | C |
| Y | | Y | Tn I &gt | 0 | 1 | C |
| Y | | | Cardiac arrest admission | 0 | 1 | C |
| | | Y | Aspirin | 0 | 1 | C |
| | | Y | Angina | 0 | 1 | C |

Spin did not find any invalid end states, and correctly reached all of the valid ones, given the search bounds of operation of each risk factor. Table 5.3 provides the execution results, in terms of states explored, memory used and time spent by the model checker for this verification. One can see substantial differences between GRACE results and both PURSUIT and TIMI. Those results are highlighted because they represent higher complexity of the model and this information was used to determine on which RS to focus on for the online verification phase.

Table 5.3: Running statistics and results of invalid end state verification for the three Risk Scores.

| | **GRACE** | **PURSUIT** | **TIMI** |
|---|---|---|---|
| **States explored** | **1.25e8** | 2.56e3 | 1.28e4 |
| **Memory (MBytes)** | **9187** | 129 | 130 |
| **Time (Seconds)** | **85.3** | 0.04 | 0.04 |
| **Unreached states** | 0 of 187 | 0 of 73 | 0 of 87 |

**Search for non-determinism** was done iterating for each of the patient data, applying the implemented RS and using the output as assertion in the specification. Using *macros* for the specification of the input variables was relevant since one could pass them as argument to the model checker, making the verification pipeline automatic. Spin did not find any non-deterministic example, for the available data, as evidenced in Tab. 5.4 with the corresponding execution statistics. Inherently this is an example of instance oriented verification, although it can be performed for all possible combinations of input patients. For that reason one does not see major discrepancies in the execution time and memory, suggesting pertinence in using online verification for more complex models.

Table 5.4: Running statistics and results of existence of non-determinism verification for the three Risk Scores.

| **Available test cases** | | 460 | |
|---|---|---|---|
| | **GRACE** | **PURSUIT** | **TIMI** |
| **States explored [average]** | 34 | 21 | 27 |
| **Memory (MBytes) [average]** | 128.73 | 128.73 | 128.73 |
| **Time (Seconds) [average]** | 0.0019 | 0.0006 | 0.0003 |
| **Non-deterministic cases** | **0** | **0** | **0** |

The ideal scenario would consider expert knowledge to perform ***a priori* knowledge verification**. Current pandemic constraints did not allow for consulting an expert on the subject. Nevertheless, given the interpretability of the models, the exercise of knowledge extraction and definition was done by model inspection. In fact, assumptions where made like expected from a medical expert, identifying simplified cases where for a given set of symptoms a strict outcome was expected. Table 5.5 presents the identified knowledge statements, and their specification translation to Promela. They were represented as Linear Temporal Logic statements, where one specifies that a specific set of input always implies a determined output. The keyword *eventually* on the left side of the implications denotes the assertion to the specific classification outcome once the model reaches a valid end state.

Table 5.6 reports on the execution results. In fact, this last design-time task of verification could be included and checked at the same time as the invalid end states search, since it is done for the same (whole) operation space. The statistics values are of the same magnitude of the ones obtained for the invalid end states search.

Table 5.5: *A priori* knowledge statements for verification of the three Risk Scores.

| | | **Assumptions [Specification]** |
|---|---|---|
| **GRACE** | **R1_G** | *Patients aged more than 89 and with Killip status above 2 are patients of high risk* [ always ((age >= 90 && killip >= 3) implies (eventually (out == true )) ) ] |
| | **R2_G** | *Patients aged 50 or older with Elevated Cardiac Enzymes (TN) and ST-segment depression are high risk* [ always ((age >= 50 && TN == 1 && DEPST == 1 ) implies (eventually (out == true)) ) ] |
| **PURSUIT** | **R1_P** | *Patients aged less than 50 are low risk* [ always (age<50 implies (eventually (out==false))) ] |
| | **R2_P** | *Patients aged more than 79, male or with at least one more symptom are patients of high risk* [ always (age>79 && (sex ==1 \|\| ccsII ==1 \|\| hf ==1 \|\| DEPST ==1 ) implies (eventually (out==true))) ] |
| **TIMI** | **R1_T** | *Patients aged less than 65 need a combination of 4 out of 6 risk factors to be high risk* [ always ( age<65 && (rf==1 && aas==1 && knCAD==1 && angina==1) \|\| (rf==1 && aas==1 && knCAD==1 && DEPST==1) \|\| (rf==1 && aas==1 && knCAD==1 && TN==1 ) \|\| (...) implies (eventually (out==true)) |
| | **R2_T** | *Patients aged 65 or older need a combination of 3 out of 6 risk factors to be high risk* always ( age<65 && (rf==1 && aas==1 && knCAD==1) \|\| (rf == 1 && aas == 1 && DEPST == 1 ) \|\| (rf == 1 && aas == 1 && TN == 1 ) \|\| (...) implies (eventually (out==true)) |

Table 5.6: Running statistics and results of *a priori* verification for the three Risk Scores.

| | **GRACE** | **PURSUIT** | **TIMI** |
|---|---|---|---|
| **States explored** | 1.82e8 | 3.60e3 | 1.54e3 |
| **Memory (MBytes)** | 13150 | 129 | 130 |
| **Time (Seconds)** | 235 | 0.04 | 0.03 |
| **Conflicts** | **0** | **0** | **0** |

## 5.3 Online Verification

Moving forward to the online verification, evaluation and experimentation was done for the GRACE RS, since it was the one that revealed superior complexity (several magnitude units) in terms of possible and explored states in the overall verification process at design-time. The idea is that online verification can solve the verification problem when this is unfeasible at design-time, but also add value for the application being verified.

Current approach to online verification allows to provide a binary confidence metric that is instance (patient) driven, based in terms of a tailored distance measure to the decision boundaries of the RS model. Model checker searches a defined vicinity space of the patient, looking for possible artificial instances (points in the search space) that are classified with a different risk category.

A key improvement allowed by this approach is that one can tailor the vicinity measure and specify it for each of the considered risk factor features. Eventually, a medical user can even select at run-time for a given patient, the vicinity (s)he wants to use as confidence region. One can, nonetheless, perform fine tuning of the ranges for the available data. This is relevant to manage the trade off between the number of patient instances that are assessed as *not confident* and the performance improvement for the pool of instances that the model checker reports as *confident*.

Table 5.7 shows the considered range variation parameters passed to the model checker to perform online verification. These were selected to consider less than half of the discriminant divisions for each of the risk factors, and to contemplate a confident group size of above 75% of the patients available for performance evaluation. There were no obvious benefits on considering uneven lower and upper bounds for a given risk factor, and as-is (an even pair) the notion of vicinity is more intuitive.

Table 5.7: Range variation parameters used in the GRACE model specification.

| **Risk factor** | **Range variation** |
|---|---|
| Age | +/- 4 (years) |
| Heart rate | +/- 5 (beats/min) |
| Systolic blood pressure | +/- 3 (mm Hg) |
| Creatinine | +/- 0.1 (mg/dL) |

Table 5.8 aggregates the relevant metric results of the application of the online verification strategy to the GRACE RS with the available data. Results are reported for an approach without online verification (first column with values - Grace baseline), in contrast with one that does, reported in the two most right columns, since the approach splits the pool of patients given their confidence assessment. Performance metrics were calculated for each approach and group of interest. One can see that for the approach with online verification, 83.5% of the patients are classified as Confident for their risk assessment, whereas the rest are labeled Not-Confident. For the first group, called Group 1, all the metrics are improved, i.e. its value is increased, in contrast with Group 2, where all but Sensitivity have decreased in its values. In fact, for Group 2, one can see an increase of Sensitivity to 100%, meaning classifying correctly all the high risk patients, but that also meant classifying more low risk patients as high risk, leading to a drop in Specificity of 8%. In comparison, Group 2 decreases are more substantial than the Group 1 increases.

Also relevant information as results are the Spin execution statistics, which allow for the discussion of the applicability of Spin tool in run time and Tab. 5.9 reports precisely on that. One can see that Spin executed always under 0.05 seconds, keeping the memory used

Table 5.8: Results from the overall GRACE risk score and for the GRACE with online verification

| | Grace Overall | Grace w/ Online Verification | |
|---|---|---|---|
| | | Group 1<br>Confident Assessed | Group 2<br>Not Confident Assessed |
| **Patients** | 460 | 384 (83.5%) | 76 (16,5%) |
| **F1 Measure** | 0.21 | 0.23 | 0.09 |
| **Accuracy** | 0.55 | 0.57 | 0.46 |
| **Sensitivity** | 0.79 | 0.81 | 1.00 |
| **Specificity** | 0.53 | 0.55 | 0.45 |

not higher than 132.05 Megabytes. The states explored, and consequently the memory and time are intrinsically related to the defined range bounds of verification for each instance. It would be to expect an increase of these values if a larger portion of the vicinity of each instance was explored by the model checker, and the opposite if the vicinity considered was smaller.

Table 5.9: Spin execution statistics: number of states stored, memory usage and execution time during online verification.

| **States** | | | |
|---|---|---|---|
| Average | 61036 | Max | 73012 |
| **Memory (Megabytes)** | | | |
| Average | 131.50 | Max | 132.05 |
| **Time (seconds)** | | | |
| Average | 0.03 | Max | 0.05 |

In terms of the explainability feature of online verification, the approach is able to generate not only one but all the synthetic generated counterexamples for each given instance. Listing all those counterexamples extensively here, for each of the patients labelled Not-Confidently assessed, would be impracticable. This is also true from a real world application point of view. In part, that is what the Italian team is currently focusing on, and pursuing further analysis to the generated set of counterexample patients would fall outside the scope of verification and overlap with the efforts of the other team. For that reason, one can experiment with the generation of all counterexamples through the Docker container, but no further results are presented in this section.

## 5.4 Verification for Ensemble Classification

This section addresses the challenges faced by the CVD Risk Assessment Tool of how to incorporate different available tools into a meaningful classification output and experiments on the use of the online verification step as an ensemble strategy for classification. Evaluation and results were done for a pairwise scenario, i.e. considering the ensemble of two models, the first one with an available online verification component. For that reason, it was also possible to study the influence of more general classification algorithms in the identified *not confident* regions.

The experimentation scenario derives from the classical machine learning pipeline, since a training phase is necessary for the complementary classification algorithms that learn from a portion of the available data. The pipeline is run several times, for stratified, randomly split sets of data. The training set is partially used to perform parameter tuning of the classifiers, and over-sampled on the positive class to provide balancing and improve learning. Test set is used to assess the performance of individual approaches versus the ensemble strategy, for the several available complementary classifiers. Validation is the result of averaging the performance for the independent test runs that provide three selected metrics: F1 score, Sensitivity and Specificity. Each of the relevant experimentation parameters are detailed in Tab. 5.10.

Table 5.10: Experimental setup parameters.

| | |
|---|---|
| **Runs** | 50 |
| **Positive Event Threshold** | 40 days |
| **Classifier w/ Online Verification** | GRACE |
| **Complementary classifiers (sklearn)** | DecisionTreeClassifier, LogisticRegression, KNeighborsClassifier, GaussianNB, SVC |
| **Complementary classifiers (RSs)** | PURSUIT, TIMI |
| **Stratified Split Train/Test** | Yes |
| **Test set** | 33% of available data |
| **Training set** | 67% of available data |
| **Parameter tuning set** | 15% of available data (part of Training set) |
| **Training oversampling strategy** | SMOTE [11](for positive class) |
| **Metrics collected** | f1 score, sensitivity, specificity (of Test set) |

Table 5.11 provides the classifier class instances of the Python *sklearn* library used as complementary risk assessment models. Some of the parameters were optimised during training, for the detailed values under the optimised parameters column. Recall that this optimisation was done as part of the training phase, with 15% of the overall stratified available data for each run.

Table 5.11: Hyperparameter optimisation values and final instances of each classifier class.

| | **Parameters optimised** | **Final Classifier Class instance** |
|---|---|---|
| **Dec Tree** | *{'criterion':('gini', 'entropy'), 'splitter':('best','random') }* | *DecisionTreeClassifier (criterion='gini', splitter='best', max_depth=10, min_samples_split=2, min_samples_leaf=2, max_features='auto', class_weight={0: 1, 1: 5})* |
| **Log Reg** | *{'solver':('liblinear','lbfgs', 'saga'), 'C':(0.5,1,2)}* | *LogisticRegression(penalty='l2', tol=0.001, C=0.5, solver='liblinear', max_iter=1000)* |
| **Knn** | *{'n_neighbors':(5,10,15), 'weights':('uniform','distance'), 'algorithm':('ball_tree','kd_tree')}* | *KNeighborsClassifier (n_neighbors=5, weights='uniform', algorithm='ball_tree', leaf_size=30, p=2, metric='minkowski')* |
| **NB** | – | *GaussianNB(priors=None, var_smoothing=1e-09)* |
| **SVM** | *{'kernel':('linear', 'poly', 'rbf', 'sigmoid'), 'C':(0.5,1.0,2)}* | *SVC(C=2.0, kernel='linear', degree=3, gamma='scale', coef0=0.0, shrinking=True, tol=0.001, decision_function_shape='ovr')* |

The performance evaluation was first done for each of the individual classifiers. Table 5.12

shows the average for the several test runs, for the different classifiers and metrics. From the table, one can see that GRACE outperformed all of the complementary approaches on F1 measure (F1) and Sensitivity (SE). This is both true for the domain specific, already trained, risk scores PURSUIT and TIMI, but also for the other models trained with the available data. Specificity metric, that represents how accurate the models are in classifying as low risk patients of that risk category, shows mixed results, were some of the simpler classifiers tended to overclassify as high risk, like TIMI and Naive Bayses (NB), while the others expressed the oppositive behaviour which resulted on higher specificity values than the GRACE baseline.

Table 5.12: Results for the individual performance of each of the classifiers.

| **Classifier** | **Performance** | | |
|---|---|---|---|
| | **F1** | **SE** | **SP** |
| **GRACE (Baseline)** | **0.20** | **0.79** | 0.53 |
| **PURSUIT** | 0.19 | 0.65 | 0.61 |
| **TIMI** | 0.16 | 0.72 | 0.44 |
| **DecTree** | 0.12 | 0.21 | 0.82 |
| **LogReg** | 0.18 | 0.51 | 0.69 |
| **Knn** | 0.10 | 0.29 | 0.65 |
| **NB** | 0.15 | 0.61 | 0.45 |
| **SVM** | 0.19 | 0.51 | 0.70 |

Evaluation was then performed for the ensemble strategy, considering the same randomly split sets of data. It was taken into consideration two configurations of the search ranges parameters, as Tab. 5.13 specifies, which lead to different average values for the GRACE overall use in the ensemble approach. Higher ranges meant less usage on average for the GRACE classifier, while at the same time increasing the complementary classifiers influence to the performance results. Table 5.13 provides the averaged result values from the experimentation on the test sets. The Decision Tree classifier with a low overall usage is the complementary model that provided the ensemble configuration with the higher F1 score, and that was due to the high discrimination of low risk patients (high value on Specificity), although degrading by 3% the sensitivity metric. An overall balanced configuration was the one that combined GRACE to PURSUIT, on the range parameters set that allowed for higher usage of the complementary model, where there was an increase of Specificity without compromising Sensitivity. The rest of the configurations show that performance in general degraded when the overall usage of GRACE is decreased.

The study of these two ensemble configurations ends the experimental evaluation performed to the verification framework.

## 5.5 General Discussion

Here, a discussion is elaborated on the obtained and presented results, highlighting their significance and implications.

One should start by assessing the results obtained at design-time. Three experimental verification checks were performed, for the three available RS. Expectedly, the model checker neither found invalid end states (Tab. 5.3), cases of non-determinism (Tab. 5.4) nor violations to the *a priori* knowledge (Tab. 5.6), in all the RS considered. These are important results as the verification at design-time was a means to set the scene for further exploration and evaluation of the potential of verification at run time. Structural checks verified

Table 5.13: Results for ensemble strategy based on two online verification sets of parameters for the GRACE risk score, combined with available complementary classifiers.

| | | | | | | |
|---|---|---|---|---|---|---|
| **GRACE (Baseline)** | **F1** | 0.20 | **SE** | 0.79 | **SP** | 0.53 |

| **Online Verification Parameters** | **age = ±6, hr = ±8, sbp = ±9, creat = ±0.3** | | | **age = ±4, hr = ±5, sbp = ±3, creat = ±0.1** | | |
|---|---|---|---|---|---|---|
| **GRACE avg usage** | 74,3% | | | 83,7% | | |
| **Complementary Classifier** | **Performance** | | | | | |
| | **F1** | **SE** | **SP** | **F1** | **SE** | **SP** |
| **PURSUIT** | **0.21** | **0.79** | **0.56** | 0.21 | 0.79 | 0.55 |
| **TIMI** | 0.19 | 0.79 | 0.51 | 0.20 | 0.79 | 0.51 |
| **DecTree** | 0.21 | 0.68 | 0.62 | **0.22** | 0.76 | **0.59** |
| **LogReg** | 0.19 | 0.68 | 0.59 | 0.21 | 0.76 | 0.57 |
| **Knn** | 0.20 | 0.72 | 0.58 | 0.20 | 0.76 | 0.56 |
| **NB** | 0.19 | 0.73 | 0.53 | 0.20 | 0.77 | 0.53 |
| **SVM** | 0.19 | 0.67 | 0.59 | 0.21 | 0.76 | 0.57 |

the dependability of the models, focusing on providing global assurances. This means, in specific for the use case considered, that medical teams know that the models will comply with pre-determined knowledge statements for all the possible input combinations and henceforth all the patients. Medical teams also get a proof that for the models input operation ranges, meaning again all the patients, the RS models will always provide a risk assessment. For the available data, it is also possible to prove that the models are deterministic when providing a risk assessment. These are the kind of assurances that lead to the adoption of such tools in day-to-day operations of emergency hospital risk assessment.

There are, nonetheless, significant differences between the three risk scores when it comes to the execution statistics. This is mainly due to the global scope of the verification process at design-time. One can see that Spin explored more states, used more memory and took considerably more time, approximately 100 times more on the invalid end state check (Tab. 5.3) and 200 times more on the *a priori* knowledge verification (Tab. 5.4), for the GRACE RS verification tasks, when compared to the other two evaluated RS. This confirms the claims that global verification at design-time can become, for more complex models, unfeasible. From these findings we can conclude that verification is time and memory constrained, and that GRACE is a more complex model than the others. This last observation was the basis for the decision of following further experimentation only for this risk score, as the others proved too trivial to justify an online verification step. This result indicates that medical teams could benefit from further information about the models that goes beyond the global assurances, and that such assurances might be for some more complex models, not obtainable in a timely manner.

As far as the results of the online verification are concerned (Tab. 5.8), the majority of the patients, 384 out of 460, fall within Group 1, i.e., the patient group where the risk assessment is categorised as an unambiguous classification, given the considered set of vicinity parameters. This is an important result that confirms the relevance of the proposed framework. If Group 2 was to have more patients than Group 1, medical teams could easily disregard the application of the risk score and all the verification framework. With the current configuration and splitting proportion, one does not penalise too much on the use of the GRACE risk score.

From Table 5.8, the risk assessment F1 measure, the weighted average of precision and recall, is improved for Group 1 by 2% and degrades in Group 2 (by 12%), when comparing with the GRACE overall application. This again justifies the pertinence of applying online verification and the assumption that patients that fall close to the decision boundaries tend to be misclassified. Accuracy falls the same trend, increasing slightly for Group 1 and degrading substantially in Group 2, but further analysis of the other metrics is required to extract conclusions.

Looking at the Sensitivity and Specificity metrics it is possible to see that the GRACE model tends to classify as high risk (positive event) in the vicinity of the decision boundary. This increases the value of Sensitivity to its maximum in Group 2, but it ruins the Specificity score (dropping 13%). This is interesting evidence that can be taken into account when setting the threshold of risk discrimination. Such information makes also medical teams more aware of patterns in the classification bias, which may be taken in consideration when the user takes action regarding or not the risk assessment provided by the risk scores.

In comparison, Group 2 decreases in performance are more substantial than the Group 1 increases. This is relevant contribution to further GRACE overall validation, meaning that without verification the RS performs already very adequately. This also means that one has successfully identified a region of the decision space where its performance degrades badly, namely close to the multidimensional decision boundary of the model, and again this information can be further used to fine tune the decision threshold. These results establish trust in the risk assessment for the confident verified instances, and allow for further due diligence since the model checker provides the medical team with counter examples that might explain the pertinence or not of such classification.

The model checker execution time, memory used and states explored show that the framework can be used together with a medical software tool. Time is an important factor for medical teams, since they need to make decisions and analysis in a timely manner. Thus, and taking into account the results presented in Tab. 5.9, it is possible to see that the usage of a model checker does not increase significantly the execution time. Another important result is concerned with the fact that only a small portion of the state space is explored each time for each patient, compared with the reported for the global states exploration done during design-time (which was 4 orders of magnitude higher as per Tab. 5.3). In fact, it can suggest that this approach is suitable to be applied to larger and more complex models.

The last set of experiments pushed the verification sphere and assessed the possibility of using online verification to decide which model should make a classification for a given instance based on the confidence indication check, using a complementary classifier whenever there was lack of confidence for the verified one. The initial experimentation performed a comparison between the application of each of the classifiers individually. Classifiers that are domain dependent were considered, like the risk scores, as well as more general approaches that were subject to a training and hyperparameter optimisation phase. From Tab. 5.12 one can see that the GRACE baseline was not surpassed for the F1 measure nor the Sensitivity by any of the other tested classifiers. One should be careful looking for the Specificity results since the test data set is highly unbalanced, so in general higher values for that metric in comparison to sensitivity indicate that some models did not learn well from the data and were therefore biased to a low risk (negative) classification. In practice, if such classifiers were to be used individually by medical teams, this would represent mistreating several high risk patients. In fact, nor the opposite approach would be ideal, being that the reason for careful analysis of both Sensitivity and Specificity.

In Tab. 5.13 are presented the ensemble results for two online verification sets of parameters.

Increasing the risk factor ranges allowed for a reduction on the overall usage ratio of the GRACE classifier by approximately 9%. This was done to simulate two different confidence binary thresholds. In general, the classifiers that learnt from the available data did not contribute to an increase in performance of any ensemble strategy, on either online verification ranges configuration. This was already a fair assumption as GRACE was derived from a much larger sample of patients, and machine learning algorithms benefit from large amounts of data (unbiased and assuming the same quality) for improving their performance. PURSUIT provided relevant complement classification, improving on the correct classification of low risk patients (as observable from the maintenance of sensitivity and increase of 3% and 2% for the specificity in the ensemble runs). TIMI did not provide a good complement classification, as in contrast to PURSUIT, since it tended to classify as high risk when the patients were not. These results can help medical teams understand the benefits and constraints of using these specific risk scores as complement to GRACE.

The only exception for the trained-on-the-available-data models was the Decision Tree Classifier. It was indeed the complementary classifier that provided the overall best F1 measure and Specificity for both ensemble configurations, although degrading the sensitivity score. If for the less used scenario it degraded only 3% on Sensitivity, it did reduce the correctly identified high risk patients by 11% when it was more applied (in the first configuration). From these observations one can conclude that the biased low risk classification benefited the ensemble configuration when it was less used (second scenario), since it provided the right counterbalance to the bias for high risk classification of GRACE near the decision boundary, which in principal reflects the pertinence of using ensemble approaches.

The results obtained are in general positive, either in terms of performance achievements or efficiency in the execution statistics. The extensive experimentation, both from a global and local verification point of view, considering its application in design and run time, provides valuable information that is relevant to medical teams currently relying on the risk assessment provided by the considered RSs in this use case.

# Chapter 6

# Conclusion

Current work addressed the problem of verifying intelligent classification systems applied in an Emergency Hospital Risk Assessment use case, analysing the use of Model Checking to perform verification both at design and run time. An approach has been detailed and experimentation and evaluation to the use case described and discussed in the previous chapters. Here, one will review and summarise the main contributions of this work, identify its limitations and provide possible avenues for future research.

This work applies the formal verification technique of model checking to increase trust in the application of medical condition Risk Scores. Verification is done at design-time to (a) prove the absence of invalid end states for a machine learnable forecasting model, (b) verify non-determinism in the classification for the available test data and (c) assert the compliance with *a priori* formulated knowledge. At run time, model checking is used to assess the confidence of classification for a given instance, based on a vicinity search for closeness to the model decision boundary. The overall contribution of this work is therefore a systematic approach to both global and local verification, performed at design and run time for intelligent classification systems.

Besides performing global verification to medical forecasting models, novelty of this work arises from locally identifying regions where the risk assessment forecast may vary for small patient risk factors changes, using a model checker to perform an efficient state exploration of the decision space. The proposed technique consists in verifying whether an input to a Risk Score model is in proximity to its multidimensional decision boundary. This principle increases trust in the Risk Score tool and identifies cases in which the binary risk assessment should be further examined by medical teams.

With online verification as defined in this approach, one performs a tailored analysis of the decision space. Common approaches disregard problem specific information that is relevant for that analysis. For instance, medics are not interested on a distance verification that is based on all the features (risk factors), because some of them are binary or categorical and patients with different values for those features are not considered "close" from a medical perspective.

The verification complexity of the model checker is sufficiently small to allow for online verification results to be produced efficiently. This is, in part, a consequence of the model checking technique itself, which expands the graph of reachable states and is therefore more efficient than both testing and sensitivity analysis.

Ensemble strategies based on the output of online verification showed that it is possible to consider a complementary classifier to balance possible bias along the decision boundary, and this is an actionable contribution to the CVD Risk Assessment Tool from where this work took on.

This dissertation describes the practical implementation of the proposed verification framework, starting with Python models for risk score computation that are translated into the Promela language. The Promela models are used throughout the approach to specify the verification checks and the verifier generated by Spin may be executed to either confirm or provide counterexamples of violations of the specification.

The expected impact for practice is a greater ability to achieve verification of machine learnable models.

## Limitations

Research was limited in time, and therefore options and decisions had to be made accordingly to guarantee success in the overall execution of the tasks programmed. One of such decisions was limiting the scope of application of the proposed approach to a specific use case. Such decision made it possible to address verification for logic-based classifiers, but left other approaches without further consideration.

Another point worth mentioning is the available data. The approach was validated on 460 patients with a positive event rate of 7.2%, or 33 events of death or myocardial infarction after 30 days of admission. From a research point of view, the more instances the better. If for the strict evaluation of the behaviour of the verification framework, for the pre-trained forecasting models, the dataset is of adequate size and representation, one might also argue that it was small in order to perform training in the ensemble phase of the complementary classifiers, even though an oversampling technique has been applied to the training data.

Due to the current pandemic situation, collecting feedback from a medical perspective was not possible. That led to the need of generating *a priori* knowledge statements synthetically, which is not the ideal scenario.

## Future work

Current work can be generalized in multiple directions. First, its scalability can be evaluated on larger data sets than the one used from the CVD Risk Assessment Tool previous validations. Second, it can be also evaluated on other tasks than binary classification, such as multi-class classification and regression. Both these directions might be feasible for healthcare or other critical domains.

Another generalization would be to use another class of model checking approach. The Spin model checker currently used in this work is based on Bű̆chi automata operations [3]. Other model checking approaches could be considered as well, notably symbolic model checkers based on Binary Decision Diagrams operations [3], Satisfiability Modulo Theory [16] or Constraint Logic Programming [18] solvers.

# References

[1] David W Aha, Dennis Kibler, and Marc K Albert. Instance-based learning algorithms. *Machine learning*, 6(1):37–66, 1991.

[2] B E Backus, A J Six, J H Kelder, W B Gibler, F L Moll, and P A Doevendans. Risk scores for patients with chest pain: evaluation in the emergency department.Backus, B. E., Six, A. J., Kelder, J. H., Gibler, W. B., Moll, F. L., & Doevendans, P. A. (2011). Risk scores for patients with chest pain: evaluation in the emergency depart. *Current cardiology reviews*, 7(1):2–8, 2011.

[3] Christel Baier and Joost-Pieter Katoen. *Principles of model checking.* MIT press, 2008.

[4] Raul Barbosa and Johan Karlsson. Formal specification and verification of a protocol for consistent diagnosis in real-time embedded systems. *SIES'2008 - 3rd International Symposium on Industrial Embedded Systems*, pages 192–199, 2008.

[5] Osbert Bastani, Yewen Pu, and Armando Solar-Lezama. Verifiable reinforcement learning via policy extraction. *Advances in Neural Information Processing Systems*, 2018-December(NeurIPS):2494–2504, 2018.

[6] Michel E Bertrand, Maarten L Simoons, Keith A A Fox, Lars C Wallentin, Christian W Hamm, Eugene McFadden, Pim J De Feyter, Giuseppe Specchia, Witold Ruzyllo, and Task Force on the Management of Acute Coronary Syndromes of the European Society of Cardiology. Management of acute coronary syndromes in patients presenting without persistent ST-segment elevation. *European heart journal*, 23(23):1809–40, dec 2002.

[7] Arjun Nitin Bhagoji, Daniel Cullina, Chawin Sitawarin, and Prateek Mittal. Enhancing robustness of machine learning systems via data transformations. *2018 52nd Annual Conference on Information Sciences and Systems, CISS 2018*, pages 1–5, 2018.

[8] Rich Caruana and Alexandru Niculescu-Mizil. An empirical comparison of supervised learning algorithms. *ACM International Conference Proceeding Series*, 148:161–168, 2006.

[9] Sagar Chaki and Arie Gurfinkel. *BDD-Based Symbolic Model Checking*, pages 219–245. Springer International Publishing, Cham, 2018.

[10] Girish Chandrashekar and Ferat Sahin. A survey on feature selection methods. *Computers and Electrical Engineering*, 40(1):16–28, 2014.

[11] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. SMOTE: Synthetic Minority Over-sampling Technique. *Journal of Artificial Intelligence Research*, 16(Sept. 28):321–357, jun 2002.

[12] Edmund M. Clarke, E Allen Emerson, and A Prasad Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 8(2):244–263, 1986.

[13] Edmund M Clarke Jr, Orna Grumberg, Daniel Kroening, Doron Peled, and Helmut Veith. *Model checking*. MIT press, 2018.

[14] Fabrizio D'Ascenzo, Giuseppe Biondi-Zoccai, Claudio Moretti, Mario Bollati, Pierluigi Omedè, Filippo Sciuto, Davide G. Presutti, Maria Grazia Modena, Mauro Gasparini, Matthew J. Reed, Imad Sheiban, and Fiorenzo Gaita. TIMI, GRACE and alternative risk scores in Acute Coronary Syndromes: A meta-analysis of 40 derivation studies on 216,552 patients and of 42 validation studies on 31,625 patients. *Contemporary Clinical Trials*, 33(3):507–514, 2012.

[15] Pedro De Araújo Gonçalves, Jorge Ferreira, Carlos Aguiar, and Ricardo Seabra-Gomes. TIMI, PURSUIT, and GRACE risk scores: Sustained prognostic value and interaction with revascularization in NSTE-ACS. *European Heart Journal*, 26(9):865–872, 2005.

[16] Leonardo De Moura and Nikolaj Bjørner. Satisfiability modulo theories: introduction and applications. *Communications of the ACM*, 54(9):69–77, 2011.

[17] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

[18] Thom Frühwirth and Slim Abdennadher. *Essentials of constraint programming*. Springer Science & Business Media, 2003.

[19] Nathan Fulton and André Platzer. Safe reinforcement learning via formal methods: Toward safe control through proof and learning. *32nd AAAI Conference on Artificial Intelligence, AAAI 2018*, pages 6485–6492, 2018.

[20] John Gagliardi. Medical Device Software: Verification, Validation and Compliance. *Biomedical Instrumentation & Technology*, 45(2):95–95, mar 2011.

[21] Barbara Gallina, Irfan Sljivo, and Omar Jaradat. Towards a safety-oriented process line for enabling reuse in safety critical systems development and certification. *Proceedings of the 2012 IEEE 35th Software Engineering Workshop, SEW 2012*, pages 148–157, 2012.

[22] Ian Goodfellow, Patrick McDaniel, and Nicolas Papernot. Making machine learning robust against adversarial inputs. *Communications of the ACM*, 61(7):56–66, 2018.

[23] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in Neural Information Processing Systems*, 3(January):2672–2680, 2014.

[24] Xianguang Gu, Jianwei Lu, and Hongzhou Wang. Reliability-based design optimization for vehicle occupant protection system based on ensemble of metamodels. *Structural and Multidisciplinary Optimization*, 51(2):533–546, feb 2015.

[25] Riccardo Guidotti, Anna Monreale, Salvatore Ruggieri, Dino Pedreschi, Franco Turini, and Fosca Giannotti. Local rule-based explanations of black box decision systems, 2018.

[26] Riccardo Guidotti, Anna Monreale, Salvatore Ruggieri, Franco Turini, Fosca Giannotti, and Dino Pedreschi. A survey of methods for explaining black box models. *ACM Computing Surveys*, 51(5):1–45, 2018.

[27] Rish Irina. An empirical study of the naive bayes classifier. *IJCAI 2001 workshop on empirical methods in artificial intelligence*, 3(22):4863–4869, 2001.

[28] Bergstra James and Bengio Yoshua. Random Search for Hyper-Parameter Optimization. *Journal of Machine Learning Research*, 13(1):281–305, 2012.

[29] John Knight. *Fundamentals of Dependable Computing for Software Engineers*. Chapman and Hall/CRC, 2012.

[30] Philip Koopman and Michael Wagner. Challenges in Autonomous Vehicle Testing and Validation. *SAE International Journal of Transportation Safety*, 4(1):15–24, 2016.

[31] Vrushali Y. Kulkarni and Pradeep K. Sinha. Pruning of random forest classifiers: A survey and future directions. *Proceedings - 2012 International Conference on Data Science and Engineering, ICDSE 2012*, pages 64–68, 2012.

[32] Stuart E. Lacy, Michael A. Lones, and Stephen L. Smith. A comparison of evolved linear and non-linear ensemble vote aggregators. *2015 IEEE Congress on Evolutionary Computation, CEC 2015 - Proceedings*, pages 758–763, 2015.

[33] Huan Liu and Hiroshi Motoda. *Feature extraction, construction and selection: A data mining perspective*, volume 453. Springer Science & Business Media, 1998.

[34] Sharad Malik and Lintao Zhang. Boolean satisfiability from theoretical hardness to practical success. *Communications of the ACM*, 52(8):76, aug 2009.

[35] Tânia Marques, Jorge Henriques, Paulo de Carvalho, Simão Paredes, Teresa Rocha, and João Morais. Personalization Based on Grouping Strategies for Short-Term Cardiovascular Event Risk Assessment. *Cardiovascular Engineering and Technology*, 6(3):392–399, 2015.

[36] João Martins, Raul Barbosa, Nuno Lourenço, Jacques Robin, and Henrique Madeira. Online verification through model checking of medical critical intelligent systems. In *50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops, DSN Workshops 2020, (to appear)*. IEEE, 2020.

[37] David A. Morrow, Elliott M. Antman, Andrew Charlesworth, Richard Cairns, Sabina A. Murphy, James A. de Lemos, Robert P. Giugliano, Carolyn H. McCabe, and Eugene Braunwald. TIMI Risk Score for ST-Elevation Myocardial Infarction: A Convenient, Bedside, Clinical Score for Risk Assessment at Presentation. *Circulation*, 102(17):2031–2037, oct 2000.

[38] Sebastian Nusser, Clemens Otte, Werner Hauptmann, and Rudolf Kruse. *Learning Verifiable Ensembles for Classification Problems with High Safety Requirements*, pages 405–431. IGI Global, 2010.

[39] W.L. Oberkampf and C.J. Roy. *Verification and Validation in Scientific Computing*. Cambridge University Press, 2010.

[40] European Society of Cardiology (ESC) Committee for Practice Guidelines (CPG). European guidelines on cardiovascular disease prevention in clinical practice: executive summary: Fourth Joint Task Force of the European Society of Cardiology and Other Societies on Cardiovascular Disease Prevention in Clinical Practice (Constituted by r. *European heart journal*, 28(19):2375–414, oct 2007.

[41] S. Paredes, T. Rocha, P. de Carvalho, J. Henriques, M. Harris, and J. Morais. Long term cardiovascular risk models' combination. *Computer Methods and Programs in Biomedicine*, 101(3):231–242, 2011.

[42] S. Paredes, T. Rocha, P. de Carvalho, J. Henriques, J. Morais, and J. Ferreira. Integration of Different Risk Assessment Tools to Improve Stratification of Patients with Coronary Artery Disease. *Medical and Biological Engineering and Computing*, 53(10):1069–1083, 2015.

[43] Simão Paredes, Teresa Rocha, Diana Mendes, Paulo Carvalho, Jorge Henriques, João Morais, Jorge Ferreira, and Miguel Mendes. New approaches for improving cardiovascular risk assessment. *Revista Portuguesa de Cardiologia*, 35(1):5–13, 2016.

[44] Andrea Saltelli, Stefano Tarantola, Francesca Campolongo, and Marco Ratto. *Sensitivity analysis in practice: a guide to assessing scientific models*, volume 1. Wiley Online Library, 2004.

[45] Luciano Sánchez and José Otero. Boosting fuzzy rules in classification problems under single-winner inference. *International Journal of Intelligent Systems*, 22(9):1021–1034, 2007.

[46] Robert E Schapire. The strength of weak learnability. *Machine Learning*, 5(2):197–227, jun 1990.

[47] Bernhard Scholkopf and Alexander J Smola. *Learning with kernels: support vector machines, regularization, optimization, and beyond.* MIT press, 2001.

[48] Johann Schumann, Pramod Gupta, and Yan Liu. Application of neural networks in high assurance systems: A survey. In *Applications of Neural Networks in High Assurance Systems*, 2010.

[49] Sanjit A Seshia, Dorsa Sadigh, and S Shankar Sastry. Towards verified artificial intelligence. *arXiv preprint arXiv:1606.08514*, 2016.

[50] Jean Souyris, Virginie Wiels, David Delmas, and Hervé Delseny. Formal Verification of Avionics Software Products. In Ana Cavalcanti and Dennis R Dams, editors, *FM 2009: Formal Methods*, pages 532–546, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[51] Margaret Tierney. Software engineering standards: the 'formal methods debate' in the uk. *Technology Analysis & Strategic Management*, 4(3):245–278, jan 1992.

[52] Willem Visser, Klaus Havelund, Guillaume Brat, SeungJoon Park, and Flavio Lerda. Model checking programs. *Automated software engineering*, 10(2):203–232, 2003.

[53] Ben Wagner. Ethics As An Escape From Regulation. From "Ethics-Washing" To Ethics-Shopping? *Being Profiled*, pages 84–89, 2019.

[54] David Wolpert. Stacked Generalization ( Stacking ). *Neural Networks*, 5:241–259, 1992.

[55] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A Comprehensive Survey on Graph Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–21, 2020.

[56] G.P. Zhang. Neural networks for classification: a survey. *IEEE Transactions on Systems, Man and Cybernetics, Part C (Applications and Reviews)*, 30(4):451–462, 2000.

# Appendices

This page is intentionally left blank.

# Appendix A

# DSML@DSN Paper

A pre-print of the submitted and accepted communication to the Dependable and Secure Machine Learning Workshop of the 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks begins in the following page.

2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)

# Online Verification through Model Checking of Medical Critical Intelligent Systems

João Martins
*Centre for Informatics and Systems*
*University of Coimbra*
Coimbra, Portugal
jpdmartins@student.dei.uc.pt

Raul Barbosa
*Centre for Informatics and Systems*
*University of Coimbra*
Coimbra, Portugal
rbarbosa@dei.uc.pt

Nuno Lourenço
*Centre for Informatics and Systems*
*University of Coimbra*
Coimbra, Portugal
naml@dei.uc.pt

Jacques Robin
*Centre for Research in Informatics*
*University of Paris 1 Panthéon-Sorbonne*
Paris, France
jacques.robin@univ-paris1.fr

Henrique Madeira
*Centre for Informatics and Systems*
*University of Coimbra*
Coimbra, Portugal
henrique@dei.uc.pt

***Abstract*—Software systems based on Artificial Intelligence (AI) and Machine Learning (ML) are being widely adopted in various scenarios, from online shopping to medical applications. When developing these systems, one needs to take into account that they should be verifiable to make sure that they are in accordance with their requirements. In this work we propose a framework to perform online verification of ML models, through the use of model checking. In order to validate the proposal, we apply it to the medical domain to help qualify medical risk. The results reveal that we can efficiently use the framework to determine if a patient is close to the multidimensional decision boundary of a risk score model. This is particularly relevant since patients in these circumstances are the ones more likely to be misclassified. As such, our framework can be used to help medical teams make better informed decisions.**

***Index Terms*—critical systems, intelligent systems, verification, model checking, medical risk scores**

## I. Introduction

Software systems are increasingly relying on Artificial Intelligence (AI) in general and Machine Learning (ML) in particular, in some cases applied to critical domains. Examples include the health and transportation domains, with applications ranging from diagnosis tools [14] to self-driving vehicles [9]. In such domains, verification, certification and explanation are fundamental.

One of the crucial steps in the development process of critical systems is *verification*, with the goal of assuring that the design and implementation of a system fulfill its requirements. Formal methods are a complementary approach to software engineering methodologies and development processes, suitable for rigorously describing and reasoning about complex systems [16]. Based on formal logic and mathematical notions, techniques such as formal proofs [15] and model checking [2] can be used to verify properties of critical systems.

A key challenge of AI and ML components is that often one cannot extract a specification directly from the machine learned models, nor is their training performed according to a verifiable specification. Some research efforts focus on logic-based algorithms that learn reasoning models from data, since their representation is suitable for extracting a specification [12]. Provided a specification, we are therefore able to apply formal methods to perform verification.

Along the several existing approaches that include perceptron techniques, instance based learning, support vector machines, probabilistic and logic-based models, the last two are among the favored choices in the medical field. Decision trees and sets of rules, which are examples of logic-based models, are preferred over other representations as these are regarded as explainable and interpretable [8]. Models for computing *risk scores*, used in medical risk assessment, are a good example and an interesting case-study, given their acceptance and validation within the medical community [1].

This paper addresses the usage of model checking, at runtime, to qualify medical risk scores with information concerning the proximity of a patient to the multidimensional decision boundaries. The proposed framework provides the medical staff with an evaluation of the risk score, as the classic approach, along with a binary indicator of proximity to a decision boundary, based on specific medical input features and ranges. In other words, our frameworks explicitly identifies whether a patient's risk assessment could be different upon small changes to some of the input risk factors variables. This feature increases trust on the calculated risk scores, because the framework brings to the user's attention the cases in which the categorical risk assessment should be questioned.

Next, Section II addresses relevant existing contributions within the scope of our work, Section III provides a detailed explanation of the proposed framework, Section IV and V lay the experimentation, results and discussion done to the framework and Section VI presents the main conclusions.

978-1-7281-7263-7/20/$31.00 ©2020 IEEE
DOI 10.1109/DSN-W50199.2020.00015

## II. Related Work

In this section we present relevant work already conducted in medical software certification, risk scores, verification strategies of intelligent systems and model checking. All these research fields lay bed to our proposed framework.

### A. Medical software

Critical medical software must follow safety standards concerning lifecycle management (IEC 62304), risk assessment (ISO 14971) and usability (IEC 62366). However, those standards are based on assumptions that are typically not verified by AI. One of such assumptions is that software is implemented manually by human programmers, specifying at design-time the whole control and data flows of the software. Another one is that this implementation can be tested extensively by running test sets also manually programmed and that verify properties of these control and data flows.

An AI component, in contrast, is often implemented by a persistent declarative knowledge base, that is specific to the medical software to be developed, but it is interpreted by a generic inference that is completely independent of the medical software at hand. It is the run-time interaction between (a) the volatile input data given to the AI to reason about, (b) its declarative persistent knowledge base and (c) its application-independent inference engine that defines control and data flow. Also, the inference engine often performs heuristic and/or non-deterministic approximate search of a very large combinatorial space. This makes extensive design-time and manual writing of tests for such AI unpractical.

This difficulty is made worse by the fact that nowadays part if not all the knowledge base is acquired from data by machine learning instead of manually declared. This learning process includes steps of data selection, transformation, choice of learning algorithms, parameter tuning of these algorithms, learning hypothesis, search space *a priori* pruning, all of which introduce biases that can completely change the resulting knowledge base and hence the reasoning of the AI component. All these questions are only starting to be explored by the ISO/IEC task force JTC-1/SC-42.

### B. Medical automated assessment of admitted patients (Risk scores)

Medical Condition Risk Scores (MCRS) were introduced in medical emergencies as tools to help prioritise and differentiate patients treatment, providing the needed balance between saving people's lives while saving on health costs through application of the right treatment.

The MCRS are based on large scale longitudinal medical research studies. They are, however, easy to implement as decision-support software since they generally apply simple weighted combinations of indicators such as blood pressure or known history of a condition. The output is the sum of the weighted values which falls under a risk category.

The various limitations of state-of-the-art MCRS were studied together with attempts to overcome them in two ways [13]: (a) composing several of them or (b) selecting the best of several for a given target population which phenotype may differ significantly from the cohorts used in the studies from which each of the considered MCRS was derived.

### C. Verification strategies for intelligent systems

One might achieve verification through explanation. That is why [11] identifies logic-based models as interpretable and therefore verifiable through inspection. Model complexity is an attribute that puts not only manual inspection in check, but also challenges both manual inspection and global automated verification. [10] is an example of the efforts put into making a complex black-box model locally explainable. Our work falls within this trend, applying a formal method to locally check the vicinity of the risk factors space for a given patient.

### D. Model checking

Model Checking (MC) is a formal method used to verify hardware and software systems [4]. It consists in building a formal model of what the system should do and then automatically verify properties of the model by state-space search techniques detecting the reachability of an error state.

MC has historical relevance in critical systems development typically following a waterfall process, meaning that it was a verification step before implementation. Nowadays, with new development processes, one can see source code as a specification itself [4]. The argument is that code is a static representation that has yet to be compiled, converted into binary and executed, so it is actually valid to see it as a requirements specification.

Model properties are often specified in a temporal logic language [3], with primitives like *eventually* and *always*, and can be used to avoid for instance deadlocks, contradictions or confirm correctness of the system output.

Many model checkers are composed of two main components [17]: (a) an executor that, for a model and entry-states, outputs reachable states and (b) a verifier that, for a given specification and state along the state exploration, checks if the specification is satisfied, returning the state and explored path as a counter-example when the specification fails.

## III. Methodology

We propose a framework for verifying the evaluation made by an intelligent MCRS. It determines whether a risk assessment falls within a gray area of the decision space. As shown in Fig. 1, our approach feeds a model checker with both the patient data passed as input to the MCRS and the risk assessment output by the MCRS. This output is used as an assertion property to be verified by the model checker, which will search in the vicinity of the patient input risk factors space for different assessment outputs. If the model checker does not identify counter examples to the baseline assertion, the framework outputs a confident classification indication.

To implement our approach we started by programming the MCRS in Python for fast prototyping. We then manually translated the logic of the Python program together with assumptions and search ranges into a semantically equivalent

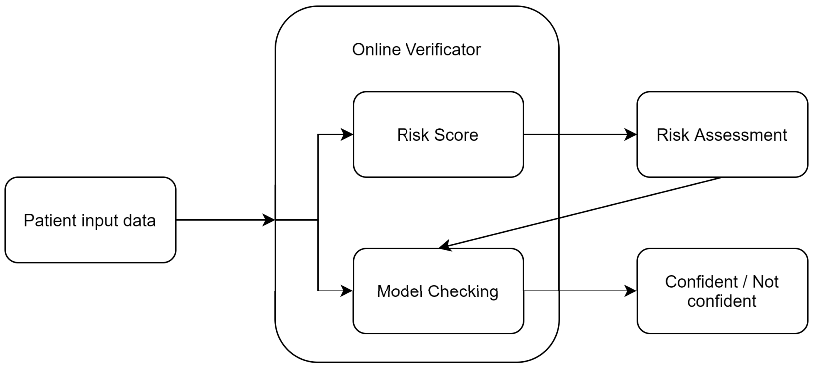


Fig. 1: High level overview of the proposed framework

Promela modeling language accepted as input by the popular model checker SPIN. We are then able to perform online verification through the model checker, for each new patient data, completing the framework pipeline. We explain each of the steps in detail in the following subsections.

## A. Implementation

Listing 1 shows the implementation of a MCRS assessment function, considering a single risk factor. We can add further risk factors, following a similar approach. The risk assessment rules are implemented by conditional expressions and statements. *Age* is a common risk factor among risk scores and often includes several ranges of discrimination of the risk, so it is a good example that we shall use across the explanation of our approach. For a given patient, if the accrued risk sum falls above a pre-determined threshold, it outputs that the patient is in risk.

Listing 1: Implementation of a risk score in Python

```python
def grace(patient_data, risk_threshold):

    risk=0

    age = patient_data['Age']
    if age >= 40 and age <= 49:
        risk += 15
    elif age >= 50 and age <= 59:
        risk += 29
    #Other age ranges...
    elif age >= 90:
        risk += 80

    #Other risk factors...

    if risk >= risk_threshold:
        patient_in_risk=True
    else:
        patient_in_risk=False

    return patient_in_risk
```

## B. Translation

At this stage we have an executable MCRS assessment function. We now need to translate the code onto a specification language.

A few considerations must be taken into account for this step. Since Python has no assigned types, one needs to decide the necessary allocation space for the variables used in the code. Usually, type *int* is suitable for the majority of the risk factors, since it can both store categorical and numerical variables. Also, conditional statements are blocking-state in Promela, so one needs to add an *else* dummy state for each conditional statement in the specification model.

Listing 2 presents the specification for the implemented risk score in the previous subsection. Besides the straight forward translation, where we just added the assigned types and *else* states, the specification considers three extra elements: (a) *AGE* and *RISK* are constants that specify the input parameters concerning the patient risk assessment, (b) the *assert* statement set the property we want to verify and (c) a *select* command specifies the search space for the risk factor.

Listing 2: Specification of a risk score using Promela

```
#define AGE
#define RISK

active proctype Grace() {
    int age, risk;
    bool patient_in_risk;

    select(age : (AGE-3) .. (AGE+3) );
    risk=0;

    if
    :: age >= 40 && age <= 49 -> risk = risk + 15
    :: age >= 50 && age <= 59 -> risk = risk + 29
    //other age ranges...
    :: age >= 90 -> risk = risk + 80
    :: else -> skip
    fi;

    //other risk factors...

    if
    :: risk >= 145 -> patient_in_risk = true
    :: else -> patient_in_risk = false
    fi;

    assert(patient_in_risk == RISK)
}
```

### C. Search space parameters

One needs to specify the bounds on which each input variable (risk factor) is going to be verified for the assumption of falling within the same risk assessment. As in Listing 2, by using constants to specify risk factors like *AGE*, we can limit the verification space through the *select* command, creating an upper and lower interval around the patient risk factor.

An important consideration is related to which risk factors we want to vary in our verification process. We do not perform ranging for the risk factors that are categorical since different values for those features represent patients that are too far apart, considering the notion of vicinity of medical risk factors search space.

### D. Online verification

Each time a risk assessment is performed, one uses the implemented MCRS assessment function to obtain a classification. We then provide the model checking tool with the specification, previously translated, which is set according the input data of the patient and the risk assessment output. We use Spin as our model checking tool. Through the execution of the tool we check for counter examples of the expected risk assessment output within the risk factors search space bounded by the range parameters. If no counter example is found, we provide the user with an indication of confidence on the risk assessment.

## IV. Experimental Evaluation

In this section we present relevant experimentation of our framework. We focused on implementing the GRACE risk score, translating it into a specification and applying the framework to a pool of patients, extracting relevant metrics.

### A. GRACE

Global Registry of Acute Coronary Events (GRACE) is an international database that allowed for the supervised learning of a risk score model [6], [7], based on eight risk factors listed in Fig. 2. GRACE risk model was built for short term assessment, based on events of death or myocardial infarction (heart attack) on patients with coronary artery disease. GRACE risk score is widely accepted and was subject to many validation studies, identifying strengths and weaknesses [1], [5].

The novelty of our work arises from locally identifying regions where the risk assessment may vary for small patient risk factors changes.

### B. Data

A collaboration with a medical team which applies the GRACE risk score allowed us to gather medical records from patients data and test and validate our framework.

The dataset used is composed of 460 patients admitted at Santa Cruz Hospital (Oeiras, Portugal) between March 1999 and July 2001, with the specific condition of acute coronary syndrome with non-ST segment elevation - ACS-NSTEMI [13].

TABLE I: Range variation parameters used in the GRACE model specification

| **Risk factor** | **Range variation** |
|---|---|
| Age | +/- 4 (years) |
| Heart rate | +/- 5 (beats/min) |
| Systolic blood pressure | +/- 3 (mm Hg) |
| Creatinine | +/- 0.1 (mg/dL) |

Within the dataset we have a positive event rate of 7.2%, comprising 33 events of death or myocardial infarction after 30 days of admission. We directly map those events with our target class, the risk indication.

### C. Range parameters

Table I details the considered ranges for each of the varied risk factors. For a given patient (data entry),the model checker will verify the vicinity of the risk factors space, according to the defined bounds.

Since the remaining risk factors are categorical, we decide not to vary them through the model checker analysis as that could lead to generating a verification space too far apart from each patient data. We recognise that this choice might limit the application of the proposed framework to simpler risk scores that only use categorical risk factors. Nevertheless, the idea of our framework is to provide local verification of complex models, meaning that for the simpler ones we can use an approach of global verification.

## V. Results and discussion

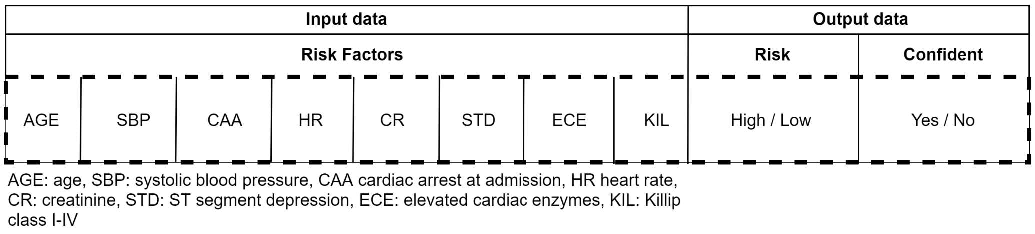
Experimental evaluation of the framework was done on a Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz machine. Relevant results are presented in Table II and Spin execution statistics are shown in Table III.

Our framework considers the patients in two groups. The first one (Group 1 - G1) contains the patients that were identified as being far from the decision boundary. The second one (Group 2 - G2) contains the patients that are very close to the decision boundary, which are the ones that are prone to misclassification. This division allows for our confidence metric to be a binary output for each patient, as depicted in Fig.2.

The majority of the patients fall within G1, i.e., the risk assessment falls within an unambiguous classification, given our set of proximity parameters. This is an important result that confirms the relevance of the proposed framework. If G2 was to have more patients than Group 1, medical teams could easily disregard the application of the risk score and our framework.

The risk assessment accuracy is improved for G1 and degrades G2, when comparing with the GRACE overall application. This justifies the relevance of our framework, as patients that fall close to the decision boundaries tend to be misclassified. Results under the F1 measure, the weighted average of precision and recall, provide further evidence of the clear distinction within both groups.

Looking at the sensitivity and specificity metrics it is possible to see that the GRACE model tends to classify as high



AGE: age, SBP: systolic blood pressure, CAA cardiac arrest at admission, HR heart rate, CR: creatinine, STD: ST segment depression, ECE: elevated cardiac enzymes, KIL: Killip class I-IV

Fig. 2: Characterisation of data entries and our framework outputs.

TABLE II: Results from the overall GRACE risk score and for the presented framework.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| **GRACE overall** | | | | | | | | | |
| Accuracy | 0.5522 | F1 Measure | 0.2077 | Sensitivity | 0.7941 | Specificity | 0.5329 | Patients | 460 |
| **GRACE with online verification** | | | | | | | | | |
| **Group 1 - Confident assessed** | | | | | | | | | |
| Accuracy | 0.5730 | F1 Measure | 0.2336 | Sensitivity | 0.8065 | Specificity | 0.5524 | Patients | 384 |
| **Group 2 - Not confident assessed** | | | | | | | | | |
| Accuracy | 0.4605 | F1 Measure | 0.0889 | Sensitivity | 1.0000 | Specificity | 0.4459 | Patients | 76 |

TABLE III: Spin execution statistics: number of states stored, memory usage and execution time.

| | | | |
|---|---|---|---|
| **States** | | | |
| Average | 61036 | Max | 73012 |
| **Memory (Megabytes)** | | | |
| Average | 131.50 | Max | 132.05 |
| **Time (seconds)** | | | |
| Average | 0.03 | Max | 0.05 |

risk (positive event) in the vicinity of the decision boundary. This will increase the value of the sensitivity to its maximum, but will ruin the specificity score. This is interesting evidence that can be taken into account when setting the thresholds of risk discrimination.

The model checker execution time, memory used and states explored show that the framework can be used together with a medical software tool. We know that time is an important factor for medical teams, since they need to make decisions and analysis in a timely manner. Thus, and taking into account the results presented in Table. III, it is possible to see that the usage of a model checker does not increase significantly the execution time. Another important result is concerned with the fact that we only explore a small portion of the state space, which allows for the application of the framework to larger and more complex models.

These results also help validate why the GRACE risk score is widely used.

## VI. Conclusion

This paper applies the formal verification technique of model checking at runtime to increase the confidence in the application of Medical Condition Risk Scores. The proposed technique consists in verifying whether an input to the MCRS model is in proximity to its multidimensional decision boundary. This principle increases trust in the risk score tool and identifies cases in which categorical risk assessment should be further examined.

With our approach we perform a tailored analysis of the decision space. Common approaches disregard problem information that is relevant for that analysis. For instance, medics are not interested on a distance verification that is based on all the features (risk factors), because some of them are binary or categorical and patients with different values for those features are not considered ”close” from a medical perspective.

The verification complexity of the model checker is sufficiently small to allow for online verification results to be produced efficiently. This is, in part, a consequence of the model checking technique itself, which expands the graph of reachable states and is therefore more efficient than both testing and sensitivity analysis.

The paper describes the practical implementation of the proposed framework, starting with a Python model for risk score computation that is translated into the Promela language. The Promela model examines input values in close proximity to the input point (the actual patient case) and the verifier generated by Spin may be executed during runtime, providing a form of online verification. Using this principle it is possible to determine if a case is close to a boundary and, to some extent, explain for which reason that occurs. The expected impact for practice is a greater ability to achieve online verification of machine learning models.

Looking forward, it might be pertinent to partially automate the specification translation step, for instance by adding a compiler component to the framework, in order to deal with scalability challenges that may arise in more complex models.

## Acknowledgements

This work is partially funded by Portuguese Foundation for Science and Technology, I.P., within the project CISUC - UID/CEC/00326/2020, by European Social Fund, through the Regional Operational Program Centro 2020 and by the project AI4EU – “A European AI On Demand Platform and Ecosystem”, project funded by the European Commission, H2020-EU.2.1.1., Grant agreement ID: 825619.

## References

[1] S. Akyuz, S. Yazici, E. Bozbeyoglu, T. Onuk, O. Yildirimturk, D. Karacimen, M. I. Hayiroglu, G. Erdogan, A. O. Oner, A. N. Calik, M. Cagdas, and N. Cam, "Validity of the updated GRACE risk predictor (version 2.0) in patients with non-ST-elevation acute coronary syndrome," *Revista Portuguesa de Cardiologia*, vol. 35, no. 1, pp. 25–31, jan 2016. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0870255115003200

[2] S. Chaki and A. Gurfinkel, *BDD-Based Symbolic Model Checking*. Cham: Springer International Publishing, 2018, pp. 219–245. [Online]. Available: https://doi.org/10.1007/978-3-319-10575-8_8

[3] E. M. Clarke, E. A. Emerson, and A. P. Sistla, "Automatic verification of finite-state concurrent systems using temporal logic specifications," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 8, no. 2, pp. 244–263, 1986.

[4] E. M. Clarke Jr, O. Grumberg, D. Kroening, D. Peled, and H. Veith, *Model checking*. MIT press, 2018.

[5] B. Elbarouni, S. G. Goodman, R. T. Yan, R. C. Welsh, J. M. Kornder, J. P. DeYoung, G. C. Wong, B. Rose, F. R. Grondin, R. Gallo, M. Tan, A. Casanova, K. A. Eagle, and A. T. Yan, "Validation of the Global Registry of Acute Coronary Event (GRACE) risk score for in-hospital mortality in patients with acute coronary syndrome in Canada," *American Heart Journal*, vol. 158, no. 3, pp. 392–399, 2009.

[6] K. A. Fox, O. H. Dabbous, R. J. Goldberg, K. S. Pieper, K. A. Eagle, F. Van De Werf, Á. Avezum, S. G. Goodman, M. D. Flather, F. A. Anderson, and C. B. Granger, "Prediction of risk of death and myocardial infarction in the six months after presentation with acute coronary syndrome: Prospective multinational observational study (GRACE)," *British Medical Journal*, vol. 333, no. 7578, pp. 1091–1094, 2006.

[7] K. A. Fox, G. FitzGerald, E. Puymirat, W. Huang, K. Carruthers, T. Simon, P. Coste, J. Monsegu, P. G. Steg, N. Danchin, and F. Anderson, "Should patients with acute coronary disease be stratified for management according to their risk? Derivation, external validation and outcomes using the updated GRACE risk score," *BMJ Open*, vol. 4, no. 2, pp. 1–10, 2014.

[8] L. H. Gilpin, D. Bau, B. Z. Yuan, A. Bajwa, M. Specter, and L. Kagal, "Explaining explanations: An overview of interpretability of machine learning," *Proceedings - 2018 IEEE 5th International Conference on Data Science and Advanced Analytics, DSAA 2018*, pp. 80–89, 2019.

[9] I. Goodfellow, P. McDaniel, and N. Papernot, "Making machine learning robust against adversarial inputs," *Communications of the ACM*, vol. 61, no. 7, pp. 56–66, 2018.

[10] R. Guidotti, A. Monreale, S. Ruggieri, D. Pedreschi, F. Turini, and F. Giannotti, "Local rule-based explanations of black box decision systems," 2018.

[11] R. Guidotti, A. Monreale, S. Ruggieri, F. Turini, F. Giannotti, and D. Pedreschi, "A survey of methods for explaining black box models," *ACM Computing Surveys*, vol. 51, no. 5, pp. 1–45, 2018.

[12] G. J. Holzmann and M. H. Smith, "Software model checking: extracting verification models from source code," *Software Testing, Verification and Reliability*, vol. 11, no. 2, pp. 65–79, jun 2001. [Online]. Available: http://doi.wiley.com/10.1002/stvr.228

[13] S. Paredes, T. Rocha, P. de Carvalho, J. Henriques, J. Morais, and J. Ferreira, "Integration of Different Risk Assessment Tools to Improve Stratification of Patients with Coronary Artery Disease," *Medical and Biological Engineering and Computing*, vol. 53, no. 10, pp. 1069–1083, 2015.

[14] S. Paredes, T. Rocha, D. Mendes, P. Carvalho, J. Henriques, J. Morais, J. Ferreira, and M. Mendes, "New approaches for improving cardiovascular risk assessment," *Revista Portuguesa de Cardiologia*, vol. 35, no. 1, pp. 5–13, 2016.

[15] J. Souyris, V. Wiels, D. Delmas, and H. Delseny, "Formal Verification of Avionics Software Products," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2009, vol. 5850 LNCS, pp. 532–546. [Online]. Available: http://link.springer.com/10.1007/978-3-642-05089-3_34

[16] M. Tierney, "Software engineering standards: the 'formal methods debate' in the uk," *Technology Analysis & Strategic Management*, vol. 4, no. 3, pp. 245–278, jan 1992. [Online]. Available: http://www.tandfonline.com/doi/abs/10.1080/09537329208524097

[17] W. Visser, K. Havelund, G. Brat, S. Park, and F. Lerda, "Model checking programs," *Automated software engineering*, vol. 10, no. 2, pp. 203–232, 2003.

This page is intentionally left blank.

# Appendix B

# Docker container

All the code and specification is made available in the form of a Docker container. It includes the latest versions of *Spin* and *Python* with the necessary libraries (*pandas, numpy* and *sklearn*), allowing for experimentation of the whole verification framework and replication of the results that do not depend on the patients database.

A Docker[1] container encapsulates all the dependencies an application, package or a portion of source code need to run or be executed, allowing portability and compatibility throughout different operative systems environments. Any host with Docker installed can run a Docker container.

The latest version of the Docker container of this work can be downloaded from:
`https://hub.docker.com/r/jpdmartins/verifiable-ai` (last accessed on June 29th)
and all the source code repository is maintained in:
`https://github.com/joaopdmartins/Verifiable-AI` (last accessed on June 29th)

All the code developed along this work is subject to a BSD-3-Clause Open Source Licence[2] as disclaimed in the code repository.

A terminal can also be directly used to obtain an image of the Docker container and run it with the following commands:

```
#!/bin/bash

docker pull jpdmartins/verifiable-ai
docker run -it jpdmartins/verifiable-ai
```

Do not forget to use the *-it* flag in order to keep *STDIN* open and the terminal attached to the container.

Access to the patients database used is not provided but can be obtained under a Non Disclosure Agreement. Nevertheless, a synthetic patient is included as example of the input format expected by the system and allows for testing of the code that requires such input.

[1]https://www.docker.com/why-docker
[2]https://opensource.org/licenses/BSD-3-Clause