UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTMENTO DE CIENCIAS DE LA COMPUTACIÓN

INTERPRETABLE METHOD FOR GENERAL CLASSIFICATION USING DEMPSTER-SHAFER THEORY

TESIS PARA OPTAR AL GRADO DE MAGÍSTER EN CIENCIAS, MENCIÓN COMPUTACIÓN

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL EN COMPUTACIÓN

SERGIO IGNACIO PEÑAFIEL AGUILERA

PROFESOR GUÍA:
NELSON BALOIAN TATARYAN

MIEMBROS DE LA COMISIÓN:
ALEJANDRO HEVIA ANGULO
GONZALO NAVARRO BADINO
MARCELO MENDOZA ROCHA

Este trabajo ha sido parcialmente financiado por Conicyt (Chile) N^o 22180506

SANTIAGO DE CHILE
2020

Resumen

Los modelos de aprendizaje supervisado han logrado resolver problemas con gran precisión en varias ámbitos como la medicina, el transporte y el financiero. Sin embargo, la mayoría de estos métodos se comportan como cajas negras para los usuarios y no explican las decisiones que realizan. Esto genera una desconfianza en los modelos y en muchas ocasiones no pueden ser utilizados porque pueden generar sesgo y discriminación en la predicción. Experimentalmente, se observa que los métodos con mayor interpretabilidad como árboles de decisión, son los que tienen errores en la predicción más altos.

En esta tesis, se plantea un nuevo modelo de clasificación interpretable basado en la Teoría de Dempster Shafer (TDS) y el método de descenso de gradiente (DG). Se espera que el modelo sea tan interpretable como los Árboles de Decisión y a la vez tenga una precisión comparable a los métodos tradicionales. El modelo está basado en reglas, las cuales consisten en afirmaciones que pueden ser entendidas con facilidad y permiten trabajar con datos incompletos o inciertos. Mediante un conjunto de datos de entrenamiento y DG los valores de estas reglas son optimizados para obtener el modelo óptimo. Una vez que el modelo está entrenado, los valores de las reglas nos indican que tan importantes son para la predicción de cada clase, y por lo tanto, se puede extraer conocimiento interpretable directamente del modelo.

El modelo fue probado en escenarios controlados y conjuntos de datos tradicionales para verificar que pudiera resolver problemas de clasificación simples. El modelo mostró ser capaz de resolver todos los problemas de forma correcta. Además, al ser comparado con otros métodos de clasificación se puede observar que alcanza valores de precisión similares.

Finalmente, el modelo fue probado en un problema real de predicción de riesgo de tener un ataque cerebrovascular (ACV). Se cuenta con datos de más de 27 mil pacientes japoneses provenientes de chequeos médicos anuales. Entre los datos se tienen resultados de exámenes, datos demográficos e historial de enfermedades. El modelo obtuvo un área bajo la curva Característica Operativa del Receptor (ROC) de 87.5% superando a otros métodos de clasificación y a otros métodos de detección de ACV.

Además, se lograron extraer las reglas más influyentes para la predicción de ACV. Estos resultados fueron validados con otros métodos de interpretabilidad, con literatura médica y con una encuesta a expertos médicos. La mayoría de las reglas que nuestro modelo encontró coinciden con el conocimiento experto real del área de la aplicación.

Abstract

Supervised learning models have been able to solve classification problems with high accuracy in several fields such as health, transport, and finance. However, most of these methods behave like black-boxes, and they do not explain the decision made. This generates a lack of trust in models, and in many situations, they cannot be used because they may produce biased results or discrimination. Empirically, high interpretable methods, like Decision Trees, are the ones with high error rate in prediction.

In this thesis, we propose a new interpretable classification model based on Dempster-Shafer Theory (DST) and Gradient Descent (GD). The model is expected to be as interpretable as Decision Trees but achieving accuracy scores similar to other models such as K-Nearest Neighbors or Support Vector Machines. The proposed model is rule-based, in which rules represent meaningful statements and can handle uncertain or incomplete data by applying the procedures of DST. Using a data training set and GD, the parameters of these rules are optimized to find the optimum model. Once the model is trained, the resulting rule values indicate their importance for the prediction of a class. Therefore, we can extract interpretable knowledge directly from the model.

The model was tested on controlled scenarios and traditional datasets to verify it can handle simple classification tasks correctly. The results show that the model was able to perform accurately classification in all instances. Besides, the model was compared to other classification methods, achieving comparable accuracy scores.

Finally, the model was tested on a real problem of prediction of stroke risk of a population. Data from more than 27 thousand of Japanese patients from their annual medical checkups was used. The information has exam results, patient demographics, and disease history. The model reaches a value of 87.5% for the area under the Receiver Operating Characteristic (ROC) curve, outperforming other classification models and stroke screening methods.

Moreover, we extracted the most contributory rules for stroke prediction. These results were validated using explanation models, medical literature, and an expert survey. Most of the rules obtained by our model coincide with medical findings and expert knowledge.

A mi familia: Cyntia, Juan, Rodrigo y Juan Pablo.

Agradecimientos

En primer lugar quiero agradecer a mi madre, Cyntia Aguilera, quien me crió, me dio cariño, siempre me tuvo paciencia y me enseñó todo para ser quien soy ahora. Ella es una de las personas que más admiro y siempre ha sido la mejor mamá.

También agradecer a mi padre, Juan Peñafiel, quien me ha apoyado siempre y confió en mí para seguir mis estudios superiores aun cuando eso significaba estar lejos. Valoro mucho su trabajo y preocupación para que siempre estuviéramos bien.

También agradecer a mis hermanos, Rodrigo y Juan Pablo, quienes siempre me dan muchos momentos de alegría, compartimos muchas experiencias juntos y siempre me hacen sentir como en casa cada vez que los visito.

También agradecer a mi tía Any, tío Carlos, y las mellis Fer y Flo, quienes me recibieron en su hogar para poder seguir mis estudios. Aprecio mucho la oportunidad que me dieron y todo su apoyo en esta etapa. Agradecer también a todo el resto de mi familia por su apoyo.

Un muy especial agradecimiento es para mi profesor guía, Nelson Baloian, el cual desde muy temprano en mi paso por la universidad me invitó a participar en docencia e investigación. También por permitirme realizar pasantías de investigación en el extranjero y mostrarme otras culturas.

También mencionar y agradecer Los Cracks: Joaquín, Beli, Pelao, Americo, Gabriel, Huan, Jaev y Rodrigo, por todos los buenos momentos que compartimos en el DCC, en las juntas y hasta los viajes que realizamos.

Un especial agradecimiento también a Joaquin y Beli con quienes hemos estado trabajando más cercanamente gracias a Create, y espero que siga siendo así en el futuro.

Agradecer también a Álvaro Riquelme, quien me ayudó bastante con la parte clínica de mi tesis y con quien hemos compartido buenos momentos.

También agradecer a los miembros de mi comisión, Gonzalo Navarro, Alejandro Hevia y Marcelo Mendoza por revisar mi trabajo de tesis y ayudarme en que sea mejor. Además agradecer al profesor Jose A. Pino quien también me ayudó a mejorar mi trabajo.

Finalmente quiero agradecer al amor de mi vida, Daniela Carrillo, quien fue la persona más importante por este paso en la universidad, en el cual los dos crecimos. La amo mucho.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

The development of methods for classifying based on machine learning techniques has been one of today's most rapidly growing technical fields in artificial intelligence in the last years [1].

Two fundamental concepts associated with classifying instruments are performance (also called accuracy for simplicity) and interpretability. Model performance is related to how good a classification method works. This concept has several dimensions, and accordingly, several indicators have been proposed, which are widely used today. Some of these indicators are accuracy, sensitivity or recall, specificity or precision, the area under the ROC curve, and F1 score [2]. All of them measure the proportion of correct and incorrect classification performed by a certain model or the error obtained in the classification process.

The concept of interpretability relates to the possibility a human user has for tracking the decisions made by the model to classify a sample in one or another class. Interpretability is also called explanability or understandability in literature. Interpretable models are generally more valuable than regular classification models, since they produce explications along with predictions.

Although both characteristics are desirable in a classification instrument, experience has shown that they collide, i.e., we often find limited or non-existent interpretability in most accurate methods and vice versa. As an example, Deep Learning methods based on multi-level Neural Networks are considered black-box algorithms. On the other hand, there are highly interpretable methods like Decision Trees, linear models, or rule-based models. However, these latter methods have restrictions that do not allow them to achieve high accuracy; for example, linear models cannot learn non-linear relationships between attributes. Rule-based models and Decision Trees are constrained to have a reduced number of rules or depth in the tree. Otherwise, the model increases its complexity and makes it less interpretable.

Sometimes the interpretability of results given by the method may be more important

than high accuracy in the prediction. As an example, consider a physician who is trying to diagnose a patient's disease based on his/her symptoms. For the physician, it is more important to know which symptoms are related to a particular illness instead of having a machine predicting the disease without any explanation [3] even in case the machine predictions are very accurate. Giving these explanations makes the users trust the system, and thus generating a helpful tool to support decision-making. Moreover, in many situations, interpretability goes beyond the trustworthiness of models because explaining decisions is a required aspect for its application. For instance, Chilean legislation on acceptance of customers' mortgage credit applications prohibits financial institutions from using certain variables to make their decision, such as ethnic group and skin color. A client whose loan application was declined may appeal to these institutions, and they must explain the cause of the credit refusal. In order to satisfy this requirement, these institutions are forced to use interpretable methods even if other models can predict better profitable potential clients; otherwise, they cannot give the previously mentioned explanation.

The exact definition of interpretability can vary depending on the context in which the model is applied. For instance, for fuzzy rule-based systems or traditional rule-based systems, interpretability is highly related to the number of rules used and the complexity of them [4]. For black-box, machine learning techniques, and particularly neural network methods, interpretability refers to the ability of the model for explaining every classified instance. These two definitions for interpretability have drawbacks that difficult its understandability.

For the case of rule-based systems, applying the rules must result in each record fitting into a class. This constraint may imply the development of complicated rules, and thus, a less interpretable model [5]. Another drawback is the lack of a standard procedure to deal with the problem of missing values. If a rule relates to an attribute that is missing, there is no clear way to continue with the classification, e.g., taking the positive or negative branch or switch to another rule. If the rule concerns many attributes and just one is missing, the decision is still unclear. Missing values is a common problem when working with real data.

For the case of instance explanation of black-box methods, authors have various definitions. Some of them only distinguish between useful attributes and non-useful attributes [6], and other explanation methods give a kind of rules or human-readable statements for that instance. However, the drawback is the fact that explanations change for each instance, e.g., for a record, an attribute X could be necessary for classification, but, for another record, the same attribute X could be useless. This change does not allow giving a clear explanation of the whole model. Thus, the model is not interpretable since the user cannot understand the decision of a completely new record without tracking the entire classification process again.

This work proposes a new meaning of interpretability, which is having simple rules concerning all possible attributes of a sample and then discovering which rules are essential to the process of deciding to which class a sample belongs. In other words, each rule has a weight indicating how important that rule is to determine whether or not a sample belongs to a particular class. This property defines an interpretability score for every rule instead of for a specific record or the whole model like previous approaches. Eventually, the model can drop the lightest rules to have fewer rules, thus obtaining a simpler model if needed.

One advantage of this approach, which existing ones cannot handle, is that rules are inde-

pendent of each other. This property means that every single rule is simple, understandable and applies evenly to all data records without needing to make any previous assumption. Another advantage is that the model allows the user to test custom rules. This ability is a desirable feature for developing expert knowledge because it helps to validate a hypothesis and measure its importance to explain the studied system.

This work presents the development of a classification method that can be applied to a wide range of decision-making scenarios. The proposed model achieves accuracy comparable to those based on Artificial Neural Networks. At the same time, the model is highly interpretable according to the definition we introduced above, thus inheriting all the mentioned advantages. Besides, this method allows classifying samples for which attributes values may be missing.

The key idea to achieve this goal is to develop a rule-based classifier based on the Dempster-Shafer Theory [7]. This theory is a generalization of Bayesian theory allowing to assign uncertainty directly. Mass assignment functions (MAF) are the elements that encode knowledge in this theory which, given the presence of a particular fact on a decision scenario (e.g., the level of humidity) assigns a weight or probability for all subsets of the possible outcomes (heavy rain, light rain, no rain). The theory also defines the Dempster Rule [8], which indicates how to combine different MAFs (e.g., temperature, air pressure, and humidity) for obtaining a single result (probability of heavy, light or no rain). The use of uncertainty allows us to develop a more sophisticated rule-based model that can express complex scenarios without losing the simplicity of rules.

The model developed in this work includes a novel way to optimize mass assignment functions based on gradient descent techniques from the evidence provided by the training data. Moreover, we show how to extract explanations for classifications once the model is trained. This process is done by generating and optimizing meaningful rules automatically using the training data, which makes the model interpretable.

1.2 Hypothesis

We hypothesize that it is possible to solve the classification problems using a rule-based method based on Dempster-Shafer Theory, with an accuracy comparable to classical machine learning classifiers, such as Support Vector Machines, Random Forest and Multilayer Perceptrons. Besides, the method will be able to give meaningful explanations for the decisions made.

1.3 Objective

1.3.1 General Objective

The main goal of this work is to develop a new machine learning technique based on the Dempster Shafer Theory that is as general as current solutions, such as SVM, Random

Forest, and ANNs, and it can solve classification tasks with tabular data. The method also provides a mechanism to evaluate the importance of defined rules to give explainable results for the predicted outcomes. Applying and testing the proposed model to the stroke risk prediction problem, evaluating both classifier performance and interpretability, is also part of the goal of this work.

1.3.2 Specific Objectives

1. Develop a module to operate with Dempster Shafer Theory elements such as frame of discernment, mass assignment functions, combination rules, belief, and plausibility computation.
2. Build a classifier that, given a rule set, predicts outcomes by applying the combination rule and selecting the class with maximum belief.
3. Create a method that automatically generates a first collection of rules using a specific strategy, for example, independent variables rules, pair-dependency, weighted variable contribution, as well as custom defined rules. To prevent overfitting, the model may drop the rules with low support (i.e., only a few instances satisfy the rule statement).
4. Implement an algorithm to automatically fit the mass assignment function values for each rule according to the data and the error produced by the outcomes.
5. Apply the model to the stroke risk prediction problem. Report performance metrics such as accuracy, sensitivity, and area under the ROC curve. Extract the most contributory rules for stroke risk.
6. Compare the obtained results for stroke risk prediction problem with other methods such as Support Vector Machines (SVM), Random Forest, and Artificial Neural Networks (ANN).
7. Compare interpretability results with the result of explanation models and by surveying medical experts.

1.4 Methodology

This section explains the proposed methodology for the work.

1. Check the current literature about interpretable methods for classification and methods for extracting interpretable results from classifiers.
2. Check literature about Dempster Shafer Theory, and Dempster Shafer applications in machine learning.
3. Develop a module that implements the necessary elements of the Dempster Shaffer elements. It includes the frame of discernment, domains, mass assignment functions, belief, plausibility, and Dempster rule. The theory defines these elements as encoders for knowledge. The theory also defines indicators for these elements and the combination between them. To validate the module, we will create unit tests based on well-known examples of Dempster Shafer Theory.

4. Implement a classifier that allows adding rules about a certain domain as a combination of a predicate and a mass assignment function.
5. Implement the rule lookup for an entry, the application of the Dempster rule to selected rules, and the belief computation to predict the outcome.
6. Implement helper functions to generate collections of rules, using different strategies. It is desirable to have a system to encode and decode rules from files.
7. Implement an algorithm to fit the mass assignment function values (i.e., masses) according to data. The algorithm should be parameterized to ensure it is applicable to different data sets.
8. Implement a method to show the most important rules that characterize an outcome.
9. Test the model with simple datasets, using custom rules and automatic generated rules. Compare the results with other classifiers.
10. Test the model with the electronic health records data (EHR) for the stroke risk prediction problem. Report the most important rules obtained and evaluate performance and interpretability.

The work will be developed in Python 3 using scientific open-source libraries like numpy, pandas, scikit-learn, pytorch, etc., because they implement most of the mathematical methods required for the development of this thesis.

1.5 Thesis Structure

In this thesis, we will present a new method for classification from scratch based on the Dempster Shafer Theory. In addition, a study case that uses the method will be presented. This thesis structure is presented below:

- The first chapter contains the introduction, motivation, objective hypothesis, and methodology for this thesis.
- The second chapter presents the related work and all technical information needed for understanding the proposed method.
- The third chapter presents the proposed model. Here we show how to perform classification using DST and our representations, and then we show how to train the model using data to obtain the best classification results.
- The fourth chapter presents the testings performed to validate the proposed method. First, we test on controlled scenarios defined by us and then in some of the most known machine learning data sets. We also compare our results with other classification methods.
- The fifth chapter presents a real application of the method to stroke risk prediction using data provided by a hospital in Japan. First, we will discuss the structure of the available data structure and its quality. Then, data exploration will be presented. After this, we will apply and review some of the existing methods for forecasting stroke risk. Finally, we define the data embedding we will use for the model and present the results of the model, including a comparison with some of the other discussed models.

- The sixth chapter focuses on discussing the interpretability of the results obtained in the stroke risk prediction problem. First, we present the interpretability results of the model, and then three different methodologies are applied to test whether the interpretable results are useful and correct.
- Finally, the seventh chapter presents the conclusion of this thesis alongside with insights for future improvements.

Chapter 2

Related Work

This section reviews the related literature. It explains the methods, theories, and algorithms required to understand this work. First, it presents a review of the machine learning methods for supervised learning, particularly those involving interpretable methods for classification. Then, it presents the relevant aspects of the Dempster Shafer Theory of evidence, which is the mathematical framework our method is based on. Additionally, it describes and discusses applications of Dempster Shafer Theory to supervised learning found in the literature. Finally, other mathematical algorithms are presented for optimization and differentiation.

2.1 Supervised Learning

Supervised Learning is the collection of methods that predict the outcomes of a process given the instance attributes. These methods take samples of these instances and their target values to train their parameters and adjust the model to get the best accuracy. [9]

Supervised Learning algorithms are divided into two groups based on whether the target values are discrete and finite, or continuous. If the target values are discrete and finite, these methods are called classifiers, since for an instance these methods assign a class label to it. For continuous target values, the methods are called regressors.

Several methods have been proposed to tackle classification problems. Some of them start from probabilistic models applying certain assumptions about data, like Naive Bayes [10] and Bayesian Networks [11]; other models have geometric approaches like Support Vector Machines (SVM) [12], K-Nearest Neighbors (KNN) [13] and Linear and Quadratic discriminant analysis [14]. There are also rule-based methods like Decision Trees [15] and Random Forest [16]. Finally, there are other models based on specific computation units that are built to learn the decision boundaries by fitting their internal parameters, e.g., Logistic Regression [17], Artificial Neural Networks [18] and its variations like convolutional and recurrent networks.

Table 2.1 presents a summary of these methods with a brief description of them.

Method	Description
Naive Bayes	The model applies Bayes' theorem with the assumption of conditional independence between every pair of features given the value of the class variable. Using the data, the model finds the values that give the highest likelihood posterior probabilities.
Bayesian Networks	A Bayesian network is a directed acyclic graph in which each edge corresponds to a conditional dependency, and each node corresponds to a unique random variable. The user must provide the structure of the network, and data is used to estimate the probabilities of the network.
Support Vector Machines	The model is a binary classifier that finds the hyperplane in the feature space that separates the classes with the largest margin. Kernel trick can be used to model non-linear relationships, and one class against the others technique can be used to multi-class classification.
K-Nearest Neighbors	To perform a prediction, the model uses the information of the k-nearest neighbors of the input vector in the feature space. Each of these neighbors "vote" for the class they belong to, the predicted class is the majority of this process.
Linear and Quadratic discriminant analysis	The model is a binary classifier that finds the decision boundary as linear and a quadratic functions.
Decision Tree	The model builds a binary tree in which each inner node is a simple rule of data attributes that decides which branch will be used to descent, and leaf nodes are the predicted classes.
Random Forest	The model is a collection of decision trees that were built using random variation of data. The predicted class is the majority of the prediction performed by the decision trees.
Logistic Regression	The model applies a sigmoid function to the input vector multiplied by a weight vector to decide the class. The model finds the optimal weight values using the training data.
Artificial Neural Network	A neuron is defined as a unit that applies a non-linear function to a weighted input vector. Many neurons are organized in layers that are fully-connected to the previous and following layers. The first layer takes the values from the feature vector, and the last layer outputs the classes probabilities.

Table 2.1: Summary of classification methods

2.2 Interpretable methods

Some of the methods mentioned above are interpretable, i.e., that they can explain the decision they make when making predictions. The most remarkable interpretable methods are described in more detail below.

2.2.1 Decision Trees and Random Forest

A Decision Tree [15] is a method for classification and regression based on simple relations among attributes. The technique builds a binary tree, where each inner node represents a simple rule of an attribute, for example, $X_1 > 4$. The branches are the results of the rules. A left branch means the above condition was false, and the right branch says that it was true. Leaf nodes are the predicted classes. The method tries to find the rules that separate most of the classes.

This method is interpretable since we can descend in the decision tree with our data and know why the method gives a response.

Figure 2.1 shows an example of a decision tree for breast cancer detection using information about the nuclei of cells. As shown in the example, this tree is straightforward and interpretable. Any expert can follow the branches of the tree and find the predicted class without any problem.

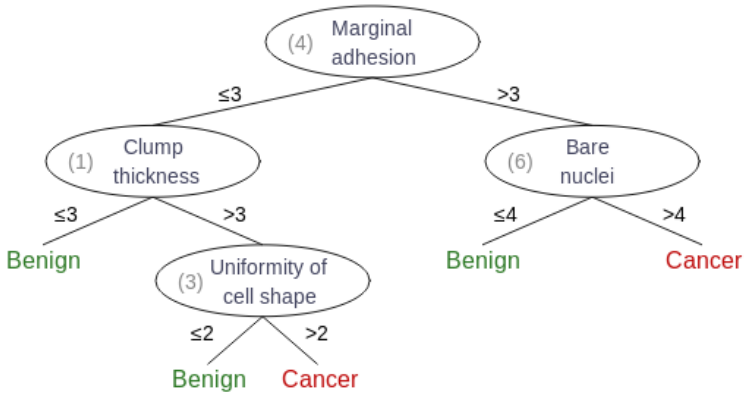


Figure 2.1: Decision Tree example for breast cancer classification obtained from an assignment of a Machine Learning course from ANU

A drawback of Decision Trees is that they cannot handle complex data rules, and consequently, the prediction accuracy is lower compared to other methods.

An extension of the Decision Tree model is the Random Forest [16], which is another classification method that uses a collection of decision trees. Each tree in the forest is built using a variation of the data, and restrictions are applied to ensure that trees with different characteristics will be obtained. To obtain the predicted outcome in a Random Forest, we perform the classification in each of the decision trees that belongs to the forest. Each of them "vote" for the outcome they classified. Therefore, the result of the Random Forest is the majority of the predicted outcomes of the decision trees.

2.2.2 Bayesian derived methods

Naive Bayes [10] is a method based on the Bayes' Theorem, which estimates the distribution of the attributes according to the training data. Then, the model can make inference over these distributions to find the class that has the highest likelihood, i.e., the class which has the highest probability according to the data and the model parameters. This method assumes that attributes are independent and identically distributed from each other. A typical implementation of Naive Bayes is assuming that variables follows a Normal Distribution, then these distributions are estimated using the mean and standard deviation of the training set.

Another similar method is the Bayesian Network [11], which is a directed graph that models a particular problem. In this graph, each node is a variable of the problem, and edges represent dependency or conditionality of these variables. In this method, interpretability comes directly from the structure of the graph because it encodes the knowledge about relationships among variables. However, the structure of the graph has to be set beforehand. Therefore this method does not allow us to find new knowledge but to validate it.

Figure 2.2 shows an example of a Bayesian Network for the detection of lung disease retrieved from book Probabilistic Models of Cognition [19]. Next to the network node, the distribution for the conditional probabilities are presented. In this case, an expert can clearly see which variables depend on the others and the measurement of these dependencies. However, making a prediction is harder than the previous case of the decision tree since the user needs to know how to apply the Bayes Rule correctly.

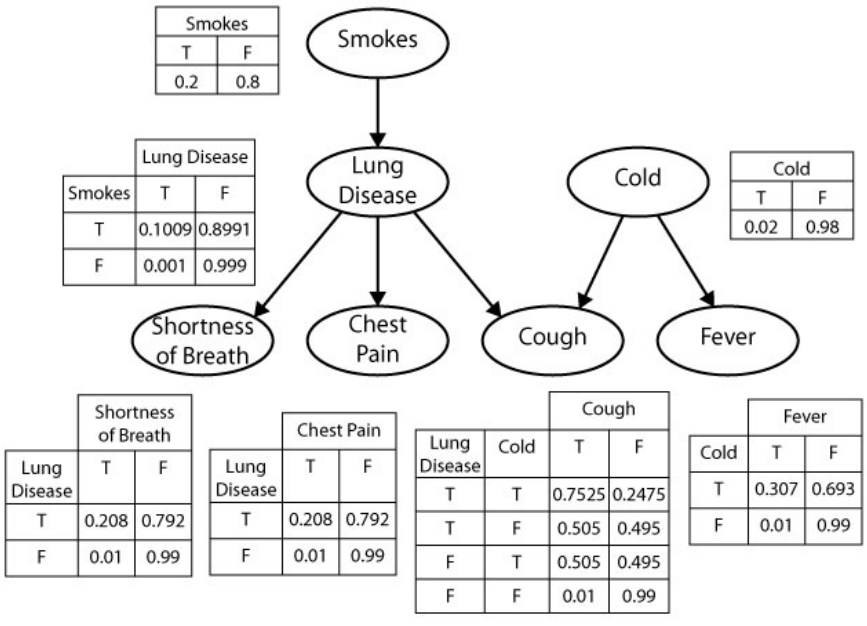


Figure 2.2: Bayesian network example for lung disease.

2.3 Interpretability indicators in Fuzzy Rule-Based Systems

Several authors have proposed indicators to measure the interpretability of a system. Generally, these indicators are applied to fuzzy rule-based systems (FRBS). For example, García et al. [4] propose a methodology to measure the performance and interpretability of rule-based genetics models. For performance measuring, they use indicators such as accuracy, true positive rate, and Cohen's kappa indicator. For interpretability, they use two measurements: the size of the ruleset and the average of the number of antecedents. Let s_i to be a rule statement, s_i can be written as $c_1 \wedge c_2 \wedge \cdots \wedge c_k$, then the number of antecedents for this rule ant_i is k. For example, a rule with a statement "BMI ≥ 35" has only one antecedent. The average number of antecedents (ANT) is defined as:

$$ANT = \frac{1}{r} \sum_{i=1}^{r} ant_i \tag{2.1}$$

Where ant_i is the number of antecedents of the i-th rule and r is the number of rules, these two indicators are widely used in FRBS. As another example, Isibuchi and Nojima [20] use the number of rules as an interpretability metric and the error of classification $(1 - Accuracy)$ as an accuracy metric to test several configurations of a model and they find the Pareto-optimal setup for these two variables.

However, the above metrics for interpretability closely depend on the problem and its data: datasets with few attributes will tend to have fewer rules than datasets with many attributes. To address this problem, a reformulation of these indicators was proposed by Gorzalczany and Rudziński [21]. The main change introduced is that the indicators are normalized, and thus they are comparable to performance indicators of other problems and datasets. In their work, the proposed indicator is the average of another three normalized indicators as defined below:

$$\begin{aligned} Q_{RATR} &= \frac{1}{r} \sum_{r_i \in RS} \frac{ant_i - 1}{m - 1} \\ Q_{ATR} &= \frac{ANT - 1}{m - 1} \\ Q_{FS} &= \frac{n_{FS} - 1}{\sum_{i=1}^{m} a_i - 1} \end{aligned} \tag{2.2}$$

where m is the number of attributes, ant_i is the number of antecedents of the i-th rule, ANT is the average number of antecedents defined above, n_{FS} is the average number of active fuzzy sets, and a_i is the number of rules concerning the i-th attribute.

Note that Q_{RATR}, Q_{ATR}, and Q_{FS} are in the range $[0, 1]$, being 0 the best value for high interpretability and 1 the worst. For Q_{RATR} and Q_{ATR}, a 0 value means that the model uses

simple rules involving only one attribute per rule, and a value of 1 means that the rule uses all the attributes. For Q_{FS}, a 0 value means that the model selects only one rule to perform the classification, and a value of 1 means that they select all the rules.

The authors proposed to average these three indicators to measure the complexity of the model Q_{CPLX}, and then the interpretability Q_{INT} is the complement value of the complexity.

$$\begin{aligned} Q_{CPLX} &= \frac{Q_{RATR} + Q_{ATR} + Q_{FS}}{3} \\ Q_{INT} &= 1 - Q_{CPLX} \end{aligned} \tag{2.3}$$

2.4 Interpretability in other methods

Another approach that gained interest over recent years is to explain and interpret the results of a non-interpretable method for both specific instances and the whole model. Many strategies have been proposed to extract interpretability. Some of them are discussed, and examples are presented below.

The first approach to understand black-box methods has been to analyze them formally by comparing or reducing them to interpretable methods. For example, Gal [22] analyzed the formulas used to optimize the stochastic regularization techniques of an artificial neural network with dropout; the author proves that the equation obtained is equivalent to the optimization of a Bayesian network. Although this kind of techniques seems to be correct for finding interpretability, the intermediate steps to build the equivalence could be as complex as the original model. Then, even if we have new representation with an interpretable model, this does not necessarily mean that we can extract explainable results from it.

Another approach is to use a strategy similar to sensitivity analysis to find the features that most contribute to determine the outcome of predictions [6, 23]. In the case of artificial neural networks (ANN), which are known to be one of the least interpretable methods, there are efforts to understand the relations between inputs and network computations using this technique. Olshen et al. [23] present four methods for understanding the mechanics of ANNs. The mentioned methods are based on testing small random variations of the data and comparing the outputs differences. A drawback of this kind of approaches is that the output of the model has to be recomputed for every perturbation the method produces to the input variables; this method is difficult to perform on problems with large sets of inputs or large ANNs.

Recent studies about interpretability focus only on the predictions of the model instead of trying to understand the model itself. This means for a certain model, an *explanation model* is built, which uses the results of the original one trying to find interpretable relationships among attributes for predictions. Some of these models can even use a different data embedding representation than the first model uses, trying to obtain interpretability from this new representation, which is often simpler and more understandable. [24, 25, 26, 27, 28]

Ribeiro et al. [29] present a technique called Local Interpretable Model-Agnostic Explanations (LIME) to generate explanations for the results produced by a classifier. LIME technique tries to find the local linear separation for a sample using a custom representation but using the original model predictions for this instance. In other words, LIME variates the initial input, then apply the embedding of the model and perform the prediction. If the variation produces a significant change in classification, then LIME declares that the affected input was necessary for the prediction. Instead, if data variation does not involve a change in classification, then this change does not contribute to the process.

LIME has been tested in text-based datasets, image datasets, and tabular data showing that it can get interpretable results independent of the representation of the data. Also, LIME text results were validated with humans who had to decide which explanation of a method fits better in a real scenario showing that the interpretations the model produces are the best results.

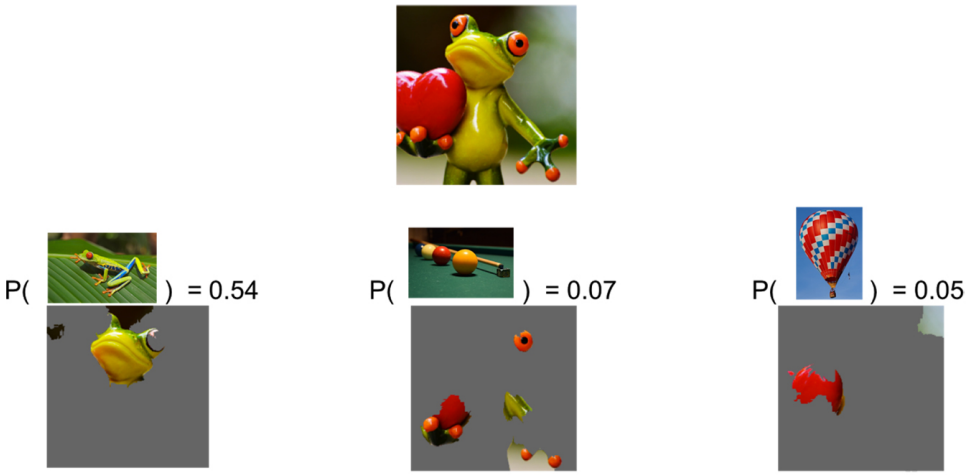


Figure 2.3: Example of Lime explanations when detecting objects in an image, in this case frog, pool table and air balloon are the top predicted classes

Another example of explanation methods for artificial neural networks is DeepLift, presented by Shrikumar et al. [25]. In this work, the authors present a process that can be applied to a trained neural network when receiving an input. This process is similar to back-propagation, but involving computing gradients for convenient functions that are directly related to the importance of the features. The output of this process is weighted by each attribute, indicating the contribution of these features to the prediction.

There are some current works that propose to combine these various models for extracting interpretable results. For example, Lundberg et al. [30] present a method to unify the results of several explanation models to provide a measure of feature importance by adding the interpretability evidence that the different techniques offer.

2.5 Dempster Shafer Theory

The Dempster-Shafer Theory (DST) [7], also called the theory of belief functions or theory of evidence, is a mathematical framework to express and make inferences about processes with uncertainty. The main feature of the approach is that it allows measuring exactly the plausibility of an outcome of a set of results have. The theory is often defined as a generalization of the Bayesian theory because it is more expressive than standard probability models.

2.5.1 Mass assignment functions

Mass assignment functions (also called basic belief assignment, basic probability assignment, or simply mass function) are the core elements of Dempster-Shafer Theory (DST). Mass assignment functions encode the evidence and uncertainty that supports or disfavor inferring certain output. Mass assignment functions (MAF) are mappings from subsets of all possible outcomes to values between 0 and 1.

Mathematically, let X be the set of all states or outputs of a system or process, called frame of discernment. A mass assignment function m is a function that satisfies:

$$m : 2^X \to [0, 1], \quad m(\emptyset) = 0, \quad \sum_{A \subseteq X} m(A) = 1 \tag{2.4}$$

The term $m(A)$ can be interpreted as the probability of getting precisely the outcomes of the set A, and not a subset of A. This is the feature that actually allows to include uncertainty in the reasoning. Note that a piece of evidence that supports a single outcome can be mapped to their singletons. Similarly, uncertainty can be mapped by giving mass to the sets that contain many different results. In this way, we can explicitly express the uncertainty about which outcomes we cannot discern. For example, an amount of uncertainty that cannot discern between any possible outcome should be mapped as mass given to the subset, which contains the complete set of possible outcomes. If we have an amount of uncertainty that can discard the occurrence of some outcomes but not others, then we should map it as assigning mass to the set that contains all the outputs but the discarded ones.

We will present an example to illustrate how mass functions work. Consider a coin toss game which a player is trying to guess the outcome heads or tails of the tossing. The player knows that the coin is evenly balanced, thus the "probability" of getting a head is equal to getting a tail. In classical probabilities, we can express this as $P(\text{heads}) = P(\text{tails}) = 0.5$. Similarly, in DST we can express the following mass assignment function $m(\phi) = 0$, $m(\{\text{heads}\}) = m(\{\text{tails}\}) = 0.5$, $m(\{\text{heads, tails}\}) = 0$. Note that we assign a value of 0 to the complete set, because in this case there are completely evidence of the possibilities of the outcomes thus there is no uncertainty. Now consider the same game but in this case the player does not know anything about the coin to be tossed, he or she does not know if the coin is fair or if it is biased to some outcome. In classical probabilities, the

expression of this scenario is unknown or incomplete, but if we were force to assign a value our best choice will be the same as before, i.e. $P(\text{heads}) = P(\text{tails}) = 0.5$. But in DST the expression for this case is completely different, the corresponding mass assignment function is $m(\phi) = m(\{\text{heads}\}) = m(\{\text{tails}\}) = 0,\ m(\{\text{heads, tails}\}) = 1$ stating that in this process there are no evidence for any singular outcome, instead we have full uncertainty for all outcomes.

Another way to understand mass assignment functions that help us building them is to consider MAFs as bets. Consider a coin toss game again, but this time the player has \$100, and he or she must bet all the money to the power set of the outcomes. Thus, there are four possibilities: The null set, that means neither heads or tails will be the outcome; the singletons for heads and tails, which means that heads or tails will be outcome respectively; and the complete set, that means either heads or tails will be the outcome. Each set has a return factor, for example, 1x for the null set, 1.5x for singletons, and 1x for the complete set. The player can split his or her bet to the different options. A rational player should comply with the following statements: Null set will never be an option to bet in because there is always an outcome, and this coincide with the fact that the mass of the null set is always 0. Complete set will always be true, but the return factor of 1 is equivalent to do not perform the bet, then, when uncertainty is low, the player should bet more in singletons since he or she knows which outcome is more likely to happen, and if uncertainty is high, the player should bet more in the complete set.

A mass function is equivalent to a discrete probability distribution in Bayesian theory.

2.5.2 Belief and Plausibility

Although MAFs can express the possibilities of outcomes of a process better than classical probability theory, many times, we need to make inference about a single output, for example, knowing their corresponding probability in the Bayesian theory. Unlike in traditional probability, this process is not trivial in the Belief Theory because the information of an outcome appears in many parts of the MAFs definition. For example, consider a frame of discernment with three elements $\Omega = \{A, B, C\}$ if we are interested on making inference about A with respect to a mass m over Ω, then we will potentially have to look at the values of $m(\{A\})$, $m(\{A, B\})$, $m(\{A, C\})$, and $m(\{A, B, C\})$ since all these sets contain A.

DST defines two indicators to measure the amount of evidence that support a single outcome or a set of outcomes as well. These indicators are the belief and plausibility, and they are explained below.

The belief metric is defined as the total evidence that supports an outcome, and it can be computed using the following expression:

$$Bel(A) = \sum_{X \subseteq A} m(X) \tag{2.5}$$

The idea of belief is to sum the evidence that we are entirely sure that supports the outcome. Then, when computing the belief of a set, it only considers the masses of the set itself and their subsets. Note that the belief of a singleton is equal to the mass of the singleton. As belief is strict with the sets to be considered, it defines a lower bound for the corresponding probability of the outcome.

The plausibility metric is defined as the total amount of evidence that can support an outcome. This formulation is the following:

$$Pl(A) = \sum_{X \cap A \neq \phi} m(X) \tag{2.6}$$

Unlike belief, plausibility indicator considers all sets that are related to the set of interest, and this is expressed as all set, which as a non-null intersection with the set of interest. Note that in this case, for a singleton, many sets contribute to its plausibility. As plausibility considers all sets related to A without any exclusion, this defines an upper bound for the corresponding probability.

Note that every set considered in the computation of the belief indicator is also considered in plausibility indicator because any non-null subset of any set intersects non-null with the set. This property hints that $Bel(A) \leq Pl(A)$. In fact, this inequation has been tested alongside the real probability of the outcomes of a process. It has been proven that for any outcome, belief and plausibility values define lower and upper bounds for the classical probability of that outcome [31].

$$Bel(A) \leq P(A) \leq Pl(A) \tag{2.7}$$

Many authors have suggested that in order to make decisions based on mass or belief functions, masses first need to be transformed to probabilities [32]. DST does not provide a formula to calculate the probability of the outcomes. The above inequation is the only formal way to estimate that probability. Note that when uncertainty is low, i.e., the values of sets with two or more elements are low, the value of $Pl(A)$ approaches the value of $Bel(A)$ and then we can estimate better the value of $P(A)$.

Finally, it is important to mention that belief theory, which is an equivalence to DST, defines belief functions as their core elements rather than mass functions. If you set the values for the belief to all sets in the power set of the frame of discernment Ω, you can still compute the mass and plausibility values. In fact, there is a one to one correspondence between belief functions and mass functions. The transformation of belief functions into mass functions was defined by Thoma [33], and the following equation describes it.

$$m(A) = \sum_{B \subseteq A} (-1)^{|A \setminus B|} Bel(B) \tag{2.8}$$

2.5.3 Probability estimation

Many authors have proposed indicators based on belief and plausibility to estimate the probability. We will show and discuss the two most important.

One of the most used and accepted is the pignistic transformation proposed by Smets [34]. The probability is estimated using the following formula:

$$P(a) = \sum_{a \in A} \frac{m(A)}{|A|} \tag{2.9}$$

Where $|A|$ is the cardinality of A, the transformation can be interpreted as the sum of all subsets that are related to a. Still, if the set contains other elements apart from a, its contribution is split evenly into these outcomes.

Another proposed approach to estimate the value of the corresponding probabilities is the plausibility transformation proposed by Cobb and Shenoy [35]. This can be defined as the following formula:

$$P(a) = \frac{Pl(\{a\})}{\sum_{b \in \Omega} Pl(\{b\})} \tag{2.10}$$

Basically, the above formula computes the ratio between the plausibility of one output with respect to the sum of the values of the plausibility of the singletons of all outcomes. This transformation has interesting properties such as when there are no uncertainty, i.e., only singletons have mass, their values are the values of the probability estimated, and when uncertainty is high, the plausibility values tends to be high and close each other, then the division with respect to their sum tends to be $\frac{1}{n}$ (where n is the number of outcomes) which coincides with the prior probability of an unknown event.

2.5.4 Dempster Rule

In order to solve inference problems, mass functions representing different pieces of evidence need to be combined in a meaningful way. The theory states that multiple evidence sources expressed by their mass assignment functions of the same frame of discernment can be combined using the Dempster Rule (DR) [8], which is also called conjunctive rule. Given two mass assignment functions m_1 and m_2, a new mass assignment function m_c can be constructed by the combination of the other two using the following formula:

$$\begin{aligned} m_c(A) &= m_1(A) \oplus m_2(A) \\ &= \frac{1}{1-K} \sum_{B \cap C = A \neq \phi} m_1(B) m_2(C) \end{aligned} \tag{2.11}$$

Where K is a constant representing the degree of conflict between m_1 and m_2, and it is given by the following expression:

$$K = \sum_{B \cap C = \phi} m_1(B) m_2(C). \tag{2.12}$$

The intuition of the Dempster Rule is that the combined mass value for a set is the multiplication of the masses of all the combinations of sets that intersect the set of interest. The purpose of the factor $\frac{1}{1-K}$ is for normalization, since if we do not include that factor in the expression, the sum of all the mass values could be less than 1. However, as we already said, K also measures the degree of conflict, and that is precisely what its formula captures since it considers the multiplication of the mass values of all sets whose intersection is empty. In other words, it considers all the amount of evidence that contradict each other. Additionally, K takes values from 0 to 1. When K is 0, there is no empty intersection among the sets, and thus the initial MAFs are complementary. When K is 1, the MAFs are contradictory, and in this case, the combination cannot be made, since it leads to a zero division.

The Dempster Rule operator is often symbolized as $\oplus$. Let $\mathcal{M}_\Omega$ be the set of all possible MAFs over a frame of discernment Ω, then $(\mathcal{M}_\Omega, \oplus)$ is an Abelian semigroup [36]. Thus implies that the operator has useful mathematical properties like associativity and commutativity.

$$\begin{aligned} m_1 \oplus m_2 &= m_2 \oplus m_1 \\ m_1 \oplus (m_2 \oplus m_3) &= (m_1 \oplus m_2) \oplus m_3 \end{aligned} \tag{2.13}$$

Also, Dempster Rule has absorbent and neutral elements. The rule is absorbent to full certainty evidence sources and neutral to full uncertainty. Let $C_A : 2^X \to [0,1]$ and $A \in 2^X$ such that $C_A(A) = 1$ and $\forall Y \neq A,\ C_A(Y) = 0$ and $U : 2^X \to [0,1]$ such that $U(X) = 1$ and $\forall Y \neq X,\ U(Y) = 0$. Then the following statements are true, demonstrating examples of the absorbent and neutral elements.

$$\begin{aligned} m \oplus C_A &= C_A \\ m \oplus U &= m \end{aligned} \tag{2.14}$$

Although the wide acceptance of Dempster Rule as the combination operator for mass assignment functions, it has some drawbacks. The most important is the undefined value of the combination when computing the Dempster Rule of two conflicting MAFs. As mentioned before, the K factor can reach the value of 1, causing a zero division. We present an example to illustrate this issue. Consider a frame of discernment $X = \{a, b\}$, a mass m_1 defined as $m_1(\{a\}) = 1$ and 0 for all other subsets, and a mass m_2 defines as $m_2(\{b\}) = 1$ and 0 for the rest, the conflict constant K for m_1 and m_2 is computed as follows:

$$K = \sum_{B \cap C = \phi} m_1(B) m_2(C) = m_1(\{A\}) m_2(\{B\}) = 1$$

As explained above, $K = 1$ generates a zero division in the definition of the Dempster Rule, making it unable to be computed. However, note that the same can happens with traditional probability functions when applying the Bayesian theorem if all prior-likelihood products are 0. So reaching a state where Dempster Rule is undefined is considered an error in the definition of the masses rather than an error in the formula. Without limiting the preceding, some authors suggest returning a full uncertainty mass as the result of the combination of two total contradictory masses.

Regarding computational implementation and algorithm efficiency, the Dempster Rule is a costly operation. In order to give an approach of the computational complexity of the combination, we will present the complexity of a brute-force algorithm. Let Ω to be a frame of discernment, let n to be the cardinality of Ω and let m_1 and m_2 masses over Ω to be combined using DR. First, note that if we write m_1 as a relation in the form of a set of pairs $(A, m_1(A))$, there is a total of 2^n pairs; the same applies for m_2. Following the Dempster Rule definition, each element of m_1 has to be intersected and multiplied with all the elements of m_2, producing a value that will contribute to only one element of the result. Thus, as we have 2^n elements in m_1 and also 2^n elements in m_2, the computational complexity will be $O(2^n * 2^n) = O(4^n)$. Although there are a lot of techniques to reduce this complexity, Orponen has proved that computing the combined MAF using Dempster Rule is #P-complete [37].

There are other combinations rules for mass assignment functions that have been proposed, such as the Disjunctive Rule [38], which is the "or"-like operation while the Dempster Rule is an "and"-like operation. The main difference is that the rule searches for sets in which their union is the set of interests rather than its intersection. Other more complex combination rules that use the results of disjunctive and conjunctive rules have also been proposed, such as the Yager's Rule [39], which aims to represent conflict in a more meaningful way. These rules can also be implemented and used, but they have at least the same complexity as the Dempster Rule. In the rest of this thesis, we will use Dempster Rule as the combination rule for MAFs.

2.5.5 Commonality functions

As we explained before, Dempster Rule is a complex operation. However, there is a technique that makes its implementation much more easier to perform: First, we need to introduce the concept of Commonality function. The Commonality function $q(A)$ states how much mass in total is committed to A and all of the super-sets of A over the frame of discernment Ω. Mathematically, it can be expressed as follows.

$$q(A) = \sum_{A \subseteq B \subseteq \Omega} m(B) \tag{2.15}$$

The commonality $q(A)$ can also be interpreted as a measure that expresses how much mass potentially supports the entire set A. Similar to belief, commonality has a one-to-one correspondence to mass assignment functions. The reverse transformation can be computed using the following formula:

$$m(A) = \sum_{A \subseteq B \subseteq \Omega} (-1)^{|B \setminus A|} q(B) \tag{2.16}$$

Dempster Rule can be rewritten in terms of commonality functions instead of mass assignment functions, given the following expression:

$$\begin{aligned} q_1(A) \oplus q_2(A) &= \eta q_1(A) q_2(A) \\ q_1(\phi) \oplus q_2(\phi) &= 1 \\ \eta &= \sum_{B \neq \phi} q_1(B) q_2(B) \end{aligned} \tag{2.17}$$

It turns out that for commonality functions, Dempster Rule is reduced to computing a pure product of commonality values where η is a normalization constant similar to K. Such a product can be computed more efficiently than the original expression for the Dempster Rule (Eq. 2.11). In many cases, (e.g., when Dempster Rule should be applied several times) transforming MAFs into commonality functions, then applying the simplified Dempster Rule, and finally applying the reverse transformation, has much lower computational effort than using original Dempster Rule formula for MAFs [40].

2.6 Dempster Shafer Theory applications in supervised learning

Several models using Dempster-Shafer Theory have been proposed to solve supervised learning tasks. These methods can be divided into two groups: Dempster-Shafer supporting post-processing of other classifiers, which are often called data fusion methods in the literature, and techniques that use DST as part of the classification process. Both approaches are described below.

Since the nature of the DST is to handle incomplete and uncertain data, their first applications were made in post-process of the classification tasks [41]. These methods usually operate as follows: Several classifiers are executed for a sample, obtaining the probability of belonging to each one of the possible classes. Then, these probabilities can be interpreted as sources of evidence supporting these classes, and finally, the DST can be applied to compute the most plausible class.

DST has been applied to the first stage of a classification process, particularly as a feature selector or data generator. For example, Mulyani et al. [42] proposed a model that combines Dempster-Shafer Theory and a Naive Bayes classifier in order to predict diseases. They constructed an expert system based on medical surveys to estimate the values of the mass assignment functions. If there is more than one outcome predicted by DST, then the Naive Bayes classifier is applied.

On the other hand, we have DST as part of the classification process. Many of these works start from other well-known methods where some of the computations or assumptions are used as evidence to assign mass, and then DST operations are applied.

For example, Denoeux [43] proposed a variation of KNN classifier, which instead of using neighbors' votes, neighbors support evidence in favor of their corresponding class. Nearest neighbors contribute with more certainty than the farthest ones. The decision is then made by the application of the Dempster Rule instead of using vote counting.

The same author proposed a model based on a modification of an RBF artificial neural network architecture [44] which behaves similar to a Multilayer Perceptron, but uses the weights of neuron links as evidence input for the Dempster-Shafer theory as well as the Dempster Rule for pooling.

Models using Dempster-Shafer principles to build classifiers instead of modifying an existing classifier have also been proposed. For example, Chen et al. [45] presented a classifier that examines the most important features. A mass is then assigned according to the training values, and finally, the prediction is obtained by the application of the combination rule. This classifier was tested with three traditional data sets and compared to other classifiers. The obtained accuracy is, in most cases, comparable to the other methods. However, there is no discussion about interpretability in this work.

Fixsen and Mahler [46] present another example of this kind of model. In their work, a modified version of the Dempster-Shafer theory that includes prior knowledge about the possible outcomes is proposed. This prior knowledge is built using the training data. Then the process of classification is explained using a custom distance function for the evidence using DST and Dempster Rule.

Peñafiel et al. [47] present a system based on Dempster-Shafer Theory to predict the risk of a patient having a heart attack or stroke based on past medical checkup data. The model is based on rules, which are statements that can be tested to be true according to data. For example, a rule could be having LDL cholesterol over 60 mg/dL. Using these rules, the system builds the mass assignment functions required by DST to operate, and the values of the masses are established by defined formulas computed with the training data. The model gets a 61% accuracy when predicting stroke occurrences for patients within the next year. A significant output of this model is the identification of the rules that most contribute to a positive prediction of having a stroke.

As mentioned above, this model is the baseline of this work. This thesis will explore a generalization of the mentioned method to handle any classification problem, including multiclass classification tasks. Also, a new way of setting the mass values is proposed, inspired by the good results artificial neural networks have obtained in recent years.

2.7 Mathematical methods

In this section, we will discuss some of the mathematical methods and their respective algorithmic representations that we used in this thesis. These methods are divided into two groups: Optimization methods and Automatic numerical differentiation methods. These are presented below.

2.7.1 Optimization Methods

Optimization methods are algorithms that find the values of certain parameters in order to obtain the minimum or maximum value of a target function, subject to some constraints. Many algorithms to solve optimization problems, also called optimization methods, have been proposed over time and they differ each other according to the problem properties (e.g., the number of parameters or the form of the target function) or whether certain assumptions are valid (e.g., the target function is convex).

Optimization methods can be classified into two groups: Basic Optimization methods and Gradient-based Optimization Methods. These two groups are discussed below.

Basic Optimization Methods

Basic optimization methods are a family of numerical optimization algorithms that only use the values of the target function to find the optimum set of parameter values. These methods are recommended to be used when the target function is relatively unknown, and its gradient is not possible to be computed. Examples of these methods are explained below.

Pattern Search [48] is an iterative method that starts with a random configuration of the parameters. In a step, for each parameter, the method computes the value of the target function for all the combinations of the parameters when adding d to each parameter and subtracting d to each parameter. The new proposed solution for the optimization problem is the configuration that has the maximum or minimum value. The process is repeated but adding and subtracting $d/2$ instead of d.

Particle Swarm Optimization [49] is another method to solve optimization problems. The method starts with n particles that have a random position in the space of the parameters initially. Each particle also has an initial random velocity, which is also a vector in the parameter space. In each step, the method finds the particle that has maximum or minimum target value, which is called the candidate. All particles are moved according to their velocity, and finally, their velocity is updated to a linear combination between a random velocity and a velocity vector that points to the candidate position. After k steps, the optimum configuration of parameters is the position of the candidate particle.

Genetic Algorithms [50] is the name of a family of optimization methods that have gained interest over the recent years. The biological evolution of species process inspires Genetic

Algorithms. A genetic algorithm starts with a randomly defined set of parameters configurations, which is called a population. In each step, the best individuals of the population according to their target values are selected. These selected individuals are recombined using a crossover operation and possibly randomly mutated value of parameters. After this process, new individuals are created, which conforms to the population for the next iteration. The process is repeated for several iterations, and the values of the best individual of the last generation (population) are used as the solution for the optimization problem.

Gradient-based Optimization Methods

Gradient-based optimization methods are another important family of optimization methods. For using this technique, the target function should be differentiable, which is often true. These methods use the gradient of the target function to "orient" the direction where the parameters should be updated. These methods can be classified according to the highest level of differentiation they use. For example, first-order gradient methods only take information from target function values and its first derivative. Second-order gradient methods use the information of these and the second derivative too.

The best known first-order gradient method is Gradient Descent (GD). For a minimization optimization problem, this method updates the values of the parameters by subtracting the gradient of the target function with respect to that parameter and scaling it by a factor known as the learning rate. Gradient Descent has many variations and extensions that will be discussed later.

The most known second-order gradient method is the Quasi-Newton method [51]. This method was originally built to find numerically the zeros of a function using their gradient information. This idea can be extended to solve optimization problems since the zeros of the gradient of a function coincide with the local minimum and maximum of the original function. This process requires to compute the gradient of the gradient of the target function, which is the second derivative of the target function, and thus is a second-order gradient method.

After this brief presentation of the most known optimization methods, the important question is to decide which method is better. The answer is that it depends on the properties of the target function and the number of parameters that will be optimized. In our case, for a typical problem, we will have hundreds of variables to be optimized, the target function will be differentiable, and many operations with these variables like sums, multiplications, and exponentiation will be part of the function. The target function will also be dynamic so that it will change structurally according to the data. Using all this information, we can state that gradient-based algorithms could be applied since our function is differentiable, and these algorithms tend to converge faster than the other methods. Among gradient methods, first-order methods should be used because the target function will be complex, and then computing higher derivative will require a lot of computational effort, and also they tend to converge faster on large data sets [52]. Finally, the Gradient Descent technique is chosen for this work. Further details about its extensions and variations are presented below.

As explained above, Gradient Descent is based on the idea of updating the variables in

the direction of the gradient. Mathematically, given the target function J, which depends on the variable x and our goal is to minimize the value of J, the Gradient Descent applies the following formula for variable updating:

$$x_{t+1} = x_t - \alpha \frac{\partial J}{\partial x} \tag{2.18}$$

Where the value α is a constant called the learning rate. This algorithm gives us a sequence of values for $x_0, \ldots, x_k$ that minimize J. Initial value for x (i.e x_0) is usually selected randomly.

When the target function depends on the data, but the parameters to be optimized are the same, several strategies can be applied to solve this kind of optimization problem. The most obvious and correct way to perform the optimization is to take all available data records and sum (or aggregate) the target functions then perform a single Gradient Descent because it will use all the information available to optimize. However, the dataset could be large, and then the aggregated target function becomes complex and hard to compute its gradient. Another strategy is to simply use one data record to build the target function and perform the gradient descent, then use the second record and repeat the process, this strategy is known as Stochastic Gradient Descent (SGD) [53].

Although SGD reduces the computational effort of GD, it has some drawbacks as well. The most important drawback of SGD is that parameters updates have high variance, thus producing erratic fluctuations over the target function values, which hinders the algorithm convergence. This happens because in a real-case data set, data records could be "outliers" (i.e., they behave or present properties different than the normal records). Then their gradient could incorrectly point to regions where the target value is higher.

When using data, GD and SGD can be seen as the extreme alternatives for data handling. GD can be seen as an algorithm that uses all data records to execute an update, and SGD can be seen as the same, but using just one data record each iteration. Another approach is to use a subset of the data records with fixed size k to perform an update. This alternative is called Mini-batch Gradient Descent. Mini-batch gradient descent generally solves the erratic behavior of SGD and keeps the computational effort bounded.

Figure 2.4 shows a comparison between GD, SGD, and Mini-batch gradient descent for a Linear regression optimization problem. The left figure shows how the target values decrease over the parameter updates, and the right figure shows the amount of time taken for processing 100 parameter updates.

In this example, we can note that GD is the smoothest method, and it requires only a few updates to reach values close to the optimum. The figure also verifies that SDG has a lot of fluctuations which implies it requires a high number of iteration to reach lower target values, and thus to converge. Also, the figure shows how mini-batch gradient descent behaves smoother and converges faster than SGD. Comparing the time to perform an update, the figure indicates that GD is the slowest, and SGD is the fastest, which coincides with the expected computational effort for each method.

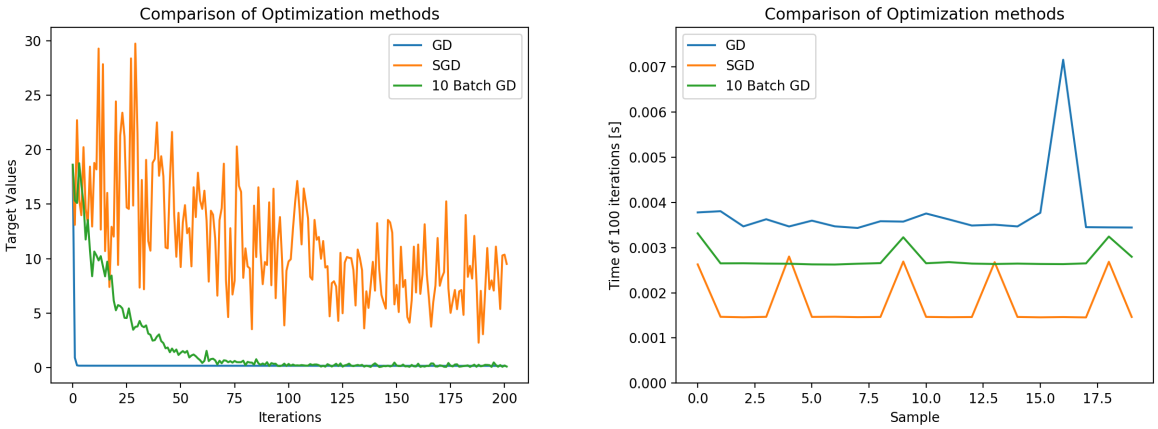


Figure 2.4: Comparison among Gradient Descent, Stochastic Gradient Descent and Mini-Batch Gradient Descent

Another major problem that concerns GD, SGD, and Mini-batch GD is the value assigned to the learning rate. Parameters new values after an update heavily depend on the value of the learning rate. A small value for the learning rate will make the updates slightly change the value of the parameters, and thus the process will take a lot of iterations to converge. On the other hand, a large value for the learning rate will make the updates to exceed the optimum value reaching to incorrect target values that will have higher error and thus the process will oscillate between lower and higher target values or even it could diverge.

To illustrate this problem, Figure 2.5 shows a comparison of small, optimal, and large values for the learning rate of a 10-batch GD for a Linear regression optimization problem.

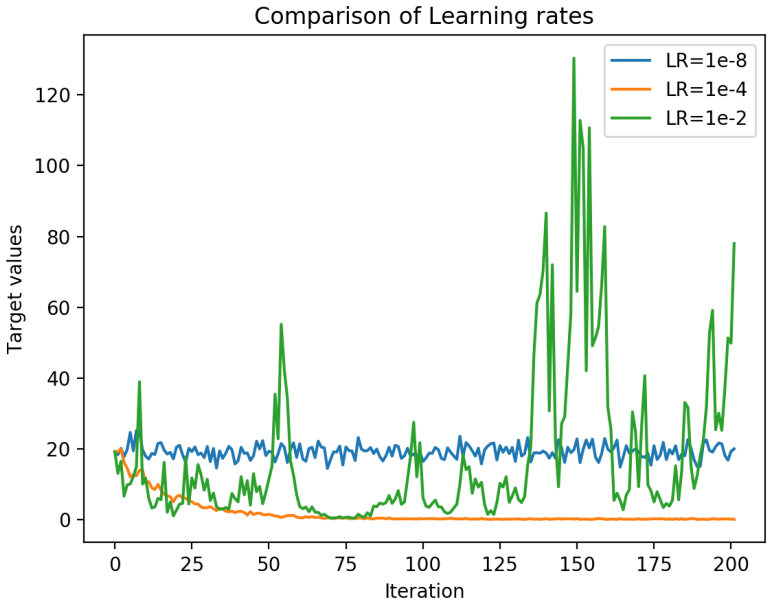


Figure 2.5: Comparison of different values for the learning rates

If the optimum value for the learning rate is unknown, the methods should find that value or adapt the learning rate while the optimization process is being executed. Several methods that adjust the value of the learning rate have been proposed. The most used are listed below.

AdaGrad [54] is a variation of GD that adapts the learning rate by decreasing it in each step according to the square of the magnitude of past gradients. The model uses an extra variable g_t to measure this decay, g_t is also called the second-order moment of the gradient. It can be computed using the following formula:

$$\begin{aligned} g_{t+1} &= g_t + \frac{\partial J}{\partial x}^2 \\ x_{t+1} &= x_t - \frac{\alpha}{\sqrt{g_{t+1}} + \varepsilon} \frac{\partial J}{\partial x} \end{aligned} \tag{2.19}$$

RMSProp [55] follows the same idea of AdaGrad, but it also includes the contribution of the magnitude of the past gradient without square them. This is called the first-order moment of the gradient m_t. It also provides momentum by a constant μ to decrease the variance of gradient differences between steps. RMSProp can be defined using the following formula:

$$\begin{aligned} m_{t+1} &= \gamma m_t + (1-\gamma)\frac{\partial J}{\partial x} \\ g_{t+1} &= \gamma g_t + (1-\gamma)\frac{\partial J}{\partial x}^2 \\ v_{t+1} &= \mu v_t - \frac{\alpha}{\sqrt{g_{t+1} - m_{t+1}^2 + \varepsilon}} \frac{\partial J}{\partial x} \\ x_{t+1} &= x_t + v_{t+1} \end{aligned} \tag{2.20}$$

Adam [56] is another variation that uses the first and second moments of the gradient. The main difference of Adam with RMSprop is that Adam applies the momentum technique to moments. Adam can be computed using the following formula:

$$
\begin{aligned}
m_{t+1} &= \gamma_1 m_t + (1-\gamma_1)\frac{\partial J}{\partial x} \\
g_{t+1} &= \gamma_2 g_t + (1-\gamma_2)\frac{\partial J}{\partial x}^2 \\
\hat{m}_{t+1} &= \frac{m_{t+1}}{1-\gamma_1^{t+1}} \\
\hat{g}_{t+1} &= \frac{g_{t+1}}{1-\gamma_2^{t+1}} \\
x_{t+1} &= x_t - \frac{\alpha \hat{m}_{t+1}}{\sqrt{\hat{g}_{t+1}} + \varepsilon}
\end{aligned}
\tag{2.21}
$$

2.7.2 Automatic differentiation

Since we opted for Gradient Descent techniques for the optimization of the model, we also need to have methods for computing the gradient efficiently. As we explained above, the target function will have several parameters to be optimized, and the function will structurally change according to the data, thus manually obtaining the derivatives of the target function with respect to the variables is not achievable.

Computer-assisted differentiation algorithms can be classified into three groups: Numerical differentiation, which is the set of algorithms that estimate gradients based on the function values only, e.g., estimating the derivative of a function by computing the slope of two close points. The second group is Symbolic differentiation; in this case, function parameters are considered symbols rather than values, and then they are not affected by the numerical precision of the system. These methods generally "parse" the function definition to represent it in a more suitable data structure (usually a directed graph or tree) and then computing the derivative by applying conventional calculus rules. The final group of methods is Automatic Differentiation algorithms (AD), which accumulates the value of the derivatives of sub-expressions of the original function. Then by iterative application of the chain rule, they obtain the value of the derivative.

One of the first works of AD was proposed by Linnainmaa [57]. His work aimed to describe algorithms to determine the coefficients of the Taylor expansion. To do this, the author defines several algorithms; in particular, he introduces the concept of accumulation of gradient values, which is the base of AD, and he also offers a visualization of the execution of his algorithm as a graph.

AD allows differentiation of any differentiable operation [58]. The method defines three elements: constants, variables or parameters, and expressions, which are explained below.

Constants are simply real numbers for example -3, 0 or $\frac{1}{2}$. Constants have the property that they do not change when the input or parameters change, and thus their derivative is always 0.

Variables are numerical values that represent parameters that can potentially change, and they are the variables in which an expression can be differentiated. As these values can change, its derivative is not 0, as in the case of Constants. The value of the derivative of a variable is defined as 1 if the derivative is with respect to the same variable or 0 elsewhere. Variables usually are expressed as letters like x or ω.

Expressions are combinations of other expressions, constants, or variables by the application of a function or operator. Some examples of expressions are $2x$ as it is the combination of constant 2 and variable x by the multiplication operator; $sin(\theta)$ as it is the application of the function sin to the variable θ; or $2x + sin(\theta)$ as it is the combination of the two previous expressions.

Each operation or function that allows build expressions, such as sums, multiplications, or the sin function, must define how the derivative can be computed. For the case of operators, the standard rules of calculus are applicable. Mathematically, let E_1 and E_2 to be expressions and x a variable, then the following rules apply:

$$\begin{aligned} \frac{\partial}{\partial x}(E_1 \pm E_2) &= \frac{\partial E_1}{\partial x} \pm \frac{\partial E_2}{\partial x} \\ \frac{\partial}{\partial x}(E_1 * E_2) &= E_1 \frac{\partial E_2}{\partial x} + \frac{\partial E_1}{\partial x} E_2 \\ \frac{\partial}{\partial x}\left(\frac{E_1}{E_2}\right) &= \frac{\frac{\partial E_1}{\partial x} E_2 - E_1 \frac{\partial E_2}{\partial x}}{(E_2)^2} \end{aligned} \tag{2.22}$$

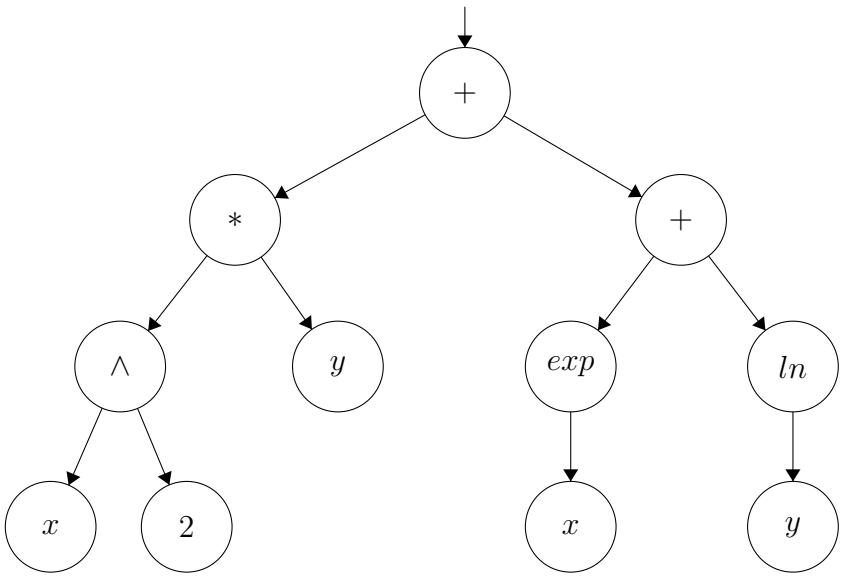
For the case of functions, we apply the chain rule. Let f to be a function applied to an expression E_1. Then the chain rule states the following.

$$\frac{\partial f(E_1)}{\partial x} = \frac{\partial f}{\partial E_1} \frac{\partial E_1}{\partial x} \tag{2.23}$$

For each function we want to include for AD, we should also provide the function that allows us to obtain its basic derivative i.e., $\frac{\partial f(x)}{\partial x}(x)$.

In order to show how AD really works, an example will be analysed. The target function to be differentiated will be $f(x, y) = x^2 y + e^x + ln(y)$ and it will be evaluated at the point $x = 2$, $y = 3$.

The target function can be converted to an expression tree, where leave nodes are the variables or constants, and the inner nodes are operations or functions. The tree that represents the function f is the following.



Using this representation, we can build the expressions that relate to other expressions only using an operator or function. The following table shows these expressions.

Expression	Symbolic value	Numerical value
E_1	x^2	4
E_2	$E_1 y$	12
E_3	e^x	e^2
E_4	$ln(y)$	$ln(3)$
E_5	$E_3 + E_4$	$e^2 + ln(3)$
E_6	$E_2 + E_5$	$12 + e^2 + ln(3)$

Table 2.2: Expression decomposition for function f.

Note that using this decomposition of the function in simpler expressions, the target function f is the same as E_6.

Using these expressions and the derivative rules explained before, we can compute the derivative of each expression with respect to a variable s.

Note that if we are interested on computing the derivative of f with respect to x, i.e., $s = x$. Then, we have that $\partial x/\partial x = 1$ and $\partial y/\partial x = 0$. Then using the expressions presented in Table 2.3 and replacing by their corresponding values, this derivative can be computed.

Note that the final value of the Table 2.4, i.e., $12+e^2$ is the actual value of $\partial f/\partial x$ evaluated at $x = 2,\ y = 3$.

We can perform the same procedure to obtain the value of the derivative of f with respect to y. This process is presented in Table 2.5.

In this case, the value 13/3 is the value of $\partial f/\partial y$ evaluated at $x = 2,\ y = 3$.

Note that even when we present the symbolic values of the expressions, they are not

Expression	Symbolic value
$\frac{\partial E_1}{\partial s}$	$2x\frac{\partial x}{\partial s}$
$\frac{\partial E_2}{\partial s}$	$\frac{\partial E_1}{\partial s}y + E_1\frac{\partial y}{\partial s}$
$\frac{\partial E_3}{\partial s}$	$e^x\frac{\partial x}{\partial s}$
$\frac{\partial E_4}{\partial s}$	$\frac{1}{y}\frac{\partial y}{\partial s}$
$\frac{\partial E_5}{\partial s}$	$\frac{\partial E_3}{\partial s} + \frac{\partial E_4}{\partial s}$
$\frac{\partial E_6}{\partial s}$	$\frac{\partial E_2}{\partial s} + \frac{\partial E_5}{\partial s}$

Table 2.3: Symbolic values for the derivative of f with respect to the variable s.

Expression	Symbolic value	Numerical Value
$\frac{\partial E_1}{\partial x}$	$2x\frac{\partial x}{\partial x}$	4
$\frac{\partial E_2}{\partial x}$	$\frac{\partial E_1}{\partial x}y + E_1\frac{\partial y}{\partial x}$	12
$\frac{\partial E_3}{\partial x}$	$e^x\frac{\partial x}{\partial x}$	e^2
$\frac{\partial E_4}{\partial x}$	$\frac{1}{y}\frac{\partial y}{\partial x}$	0
$\frac{\partial E_5}{\partial x}$	$\frac{\partial E_3}{\partial x} + \frac{\partial E_4}{\partial x}$	e^2
$\frac{\partial E_6}{\partial x}$	$\frac{\partial E_2}{\partial x} + \frac{\partial E_5}{\partial x}$	$12 + e^2$

Table 2.4: Computing the derivative of f with respect to the variable x using AD.

required. The values of the expression and its derivative with respect to all the possible variables are the only information needed for computing further derivatives. No matter how this expression combines with others, the resulting expression cannot use other information than these values. Also, note that this information is only numerical, not functions; thus, they can be encoded in tuples or any different simple data structure.

This strategy is called Forward Accumulation in AD. The name comes from the fact that the expressions accumulate their gradients as they add more operation or function and

Expression	Symbolic value	Numerical value
$\frac{\partial E_1}{\partial y}$	$2x\frac{\partial x}{\partial y}$	0
$\frac{\partial E_2}{\partial y}$	$\frac{\partial E_1}{\partial y}y + E_1\frac{\partial y}{\partial y}$	4
$\frac{\partial E_3}{\partial y}$	$e^x\frac{\partial x}{\partial y}$	0
$\frac{\partial E_4}{\partial y}$	$\frac{1}{y}\frac{\partial y}{\partial y}$	$\frac{1}{3}$
$\frac{\partial E_5}{\partial y}$	$\frac{\partial E_3}{\partial y} + \frac{\partial E_4}{\partial y}$	$\frac{1}{3}$
$\frac{\partial E_6}{\partial y}$	$\frac{\partial E_2}{\partial y} + \frac{\partial E_5}{\partial y}$	$\frac{13}{3}$

Table 2.5: Computing the derivative of f with respect to the variable y using AD.

become more complex.

A drawback of Forward Accumulation is that it requires to keep the derivatives of all possible variables while computing the expression, or another way to see the process is that it should be repeated or recomputed for each variable. Therefore, Forward Accumulation is not recommended when the target function has numerous variables.

In order to deal with this problem, AD defines another strategy called Backward Accumulation. As its name suggests, this strategy accumulates the value of the expressions and derivatives from the final expression to the underlying variables.

Backward Accumulation exploits the properties of the chain rule to compute the derivatives. Note that the derivative $\partial f/\partial x$ can be rewritten as $\partial f/\partial E\ \partial E/\partial x$ for any expression E. In our example, we can start with E_6 as the expression we are differentiating the target function, and then using $E_5, E_4, \ldots$ up to E_1.

Table 2.6 presents the results of these computations for the same example defined previously.

Note that this process is independent of the final variable that we want to differentiate the target function. Also, note that as our expressions are composed of only one operation or function, the application of the chain rule regards the expression where they are defined.

Chain rule has to be applied again to calculate the derivative of the function with respect to a variable. In this case, we need to check all the expressions that concern the variable and use the previous values to compute that. For example, the following are the operations needed to calculate the derivative for x.

Expression	Symbolic value	Numerical value
$\frac{\partial f}{\partial E_6}$	1	1
$\frac{\partial f}{\partial E_5}$	$\frac{\partial f}{\partial E_6}\frac{\partial E_6}{\partial E_5}$	1
$\frac{\partial f}{\partial E_4}$	$\frac{\partial f}{\partial E_5}\frac{\partial E_5}{\partial E_4}$	1
$\frac{\partial f}{\partial E_3}$	$\frac{\partial f}{\partial E_5}\frac{\partial E_5}{\partial E_3}$	1
$\frac{\partial f}{\partial E_2}$	$\frac{\partial f}{\partial E_6}\frac{\partial E_6}{\partial E_2}$	1
$\frac{\partial f}{\partial E_1}$	$\frac{\partial f}{\partial E_2}\frac{\partial E_2}{\partial E_1}$	y = 3

Table 2.6: Backward computation of the derivative of f using AD.

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial E_3}\frac{\partial E_3}{\partial x} + \frac{\partial f}{\partial E_1}\frac{\partial E_1}{\partial x} = 1 * e^x + 3 * 2x = e^2 + 12 \tag{2.24}$$

Using this procedure, we got the same value that Forward Accumulation obtains. Similarly, we can compute the derivative for y.

$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial E_4}\frac{\partial E_4}{\partial y} + \frac{\partial f}{\partial E_2}\frac{\partial E_2}{\partial y} = 1 * \frac{1}{y} + 1 * x^2 = \frac{13}{3} \tag{2.25}$$

As expected, we get the same value. In this case, computing the derivative to another variable does not requires gradient recomputation for all the expressions. In fact, all the values of Table 2.6 are reused in both instances. This property allows us to have a significantly more efficient algorithm to compute the derivatives when the target function has several variables.

AD methods have gained interest over recent years, especially in the Machine Learning area [59], because they have been proven to be efficient and effective alongside Gradient Descent techniques for optimization. In a typical supervised learning problem, hundreds or thousands of variables, called model parameters, define the output value of the classification or regression. This case is suitable to apply Backward Accumulation and Gradient Descent to optimize these variables. In particular, artificial neural networks use the back-propagation algorithm to tune their parameters. In terms of AD, back-propagation is just an instance of a Backward Accumulation problem.

Chapter 3

Proposed Model

In this chapter, the proposed model is presented. We will describe it according to the three main tasks the model performs: classifying an instance, optimizing the model to get the best results, and interpreting and explaining the predictions/classification (understanding why a sample is classified in a particular category).

3.1 Classification

3.1.1 Dempster Shafer Implementation

This model uses the elements of the Dempster-Shafer Theory to implement a classification model; the elements and way the uses them are described below:

Mass assignment functions (MAF) are represented as unidimensional tensors (i.e., vectors) with $k+1$ values, where k is the cardinally of the set of all possible classes a sample can belong to. The vector contains one value for each singleton and another value one for the complete set. Mass for the null set is omitted because it is always 0. Masses for subsets having other combinations of classes are also omitted. This decision means that our model supports only general uncertainty, instead of specific uncertainty as the theory indicates. The main reason for this choice is that the power set grows exponentially with the number of classes, which implies that all computations would involve exponential-length vectors. Then the complexity of the model would have been exponential too. The trade-off solution proposed here includes the mass value for the complete set, which is used to measure the uncertainty, but keeps the length of the vector linear with the number of classes of the problem.

$$\text{MAF: } (m_1, m_2, \ldots, m_k, m_U) \tag{3.1}$$

To illustrate this solution, consider a disease detection problem where a model has to determine if a specific disease is present (P) or absent (A) in a patient. In this problem,

there are two classes P and A, and then a MAF M over this frame of discernment, assigns values to the P singleton, A singleton, and the complete set, for example, if M assigns these values

$$\begin{aligned} M(\phi) = 0, \quad M(\{P\}) = 0.5 \\ M(\{A\}) = 0.2, \quad M(\{P, A\}) = 0.3 \end{aligned} \tag{3.2}$$

The corresponding vector to represent M is the following one:

$$M : (0.5, 0.2, 0.3) \tag{3.3}$$

Since MAFs have a finite number of values, we can represent them as tables as well. Figure 3.1 presents the representation of two MAFs omitting the null set, which is always 0.

X	m(X)
A	0.2
B	0.3
A,B	0.5

X	m(X)
A	0.55
B	0.18
A,B	0.27

Figure 3.1: Example of two MAFs represented as tables

Methods to compute belief and plausibility for each outcome given a mass assignment function are implemented in the model using the formulas presented in 2.5 and 2.6. Note that for a single outcome, the belief is just the mass of the singleton. Therefore no computation is needed to obtain this value. The plausibility is computed just as the sum of the mass value of the singleton for the class (m_i) and the mass for the complete set (m_U) since our method does not consider subsets that are combinations of outcomes except for the complete set.

$$\begin{aligned} \text{Bel}(m, i) &= m_i \\ \text{Pl}(m, i) &= m_i + m_U \end{aligned} \tag{3.4}$$

In our example, the belief and plausibility for each class is $Bel(M, P) = 0.5$, $Bel(M, A) = 0.2$, $Pl(M, P) = 0.5 + 0.3 = 0.8$ and $Pl(M, A) = 0.2 + 0.3 = 0.5$.

A method for computing the Dempster rule between two mass assignment functions m_A, and m_B is provided by the model using the Dempster Rule presented in Equation 2.11. This

method returns the result in a tensor with the same characteristics as the inputs. Then the output can be used directly as a new mass assignment function in the module.

Due to the representation of mass assignment functions, the sum of the Dempster Rule formula has at most $k+1$ elements. For each singleton, this expression is the sum of each singleton of one MAF with the uncertainty of the other MAF, and the product between singleton masses. The resulting mass for the uncertainty of the combined MAF is simply the product of the uncertainty of each MAF. The formula is presented below:

$$\begin{aligned} m_A &= (m_{A1}, m_{A2}, \ldots, m_{Ak}, m_{AU}) \\ m_B &= (m_{B1}, m_{B2}, \ldots, m_{Bk}, m_{BU}) \\ m_A \oplus m_B &= K'(m_{A1}m_{B1} + m_{A1}m_{BU} + m_{AU}m_{B1}, \\ &\quad \vdots \\ &\quad m_{Ak}m_{Bk} + m_{Ak}m_{BU} + m_{AU}m_{Bk}, \\ &\quad m_{AU}m_{BU}) \end{aligned} \tag{3.5}$$

Where K' is the normalization term in Dempster Rule. Note that K' can be computed afterward as the reciprocal of the sum of the vector elements.

For example if we have $m_A : (0.5, 0.2, 0.3)$ and $m_B : (0, 0.3, 0.7)$ then the combination of these MAFs is the following:

$$\begin{aligned} m_A \oplus m_B &= K'(0.5 * 0 + 0.5 * 0.7 + 0.3 * 0, \\ &\quad 0.2 * 0.3 + 0.2 * 0.7 + 0.3 * 0.3, \\ &\quad 0.3 * 0.7) \\ &= K'(0.35,\ 0.29,\ 0.21) \end{aligned} \tag{3.6}$$

As explained above, K' can be computed as $K' = (0.35{+}0.29{+}0.21)^{-1} = (0.85)^{-1} = 1.176$, and then the final result for the combination is the following:

$$\begin{aligned} m_A \oplus m_B &= K'(0.35,\ 0.29,\ 0.21) \\ &= (0.412, 0.341, 0.247) \end{aligned} \tag{3.7}$$

Moreover, we can see how the combination rule applies when MAFs are represented as tables. Figure 3.2 shows an example of the combination of two mass functions using the Dempster Rule (the left and the middle table) producing the last MAF table. In this figure, the lines indicate which values of the original MAFs are multiplied to obtain the combination, and their color indicates to which set this multiplication contributes.

Figure 3.2: Example of the combination of two MAFs represented as tables

3.1.2 Rules

Dempster-Shafer Theory is often applied in the design of expert systems because it is a good method to combine knowledge originated from various sources which may not be related to each other and even contradictory. For example, various stakeholders participating in a complex decision-making process [60]). These systems usually use rules or hypotheses to express conditions for the evidence of the problem. For example, in disease detection problems a rule could be "if a patient has high blood pressure then she/he is more likely to have a stroke"; in this case the condition is to have high blood pressure and the evidence is to be more likely to have a stroke.

Concerning the classification process for the model, in a general case, a classifier tries to find the class to which an observation belongs. The observation encodes information in the form of a vector X, often called the feature vector. For example, in disease prediction problems, samples could contain age, blood pressure, past diseases, etc. Then, a rule can be mathematically defined as a pair (m, s) that relates a mass assignment function m, with a predicate or boolean function s (i.e., a function that evaluates to true or false) with domain in the feature space.

Figure 3.3 shows a visualization of a rule using the tables for MAFs. In this case, the representation is the same, but adding the statement at the top of the table.

Using this representation, rules defined by experts are still possible to be used in this context. However, this model aims to perform automatic classification based on the evidence presented by suitable data. The method itself should create the rules. In order to do it, an algorithm to generate rules from the available data is required; this method, called rule generator, creates rules using only training data and defined parameters. Some of the most basic rule generators are presented below, and they were implemented in the model.

3.1.3 Support

Rule statements can consider any number of attributes. When a statement has more than one attribute, they can be joined using any logical operations, i.e., the conjunctive and disjunctive

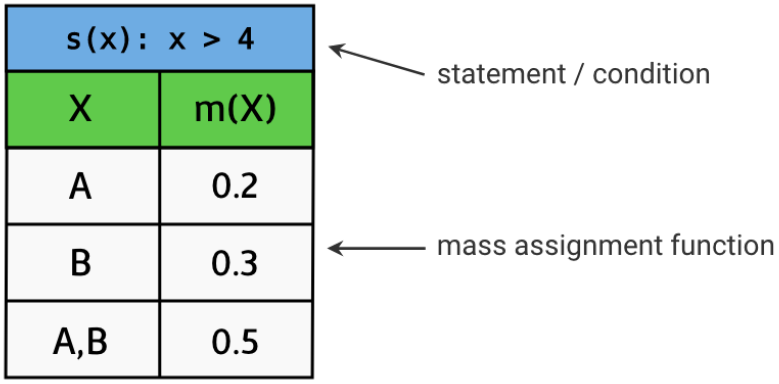


Figure 3.3: Example of a Rule

combinations.

Note that disjunctive combination makes a statement to have the form $s(x) : c_1 \vee c_2$, where c_1 and c_2 are other simpler conditions related to the input x. This structure implies that the rule will be applied either when c_1 or c_2 is true. In our model, this kind of statements can be split into two rules each one concerning one part of the disjunction, i.e., having $s_1(x) : c_1$ and $s_2(x) : c_2$ as separated rules. Applying this procedure, the prediction becomes more accurate because each condition will have an independent MAF associated.

Therefore, rule statements regarding two or more attribute are useful only if they apply conjunctive combinations between them. However, when using disjunctive combinations, rules statements become more specific and intricate, and then fewer records can satisfy these statements. If the number of data records are low, this could make the model overfit these values. In order to address this problem, we introduce the support indicator. The support of a rule is defined as the proportion of the data records that satisfies the rule. Usually, the training set is used to estimate the support. If the support of a rule is low, e.g., less than 5%, we can state that this rule is very likely to overfit.

In order to understand better how the support of a rule is related to the number of disjunctive combinations of attributes, we can perform the following experiment: Create a random dataset of size N with k attributes. For all data records, each attribute value is a sample from a random uniform distribution between 0 and 3. Then, we can compute the support for the rules: $s_1(x) : x_1 > 2$, $s_2(x) : x_1 > 2 \wedge x_2 > 2$, ..., $s_k(x) : x_1 > 2 \wedge x_2 > 2 \wedge \cdots \wedge x_k > 2$. The intuition of this definition of rules is that s_1 only uses one attribute, s_2 uses two attributes, and so on. The range 0 and 3 and the condition "greater than 2" are established using the assumption that for an attribute, we are interested in separate it into three groups with an equal number of records.

The chart of Figure 3.4 shows the results of this experiment for $N = 1000$ and $k = 10$.

We can observe that there is an evident decay in the support when the rules concern more attributes. The expected result for the experiment above is to obtain a geometric distribution for the support. And thus, an exponential decay for the support with respect to the number

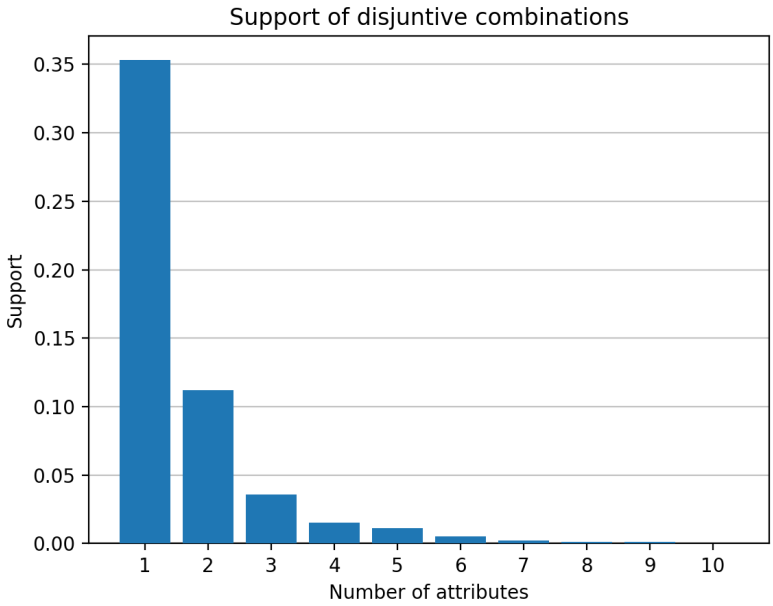


Figure 3.4: Support of rules with disjuntive combinations of attrbiutes

of attributes.

From this analysis, we can conclude that when a rule concerns three or more attributes, it is below the range of 5% that we defined above, and thus, these rules are likely to overfit and should not be considered in the model.

If we consider another range other than 0 and 3 or another condition, the support chart will change. However, it will never invalidate the exponential decay behavior; thus, our conclusion is still valid. This result still holds for distributions different from uniform; for example, a normal distribution also presents exponential decays when combining attributes using conjunctions.

This analysis was presented because, in the following sections, we will propose automatic rule generators. These generators are algorithms that automatically generate a set of rules to be used by the model. By the results discussed, the generators should consider at most two attributes in order to prevent overfitting.

3.1.4 One-attribute Rule Generators

The most simple rule generator is to assume that attributes of the feature vector are independent, and then the rule statements can consider only one attribute at the same time. Considering this, rules for attributes can be generated in the following way:

- If the attribute is categorical, then a rule for each possible category can be generated, the rule just checks that the value of this attribute belongs to the corresponding class. If Q represent the set of all categories for the attribute x_i in the feature vector, the

statements are defined by:

$$s(x) : x_i = q_j \quad \forall q_j \in Q \tag{3.8}$$

For example, if a categorical attribute x_i of a feature vector indicates whether a person smokes having the categories "Smoker" and "Non-Smoker", then we create a rule covering all possible options, i.e., the first rule is $s(x) : x_i =$ "Smoker" and the second rule is $s(x) : x_i =$ "Non-Smoker".

- If the attribute is continuous, we can compute the mean and the variance of the values, and then the domain of the attribute can be partitioned in ranges that statistically contain the same number of samples or ranges that are equally distributed. If $q_0, q_1, \ldots, q_n$ represent the break values for the ranges, the following rules are generated:

$$\begin{aligned} &s(x) : x_i < q_0 \\ &s(x) : q_i \leq x_i < q_{i+1} \\ &s(x) : x_i \geq q_n \end{aligned} \tag{3.9}$$

Figure 3.5 shows an example of rule generation for a continuous variable using two breaks and the strategy of getting equal number of samples in each range.

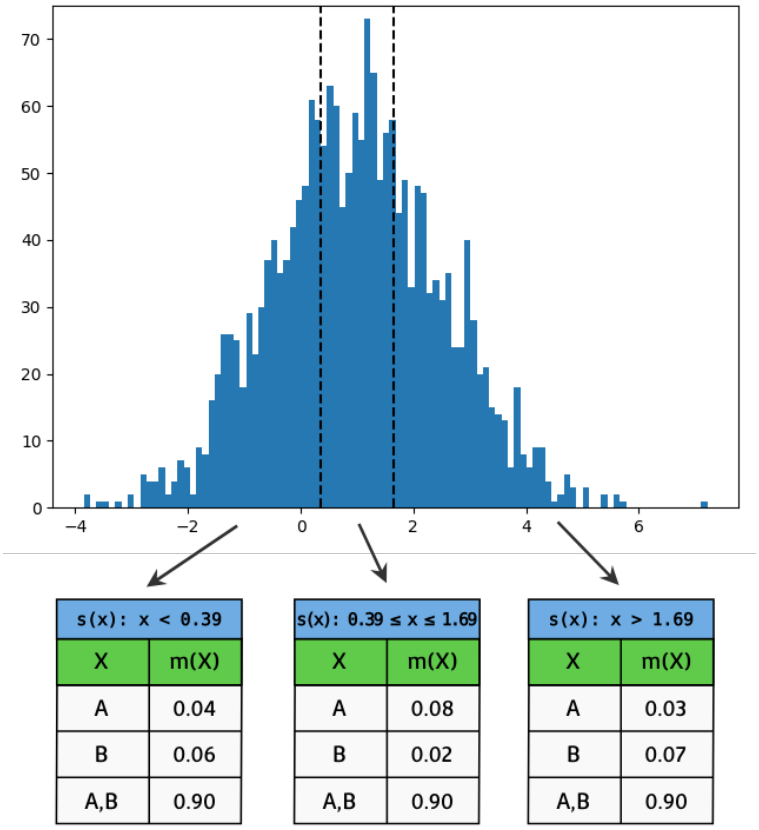


Figure 3.5: Example of the statistic simple rule generator for a continuous variable using two breaks

3.1.5 Two-attribute Rule Generators

A helpful two-attribute rule generator can be defined when two continuous characteristics are outside a particular range. For continuous attributes, we can create rules that captures the behavior when two variable takes extreme values. This kind of rules makes sense in many real case scenarios. As an example of these rules, consider a screening of certain diseases problem. In this problem, when many symptoms are outside their typical values, the disease can be detected, e.g., high body temperature and high levels of sugar in the blood.

Mathematically, let x_i and x_j to be continuos attrbiutes, let l_i and l_j to be the lower "normal" bounds for attribute i and j, and let u_i and u_j to be the upper "normal" bounds for attribute i and j. Then, four rules can be created, which are the following:

$$\begin{aligned} s_1(x) &: x_i < l_i \wedge x_j < l_j \\ s_2(x) &: x_i < l_i \wedge x_j > u_j \\ s_3(x) &: x_i > u_i \wedge x_j < l_j \\ s_4(x) &: x_i > u_i \wedge x_j > u_j \end{aligned} \tag{3.10}$$

These statements represent all the possible combinations when these two attributes are both outside their normal ranges.

Similarly, we can create rule statements for conditions like having a variable in the normal range and the other outside its range or having the two variables inside their normal range.

These rule generators define the conditions for the rules. However, we need to set the initial values for MAFs as well. A possible strategy to set these values is to use prior knowledge about the problem. Again this knowledge can be obtained from experts or automatically, e.g., by using statistical measures of the training set [47]. Since these values will be optimized afterward, another valid strategy is to consider that at the beginning, we do not know anything about the problem, and then MAFs should have high uncertainty. In fact, we can set the value 1 for the complete set and 0 for the rest, and then this assignment expresses full uncertainty. However, using this full uncertainty MAF as the initial state is useless because the model cannot distinguish the classes (all of them have the same values), and this is required to perform the first classification. Alternatively, a better solution is to assign a high value for the complete set, e.g., 0.9, and the remaining 0.1 is distributed among the other singletons randomly. For a binary classification problem, a MAF with values $(0.04, 0.06, 0.9)$ could be a possible initial MAF for a rule.

3.1.6 Prediction

The model operates using a rule set RS defining the knowledge, regardless of the source (experts or automatically), and the input feature vector x describing the features of the sample we want to classify. The model performs the following algorithm to obtain the predicted class.

1. Apply the predicate of each rule in the rule set using the feature vector x as input. Filter out the rules that do not satisfy the predicate.

2. Combine the mass assignment function of the selected rules using Dempster Rule, obtaining a combined mass assignment function.

3. Compute the estimated probability for each class from the combined mass assignment function using any of the estimation presented in section 2.5.3. The predicted class will be the class with the maximum estimated probability. Regardless of the estimation used, the class with the maximum estimated probability is the one that has maximum value for belief of that class.

Mathematically if RS represents the rule set and x the input, the mass set for x, M_x, and the predicted outcome $\bar{y}$ can be defined as follows:

$$\begin{aligned} M_x &= \{m \mid (m, s) \in \text{RS} \wedge p(x)\} \\ m_f &= \bigoplus_{m \in M_x} m \\ \bar{y} &= \underset{class}{argmax}\ \text{Bel}(m_f) \end{aligned} \tag{3.11}$$

Figure 3.6 and Algorithm 3.1 summarize the process of classification.

Algorithm 3.1: Model prediction

Input: Feature vector x and Rule Set RS
Output: Predicted class y' for the feature vector

```
M ← Empty list ;
foreach Rule (m_i, s_i) ∈ RS do
    if s_i(x) is True then
        Add m_i to M ;
    end
end
m ← Full uncertainty mass assignment function ;
foreach m_i ∈ M do
    m ← DempsterRule(m, m_i) ;
end
y' ← argmax Belief(m) ;
return y' ;
```

Besides knowing the predicted class, in many cases, we want to obtain the probability our model assigns to each class. In this case, we need to return a vector, indicating the estimated probability for each class. This process is particularly important for the optimization process discussed later.

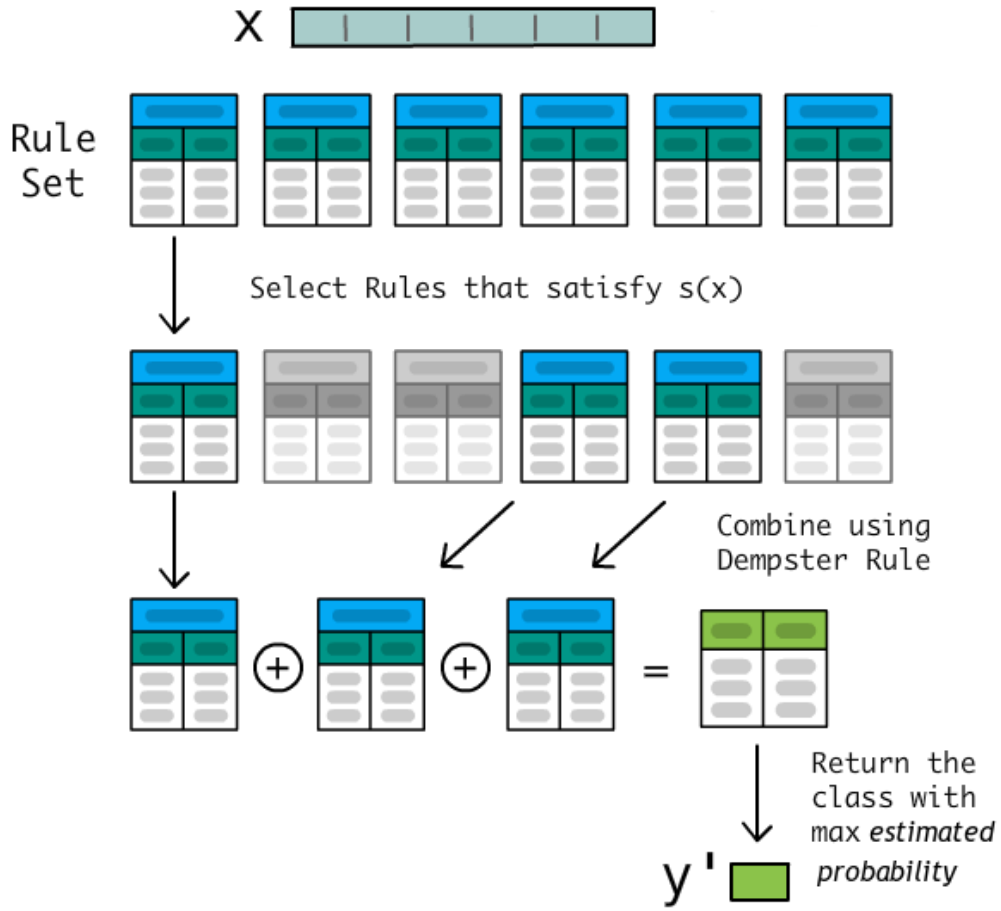


Figure 3.6: Classification Process in DS Model. Starting from a feature vector x and a rule set, the model selects the rules that satisfy the predicate, the MAF of these rules are combined using Dempster Rule, and then the predicted class is the one with maximum belief.

3.2 Optimization

In order to get the best results for the model, it is necessary to fit the mass values of each rule according to the training data.

To do this task, a loss function and a method of optimization should be used to update mass values. Loss functions measure the error a model obtains in predictions comparing the predictions with the actual outcomes. In an optimization problem, the model aims to minimize the error computed by the loss function and an algorithm of optimization is used to accomplish it. The existing literature reports on various optimization methods that can be applied depending on the nature and structure of the problem. In our case, we opted for gradient descent as an optimization method because it has been widely used successfully in other machine learning methods like artificial neural networks.

3.2.1 Loss Functions

One of the loss functions the model implements is Mean Squared Error (MSE), which is the average of the square of the euclidean distance between the predicted outcome and the real outcome. Since categories often do not have a numeric representation, the model should operate with the results one-hot encoded, i.e., that the output will be a $k-$dimensional vector where each value indicates the belonging to each class. This vector is compared to the vector of the estimated probability of each class generated by the prediction process. The formula for MSE is presented below:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n} ||y_i - \bar{y_i}||^2 \tag{3.12}$$

Where n is the number of samples, y_i is the true class for the element i and $\bar{y_i}$ is the predicted class for the element i. Both y_i and $\bar{y_i}$ may be one-hot encoded.

The model also implements Cross-Entropy (CE) [61] as another loss function. Cross-Entropy Loss is the average of the entropy of the distribution of the real class using the predicted outcomes. It is given by the following formula:

$$\text{CE} = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{k} y_{ij} \cdot \log(\bar{y_{ij}}) \tag{3.13}$$

Where n is the number of samples, k is the number of different classes, y_{ij} is 1 if the true class of the element i is j, or 0 otherwise, and $\bar{y_{ij}}$ is the predicted probability for the element i to belong to class j.

Both functions are normally used as loss functions in other classifiers like neural networks. Also, both functions are differentiable, so they can be used as target functions in gradient-based optimization.

3.2.2 Gradient Descent

In this thesis, we opted for gradient-based methods for optimization of parameters, often called first-order optimization algorithms [51], because they have a more direct and clear mathematical justification of reaching optimal values than other methods that only use target function values. Also, gradient-based methods are preferred over Second-Order Optimization Algorithms, such as Newton Method or other more complex optimization methods since gradient-based methods are easy to compute, require less time to perform, and they converge fast on large data sets [52].

The gradient descent and any of its variations algorithm is an iterative algorithm that optimizes variable values in order to minimize or maximize the value of a target function, as

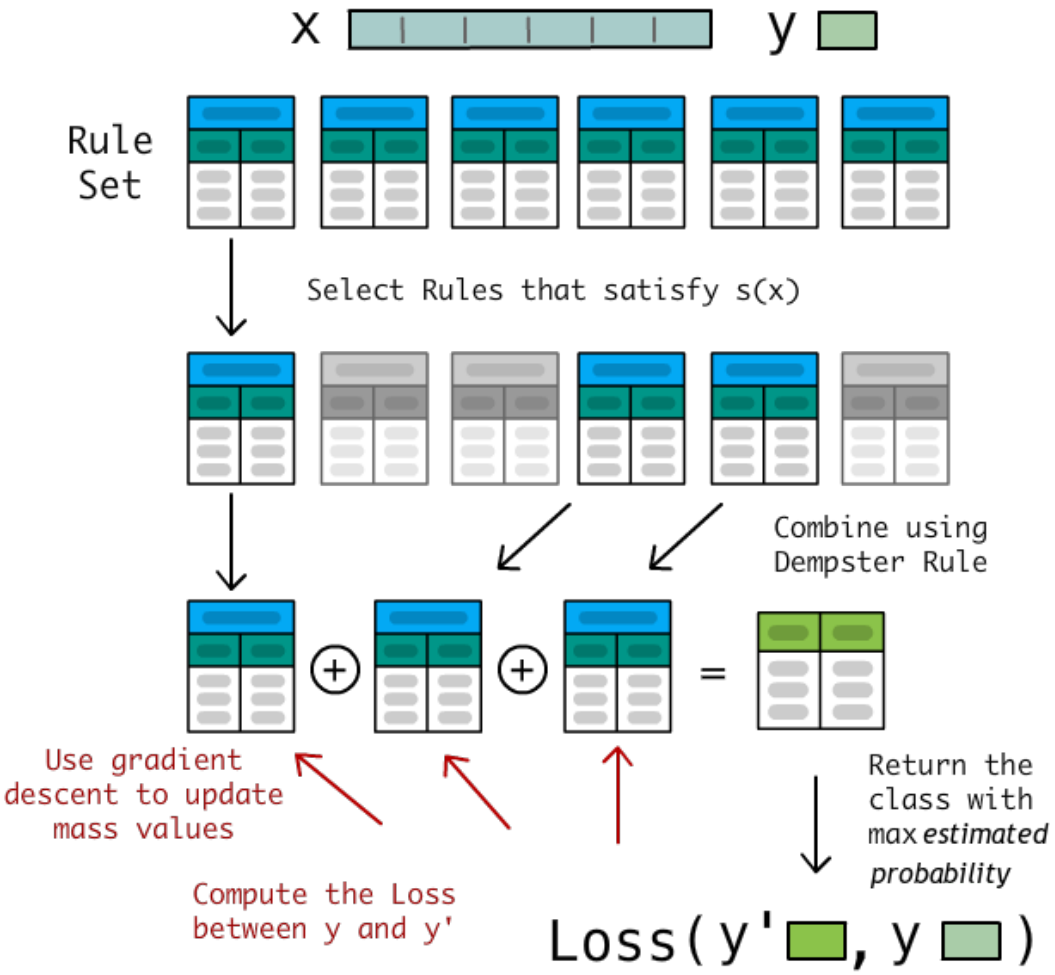


Figure 3.7: Optimization Process in DS Model

explained in section 2.6.

In our case, the target function is any of the mentioned loss functions, and the variables to be optimized are the mass values for each rule. The next formula shows the optimization for the mass $m^{(i)}$

$$m_{t+1}^{(i)} = m_t^{(i)} - \alpha \frac{\partial \text{Loss}}{\partial m^{(i)}} \tag{3.14}$$

In the implementation of the model, mass values are defined as symbolic variables rather than simple values, allowing the computations of the gradient automatically using numerical differentiation techniques.

Besides traditional gradient descent algorithm, the model supports any variation of gradient descent methods such as Stochastic Gradient Descent (SGD) and Adam [56]. The first value for the learning rate is a hyper-parameter of the model.

3.2.3 Projection

Recalling the restrictions of mass assignment functions from the definition 2.4, the method of gradient descent presented in equation 3.14 cannot be used because after updating the values, these may not satisfy the restrictions. For example, after updating, some mass values could become negative or the sum of the masses could exceed 1.

To solve this problem, it is important to notice that our optimization task is not unrestricted. For optimization with restrictions, an equivalent method to gradient descent is presented in [62], called projected gradient descent. It consists of the same idea of gradient descent but projecting the values on the domain where variables are defined after updating. The same work proves that this method converges to the optimum for the restricted problem. Therefore, the correct formula for updating the mass values is the following:

$$m_{t+1}^{(i)} = \pi_C \left(m_t^{(i)} - \alpha \frac{\partial \text{Loss}}{\partial m^{(i)}} \right) \tag{3.15}$$

Where π is the orthogonal projection function, and C is the set of masses that satisfy Dempster-Shafer constraints. Another important aspect to consider is that projected gradient descent works with any of the variations of gradient descent that use adaptive learning rate or momentum such as SGD or Adam.

We will define π_C in our particular case in order to implement it in our model correctly. First, note that C can be defined mathematically as the following equation.

$$\begin{aligned} C: \ & m_1 + m_2 + \cdots + m_k + m_U = 1 \\ & \text{where } m_1 \geq 0, m_2 \geq 0, \ldots, m_k \geq 0, m_U \geq 0 \end{aligned} \tag{3.16}$$

Geometrically, C defines an hyperplane on the positive region of space. We can also define the normal vector of that hyperplane as a vector containing only 1s i.e. $\vec{N} = (1, 1, \ldots, 1)$. For any arbitrary point $\vec{x} = (x_1, x_2, \ldots, x_n)$, the line that passes through x and is parallel to N can be defined parametrically as the following:

$$\begin{aligned} L(t) &= \vec{N} * t + \vec{x} \\ &= (1, 1, \ldots, 1) * t + (x_1, x_2, \ldots, x_n) \\ &= (x_1 + t, x_2 + t, \ldots, x_n + t) \end{aligned} \tag{3.17}$$

Now, we can find the orthogonal projection of x over C by computing the intersection between L and C. From this intersection, we can obtain the value of t that must be added to the vector x to be projected.

$$
\begin{aligned}
x_1 + t + x_2 + t + \cdots + x_n + t &= 1 \\
\sum_{i=1}^{n} x_i + nt &= 1 \\
t &= \frac{1 - \sum_{i=1}^{n} x_i}{n}
\end{aligned}
\tag{3.18}
$$

The Figure 3.8 shows a geometrical example of the projection of the vector $M = (0.4, 0.5, 0.7)$ over C resulting in the vector $M' = (0.2, 0.3, 0.5)$. Note that using the formula for t presented above, we obtain $t = -0.2$ which is the value added to M to transform it into M'.

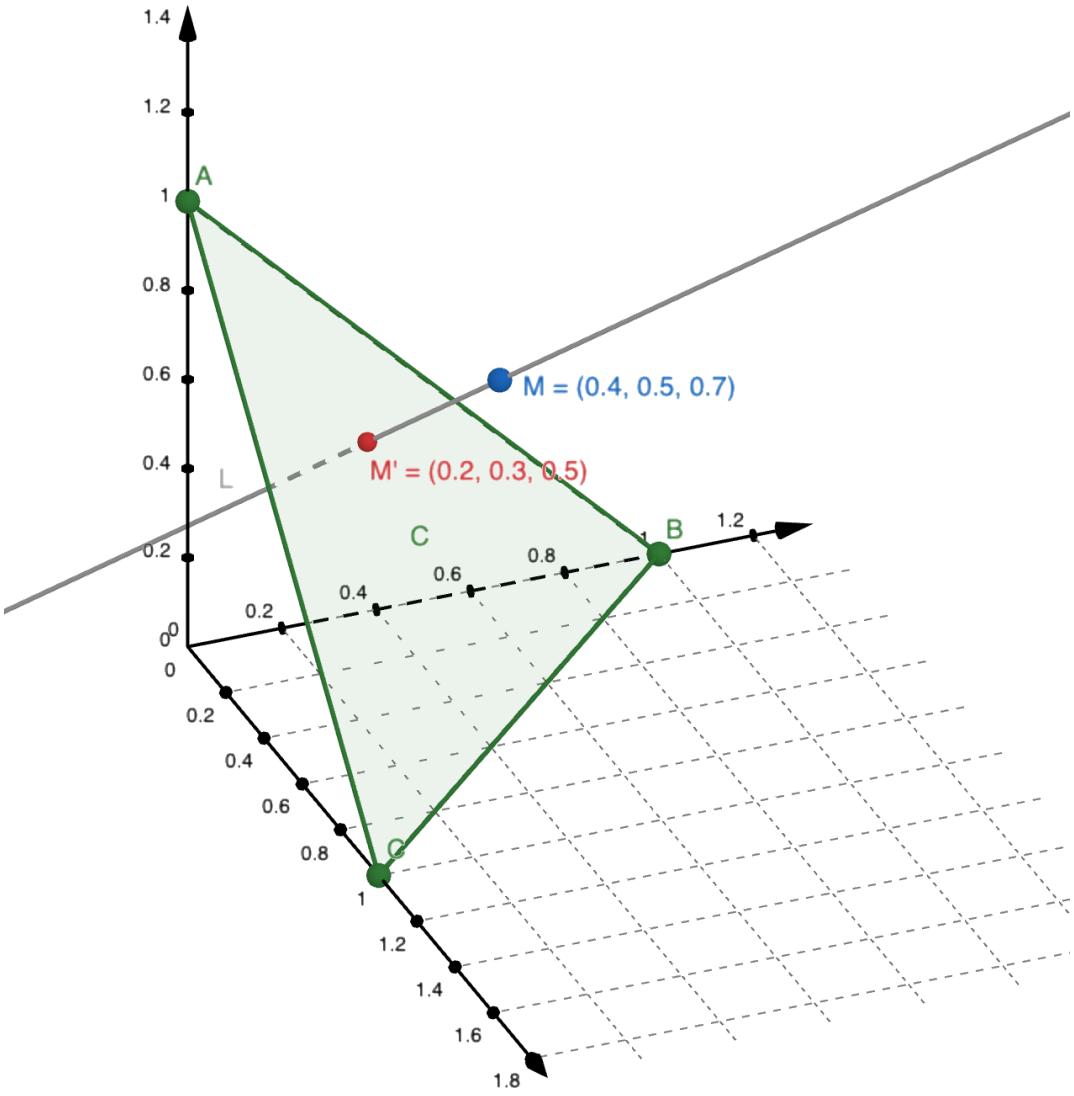


Figure 3.8: Projection of masses process example.

Finally, using the result presented above, the normalization process can be written as the

following algorithm.

Algorithm 3.2: Projection of masses

Input: Mass Vector x Unnormalized
Output: Projected Mass Vector m'

$s \leftarrow \text{Sum}(x)$;
foreach $x_i \in x$ **do**
  $x_i \leftarrow x_i + \frac{1-s}{n}$;
end
return x

3.2.4 Convergence Conditions

The model processes the complete training set several times in order to adjust the mass values correctly. Each iteration of the model within the training process is called an epoch. The next step is to define a condition to stop iteration, so we can state that the model has converged.

The condition used in the model consists of evaluating the difference between the loss in two adjacent epochs; if this difference in absolute value is smaller than a threshold ε, then it is said that the model has converged and it stops iterating. Mathematically, it is defined by:

$$|Loss_t - Loss_{t-1}| \leq \varepsilon \tag{3.19}$$

In order to control the convergence under unexpected cases (e.g., where the condition above may be unreachable), the model also accept two hyper-parameters to control iteration: `max_iter` and `min_iter`. These values define the maximum and minimum number of epochs that the model can iterate.

3.2.5 Batching

Working with large datasets (i.e., datasets that have many records) usually adds complexity and requires more computational resources because many parts of the training process need to examine all these records generating data structures and operations that are proportional to the number of records being processed. Depending on the characteristics of the dataset and the hardware where our model runs, handling large data could be not feasible.

To solve this problem, the model allows working with large datasets using the batching strategy [53]. That consists in separating the dataset in subsets called batches with a fixed size, and use them to evaluate the model predictions and loss computation.

As explained in section 2.7.1, a kind of Stochastic gradient descent method in mini-batch

gradient descent, which has properties such as it is easy to compute and converges fast and loss do not oscillate due to outliers.

To implement batching in our model, all we need to do is divide the training set into subsets of records called batches. Then, we perform the standard training process (see Algorithm 3.3) but passing a batch instead of the entire training set. All of the rest of the training process remains the same.

3.2.6 Pseudo-code for the optimization process

A summary of the training and optimization process is presented below as a pseudo-code.

Algorithm 3.3: Optimization process

Input: Batch of Feature vectors X, their corresponding classes Y, the Rule Set RS, a Loss function *Loss* and a Optimizer update function *Optim*
Output: Rule Set with updated mass values RS'

$RS' \leftarrow$ Empty Set ;
$Y' \leftarrow$ Empty List ;
foreach $x \in X$ **do**
  $y' \leftarrow \text{Predict}(X, RS)$;
  Add y' to Y' ;
end
$L \leftarrow \text{Loss}(Y, Y')$;
foreach *Rule* $(m_i, s_i) \in RS$ **do**
  $m'_i \leftarrow \text{Optim}(L, m_i)$;
  $m'_i \leftarrow \text{Project}(m'_i)$;
  Add (m'_i, s_i) to RS' ;
end
return *RS'*

3.3 Interpretability

Let us recall that one of the primary purposes of the classifier is to be interpretable. In this section, we will explain why interpretability is a first-class citizen of this model.

Rules, as we defined in the implementation, are the combination of a mass assignment function and a predicate or statement. After the training and optimization phase, mass values for each rule have changed, and they have converged to the optimal values for prediction. From an interpretability point of view, that means that the model "learned" how much a rule statement is contributing to the prediction and which outcome the statement predicts; mass values can be analyzed to distinguish informative rules from redundant rules.

For example, in the disease detection problem, if after training, a rule with predicate "if blood pressure is high", ends up with the following values using our representation $m = (0.7, 0.08, 0.22)$, that means the mass is 0.7 for the P singleton, it is 0.08 for the A singleton, and it is 0.22 for the complete set. Then, this rule contributes to predict that the disease is present, since the mass of P singleton is much higher than the corresponding one for the A singleton. Also, the mass of the complete set shows the uncertainty for the rule (0.22) is low. We can then state that high blood pressure is related to the presence of the disease.

Therefore, interpretability and knowledge discovery are directly extracted from the analysis of the mass values of a rule after training.

Furthermore, it is possible to define "contribution score" functions that given a mass assignment function m and a class k returns a value that represents how much important or contributory is the mass to the prediction of a certain class. A possible definition of a contribution score function could be the geometric mean between the singleton class and the complement of the uncertainty, we will call this function γ and it is mathematically defined by the following expression.

$$\gamma(m, k) = \sqrt{m_k(1 - m_U)} \tag{3.20}$$

The function γ is bounded between 0 and 1. A value of 0 means no contribution, and a value of 1 indicates full certainty to the predicted class. Moreover, the function γ satisfies many properties that make it a good indicator of prediction contribution. The most important are listed below.

- Let m be a full uncertainty mass, i.e. $m_U = 1$ and $\forall k,\ m_k = 0$, then $\forall k,\ \gamma(m, k) = 0$. This property establishes that no matter which class is being tested, a contribution of a full uncertainty mass is always 0.
- Let m be a mass function that has full evidence to the class k, i.e. $m_k = 1$, $m_U = 0$ and $\forall i \neq k,\ m_i = 0$, then $\gamma(m, k) = 1$ and $\forall i \neq k,\ \gamma(m, i) = 0$. This property means that for a full evidence mass function to certain class, if we ask for the contribution of this class γ returns 1, i.e. the rule assigns all mass to that class. If we ask for another class distinct from k then there is no contribution.

The function γ will be used in this work for measuring interpretability. However, any other score function can be used.

Using this contribution score or any other, we can sort all the rules of the classifier by their contribution scores and know exactly which rules are the most important for the prediction of each class.

The support of a rule is another important aspect to consider when interpreting the results of the model. Support is defined as the number of dataset elements that satisfy the rule normalized by the dataset size. Rules with low support may not generalize data correctly since they are only a few specific cases. For example, if a rule is satisfied only by one element and this element is from class A, then the model will report that this rule is very contributory to A class prediction. But it is not correct to state that because only one case satisfies this condition and then we cannot generalize knowledge about A class prediction.

3.4 Improvements and Implementation notes

Besides the theoretical behavior of the model explained in the previous sections, many procedures were rewritten to decrease its computational effort and thus obtain a more efficient solution. Every improvement applied to the implementation is discussed in this section. Most of these improvements are valid because of the assumptions we made before, then they may not be applicable to other DST implementations.

3.4.1 Rule predicate precomputation

The first step for predicting the class of a record is to obtain which rules the record satisfies. In the training phase, a record is examined many times while the model parameters adjust their values. In this process, the rules that this specific record satisfies does not change over the iterations. Then, we can store or "remember" which are these rules instead of computing the predicate of all rules again.

In order to do this, we need a data structure to store these values indexed by a record identifier. Then, the first time we examine a record, we need to compute the satisfied rules and save them, and the next times we tested the same record, we only have to look up the stored value in the data structure.

In the worst case, all records satisfy all rules, and then the data structure will use a space of order $O(nr)$ where n is the number of samples, and r is the number of rules. Depending on the dataset size and the number of rules used, this optimization may not be possible.

3.4.2 Dempster Rule vectorization

The Dempster Rule is the most costly operation in our model, even when we decided to simplify the uncertainty measurement, it is still the heaviest computation in terms of arithmetics operations. The formula for combining two MAFs presented in Equation 3.5 gives the steps for the implementation of this operation. However, implementing this operation manually using for loops and standard scalar multiplications is worse than using vectorization and linear algebra operations. Because the latter ones are implemented in libraries and frameworks that allow performing these computations very efficiently using parallelism and mathematical properties to reduces their cost.

In order to vectorize the operation, note that all the values except the last one have the same form. This form can be expressed as the sum of vector point to point multiplication and scalar multiplications to vectors. Mathematically it can be expressed as the following formula:

$$m_A \oplus m_B = m_A * m_B + m_{AU} m_B + m_{BU} m_A \tag{3.21}$$

In order to solve the problem with the value of the uncertainty (the last value of the vector), note that if we applied the above procedure, the resulting value is $m_{AU}m_{BU} + m_{AU}m_{BU} + m_{BU}m_{AU} = 3m_{AU}m_{BU}$. In other words, the resulting value is the triple of the desired value. Then, we can correct this computation by merely dividing this value over 3.

Finally, the following procedure is equivalent and more efficient to the Equation 3.5:

$$\begin{aligned} m_A \oplus m_B &= m_A * m_B + m_{AU}m_B + m_{BU}m_A \\ m_{A\oplus BU} &\leftarrow \frac{1}{3}\, m_{A\oplus BU} \end{aligned} \tag{3.22}$$

3.4.3 Commonality transformation

As we anticipated in section 2.5.5, commonality transformation is another way to express MAFs that helps reduce the cost of Dempster Rule. In this section, we will explain how this transformation is used in our model.

The commonality function $q(A)$ is defined as the amount of mass committed to A and the super-sets of A. In our simpler representation of uncertainty for mass functions, we can compute the commonality of a set by adding its mass and the mass of the uncertainty. Mathematically it can be expressed as:

$$q_i = m_i + m_U \tag{3.23}$$

For example, if we used the mass m_A and m_B used in the Equation 3.5, their equivalent values for the commonality functions will be:

$$\begin{aligned} q_A &:(0.8, 0.5, 0.3) \\ q_B &:(0.7, 1, 0.7) \end{aligned} \tag{3.24}$$

Recalling the Equation 2.17, which states that the commonality of a combined expression is the multiplication of the commonalities of the single components, the result for the combination of A and B in our example can be computed as follows:

$$\begin{aligned} q_A \oplus q_B &= K'(0.8 * 0.7, 0.5 * 1, 0.3 * 0.7) \\ &= K'(0.56, 0.5, 0.21) \end{aligned} \tag{3.25}$$

Note that this expression has a lot few computations in comparison with Equation 3.6.

Then, we can transform back the commonality of the combination to their mass values by subtracting the value of the uncertainty. In our example, we will have:

$$
\begin{aligned}
m_A \oplus m_B &= K'(0.56 - 0.21, 0.5 - 0.21, 0.21) \\
&= K'(0.35, 0.29, 0.21) \\
&= (0.412, 0.341, 0.247)
\end{aligned} \tag{3.26}
$$

This procedure obtains the same result as the standard process (see Equation 3.7).

At first glance, it is not clear whether this transformation and the new process to combine masses produces a real reduction in the computational cost of our model. However, this transformation allows doing other optimizations, which are explained below.

3.4.4 Commonality and normalization

The multiplicative behavior of the commonality functions when applying the Dempster Rule allows making improvements in the normalization. The last step in the Dempster Rule is the normalization of the values to comply with the constraints of the mass definition.

We will prove that, due to the new way of computing the combination rule, it is not necessary to normalize the values in the intermediate steps.

Let us suppose that after the application of the Dempster Rule with commonality functions, we have a commonality with the form $q_A = K(a_1, a_2, \ldots, a_k, a_u)$ where K is the normalization factor, and let $q_B = (b_1, b_2, \ldots, b_k, b_u)$ another commonality function which we want to combine with the previous one.

Following the standard formula for the combination we obtain:

$$
\begin{aligned}
q_A \oplus q_B &= S(Ka_1b_1, Ka_2b_2, \ldots, Ka_kb_k, Ka_Ub_U) \\
q_A \oplus q_B &= SK(a_1b_1, a_2b_2, \ldots, a_kb_k, a_Ub_U)
\end{aligned} \tag{3.27}
$$

Where S is the new normalization factor that by definition is the reciprocal of the sum of the values, Mathematically,

$$
\begin{aligned}
S &= (Ka_1b_1 + Ka_2b_2 + \ldots Ka_kb_k + Ka_Ub_U)^{-1} \\
&= K^{-1}(a_1b_1 + a_2b_2 + \cdots + a_kb_k + a_Ub_U)^{-1}
\end{aligned} \tag{3.28}
$$

For simplicity, let us call the sum $(a_1b_1 + a_2b_2 + \cdots + a_kb_k + a_Ub_U)$ as K_2, then $S = K^{-1}K_2^{-1}$. If we replace S on the expression for the combination, then we have

$$
\begin{aligned}
q_A \oplus q_B &= SK(a_1b_1, a_2b_2, \ldots, a_kb_k, a_Ub_U) \\
&= K^{-1}K_2^{-1}K(a_1b_1, a_2b_2, \ldots, a_kb_k, a_Ub_U) \\
&= K_2^{-1}(a_1b_1, a_2b_2, \ldots, a_kb_k, a_Ub_U)
\end{aligned} \tag{3.29}
$$

Note that after doing this process, the initial factor K is canceled out, and it is not needed anymore. K_2, which is the new normalization factor depends only on the original values and not the normalization factor.

This finding implies that we can combine by multiplication all the commonality functions without normalizing the result obtained in the intermediate steps. Therefore, many of the computations used for normalization can be omitted, and thus, the model becomes more efficient.

3.4.5 Plausibility probability estimation and commonality

The final step of the prediction in our model is to estimate the probability for each class. One of the estimators presented is the plausibility estimator, which, in combination with the commonality transformation, allows implementing a new improvement.

First, note that the equations for the commonality (Equation 3.23) and plausibility (Equation 3.4) in our implementation are the same, suggesting that for each set A, $Pl(A) = q(A)$.

Then, the equation for the plausibility estimation proposed by Cobb and Shenoy [35] (Equation 2.10) can be rewritten in terms of commonality values as follows:

$$P(a) = \frac{q(\{a\})}{\sum_{b \in \Omega} q(\{b\})} \tag{3.30}$$

This change implies that we do not need to transform back the values of the commonality to mass values, and we can finish the prediction process using the transformed values.

Furthermore, the fact that we can finish the prediction using the transformed values implies that the values for the uncertainty are not needed. And then, they can be dropped just after applying the transformation, avoiding many redundant computations to be done.

3.5 Model Complexity

A description of the model computational complexity is presented in this section. The three main parts of the model will be analyzed: Prediction, Training, and Interpretability.

For all these cases et X to be the set feature vectors with size n and m attributes for each vector, let k to be the number of classes and let RS to be the set of rules with length r.

3.5.1 Prediction

In order to predict all feature vectors of X, then n single predictions must be performed. For a single prediction, first, we need to check which statements of all rules in RS should be applied, this can be done in $O(r)$. Then, Dempster Rule is applied to the selected rules, which have a complexity proportional to the length of the mass vector, i.e., it is $O(k)$, therefore applying r times the Dempster Rule is $O(rk)$. Finally, as this process is repeated n times, the final complexity for the prediction is $O(nrk)$.

3.5.2 Training

The training process has an additional variable that is the number of epochs, let E be this number. In one epoch, we perform n single training processes; each of these single processes has a single prediction phase, which is $O(rk)$ as described above. Then, the gradients are computed, and the values are updated. This is proportional to the number of mass values to be optimized, which is $O(rk)$. Finally, the masses are projected to satisfy the constraints, which again is $O(rk)$. This training is repeated E times, which implies the complexity of the whole training process is $O(Enrk)$.

3.5.3 Interpretability

Once the model is trained, compute the contribution scores for each rule is iterate over all rules (r times) and compute the score for each class (k times), this can be done in $O(rk)$. After that, we can sort these values according to each class. Sorting is $O(rlog\ r)$, then k sorts can be done in $O(rklog\ r)$, which is also the complexity of this part.

For all the processes analyzed, note that if we had used all the uncertainty subsets for Dempster Shafer Implementation, Dempster Rule would have been $O(2^k)$ instead of $O(k)$. We can also estimate the size of RS (i.e., the value of r) if the model uses one of the automatic generators of rules. For example, if the model uses the single-attribute rule generator, then the number of rules is proportional to the number of attributes, then r is $O(m)$. And if the model uses the two-attribute rule generator, the number of rules is proportional to the number of pairs of attributes that can be formed, which are proportional to m^2, then r is $O(m^2)$.

Chapter 4

Results

In this chapter, experiments and results of the proposed model are presented. The main aim of this chapter is to prove that the proposed method is a valid classifier able to solve classification tasks correctly. In order to do this, the model will be tested on controlled scenarios where the classification output is known. Then, our model will be compared to other classification methods using traditional datasets that are popularly used in introductory courses of data mining and machine learning and are available in many websites, thus frequently used for benchmarking. Besides, we will present the interpretability results of some of the experiments to show this aspect as well.

4.1 Controlled Scenarios

The aim of testing the model in controlled scenarios is to prove that the output (classifications) and interpretability are correct. In this section, several experiments will be described, and results presented. Since we create these experiments, the expected results are known.

4.1.1 1-D distribution

The first experiment is one of the simplest possible classification task: we will have only one real-value variable x, which will be our unique attribute. We will have two classes "Red" and "Blue" which are defined by the sign of the attribute. Positive values are of the "Red" and negative values are "Blue". Five-hundred uniform random samples in the range $[-5, 5]$ will be generated using the random function form Python's random module to build a dataset, which we call A0. Figure 4.1 shows the distribution of this dataset, each sample is plotted as a vertical line, and their color indicates their class.

The model was tested over this dataset using the Mean Squared Error as loss function, and Adam algorithm as the optimizer with an initial learning rate of 0.025.

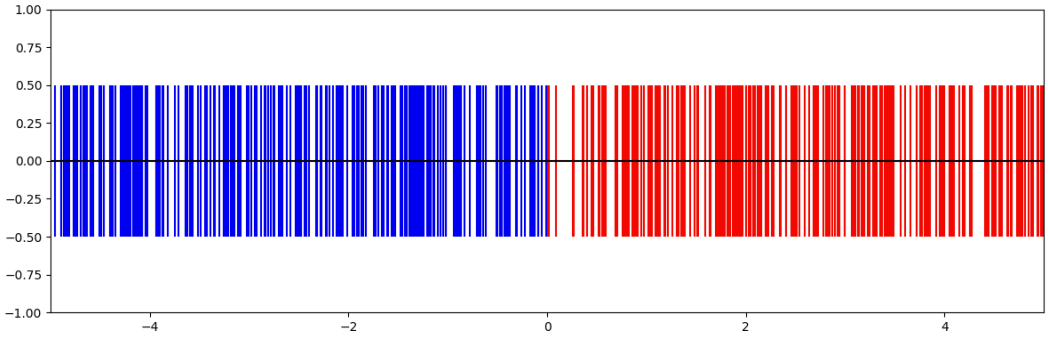


Figure 4.1: Distribution of the samples for the sign experiment.

The rules of the model will be directly related to the classes to be predicted using the following two rules:

1. Rule A, which has the statement $x \geq 0$.
2. Rule B, which has the statement $x < 0$.

As intended, Rule A is directly related to the "Red" class, and Rule B is directly related to the "Blue" class since their statements match the class formulation. Thus, our expected result is that Rule A assigns high mass to "Red" singleton, whereas Rule B assigns high mass to "Blue" singleton.

The experiment was performed, we used 80% of the dataset A0 for training, and the model was forced to be trained for 50 epochs. Figure 4.2 shows the value of the loss over the iterations of the model.

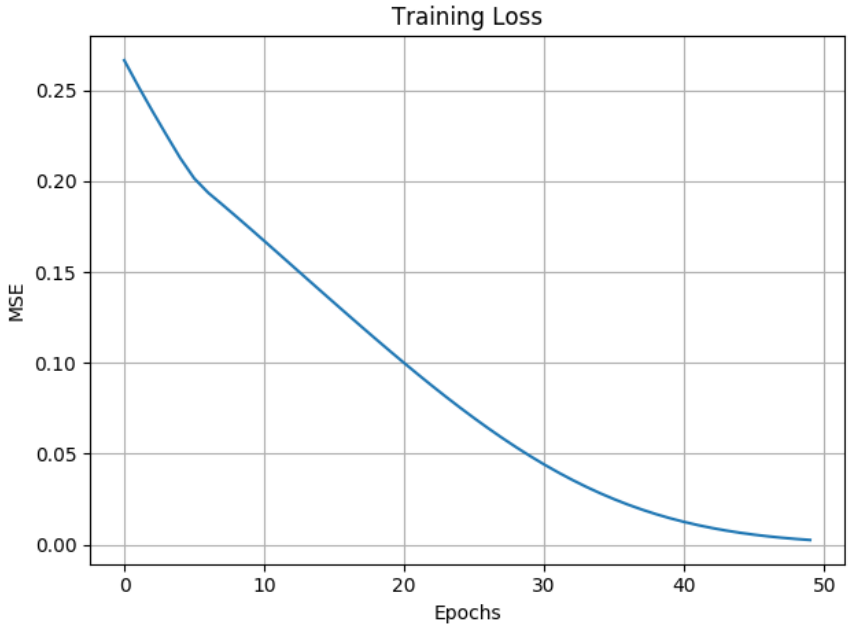


Figure 4.2: Training Loss of the model through epochs for the sign experiment.

Figure 4.3 shows the exact values of the mass assignment function over the iterations. The left chart shows values for Rule A and the right chart for Rule B.

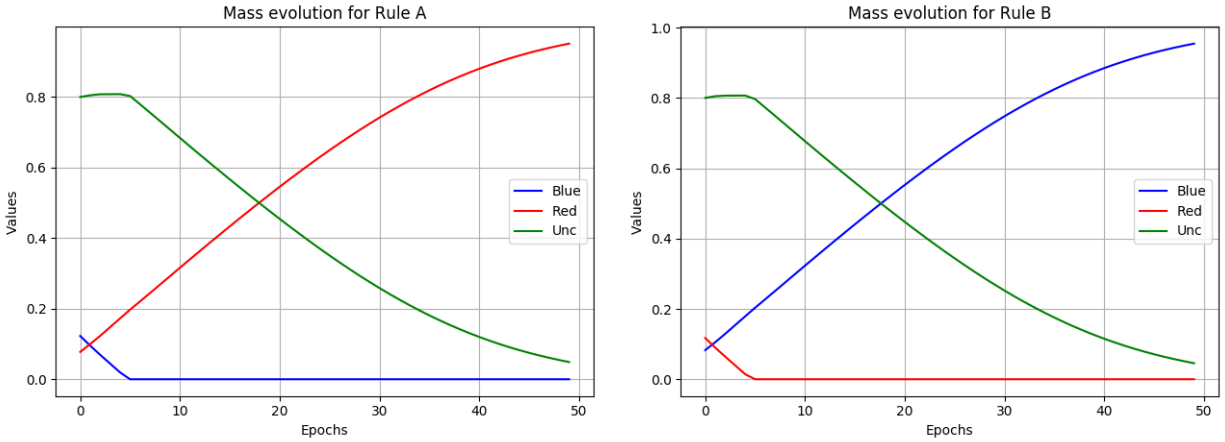


Figure 4.3: Mass values through epochs for rules A and B for the sign experiment.

From the analysis of these results, we can observe that the loss for this classification task strictly decreases through the epochs, and even it approaches 0, which means a perfect classification. Regarding the mass values for the rules, we can notice the experiment produces the results that were expected. Through the epochs, Rule A assigns more mass to the "Red" class, whereas the "Blue" class and the uncertainty were decreased. On the other hand, Rule B assigns high values of mass to the "Blue" class and lowers "Red" and uncertainty.

The trained model achieves an accuracy of 100% when requesting the classification of the 20% of the instances of dataset A0 used for validation. In fact, after the second epoch, the model reaches the perfect classification.

The experiment was repeated five times using different variations in the dataset A0 construction. In all these variations, the results were very similar, and the model always reaches the accuracy of 100%.

This experiment proves that our model can perform correct classifications when it has complete information about how to distinguish the classes. Besides, this experiment shows that the mass values for classes and uncertainty have the expected meaning, i.e., when a rule supports the classification of a particular class, the mass value of the mass function assigns high values to that class.

4.1.2 Full random classification

Another useful experiment to analyze is the case of an impossible classification task. A full random classification is defined as a problem where the target class is random; i.e., none of the attributes in any possible combination can predict the class correctly. This case is the opposite of the case presented previously since no classifier should be able to perform a correct classification. Thus, classifiers are expected to get accuracy similar to a random classifier.

We can create a dataset of full randomness as follows: take two independent samples from a uniform random distribution in the range from -5 to 5 and use one of them as the data attribute and the sign of the other to define the target class "Red" for positive and "Blue" for negative values. As they are independent samples, they do not relate to each other; thus, it is impossible to predict one from the other. Similarly to the previous case, 500 samples were created using this procedure, and they are used as our dataset for this experiment. Figure 4.4 shows the distribution of this dataset.

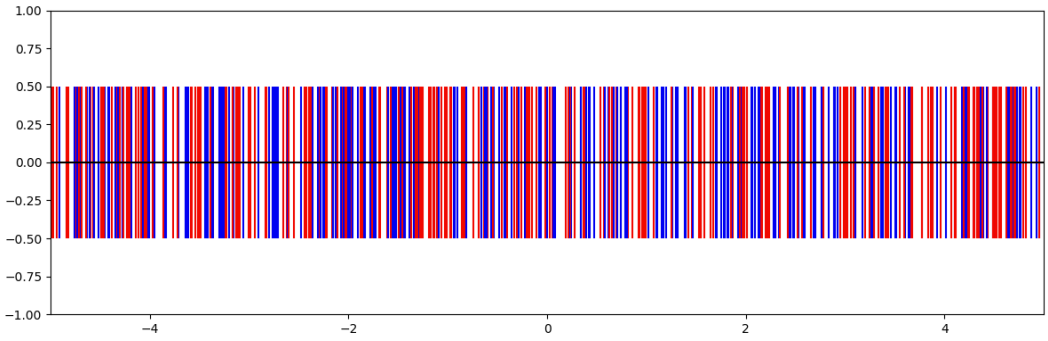


Figure 4.4: Distribution of the samples for the random classification experiment.

In this situation, any classifier is expected to reach an accuracy close to 50%, which is the same accuracy of a random classifier for a binary balanced classification task.

We use the same configuration for the model presented in the previous case, including the definition of the rules A and B, and the separation of the dataset in a training set (80%) and a validation set.

The experiment was performed, and again the model was forced to be trained for 50 epochs. The results of this experiment are presented in Figures 4.5 and 4.6.

For the case of the loss, we can see that it slightly decreases at the beginning, but after that, it remains constant with low fluctuations. The stationary value is around 0.25, which means that the model was not able to predict better than this loss level.

For the case of the mass distribution over epochs, we can observe that the mass of the singleton classes does not reach high values and tends to oscillate between 0 and 0.2; in both rules, the mass of the "Blue" class is greater than the mass of the "Red" class. The uncertainty value increases slowly and keeps always a high value.

When requesting the model to predict the validation set, it classifies all instances as "Blue". The model behaves in that way because, in the generated training set, there were imperceptibly more "Blue" than "Red" samples, and then the model learns that the best way to respond is to classify all of them as the majority. The model achieves an accuracy of 49.4% on the validation set, which is very close to 50%, which is the theoretical expected result.

Another aspect to note of this experiment is that the model was forced to perform 50 epochs. If we have used the convergence condition explained in section 3.2.4, the model would have stopped the training process at the third iteration because the changes in the

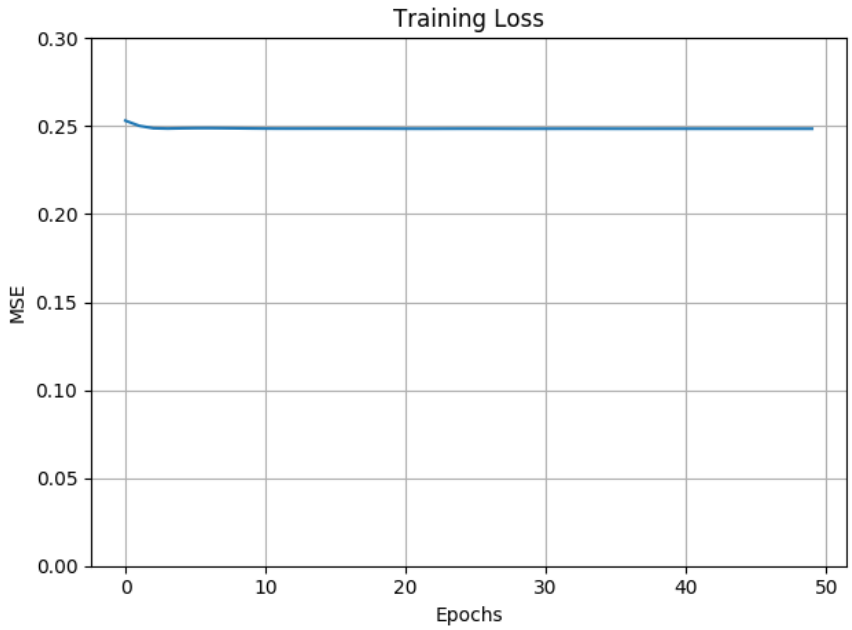


Figure 4.5: Training Loss of the model through epochs for the random classification experiment.

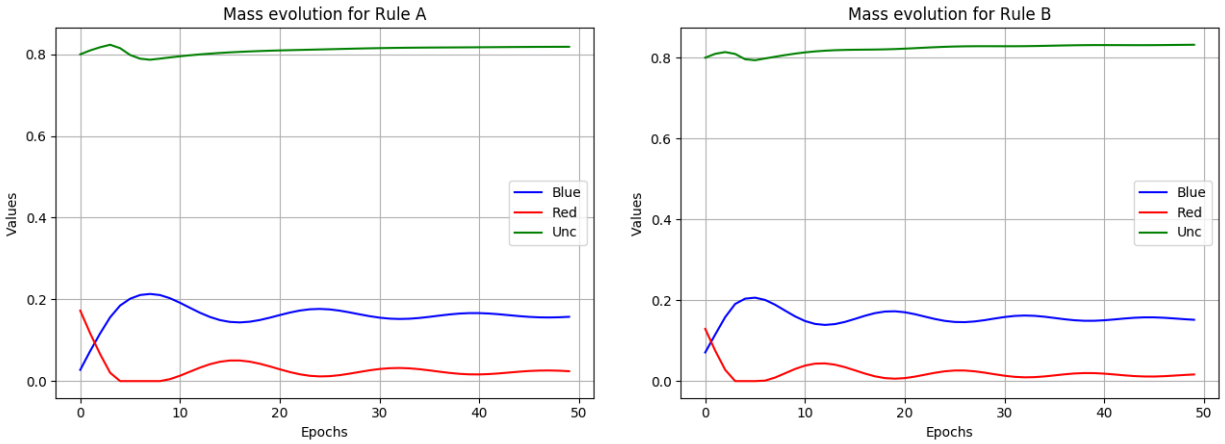


Figure 4.6: Mass values through epochs for rules A and B for the random classification experiment.

loss values between two contiguous epochs are minimal.

From this experiment, we can derive two ideas. First, we can verify once again that the mass function values have the expected meaning; in this case, the uncertainty takes high values, which is desirable since the model cannot perform a classification correctly, because the process is intrinsically uncertain. However, we observe that the model is more conservative increasing the value of the uncertainty than when increasing the value of the singletons when compared with the previous experiment. The second important observation about the model is its behavior of returning the majority class when the attributes do not provide any useful information. This behavior can help to discover trivial relationships. Still, it can bias the

classification when trying to find uncommon events, such as detecting a disease with a very low prevalence.

4.1.3 Imperfect classification

The last experiment concerning datasets with only one attribute will use the A0 dataset presented in the first experiment. In this case, we will change the rules in order to force the model to perform classification with error, but expecting that the error is as low as possible. For this case, we will use the following rules:

1. Rule A, which has the condition $x \geq 1$.
2. Rule B, which has the condition $x < -1$.
3. Rule C, which has the condition $-1 \leq x < 1$.

The difference between this case and the first one is that no rule can completely match the class definition. However, we can verify that samples that satisfy Rule A condition are always of the "Red" class, and the ones that satisfy Rule B are from the "Blue" class. The records that comply with Rule C condition can be either from the "Red" or "Blue" class.

With this in mind, the expected results are: Rule A assigns high mass to "Red" class, Rule B assigns high mass to "Blue" class, and Rule C assigns high mass to the uncertainty.

If the model follows the expected behavior, then we can observe that all values over 1 and below -1, which are 80% of the dataset should be classified correctly. The other values, which are the 20%, should be classified as a random classifier, which has an expected accuracy of 50%. If we combine these two results in one, we should expect a 90% of accuracy for the trained model over the dataset.

The experiment was performed using the same model configuration for 50 epochs and the same separation of training and validation set. The Figure 4.7 shows the values of the loss functions through iterations, and Figure 4.8 shows the values of the mass assignment functions for these three rules.

From the analysis of the results, we can observe that all three rules behave as we anticipated. Once again, the value of the uncertainty of Rule C is the one that increases slower.

Moreover, when requesting the model to predict unseen records from the validation set, it achieves an accuracy of 90.4% on average after repeating the experiment five times, which is very close to the expected value as explained before.

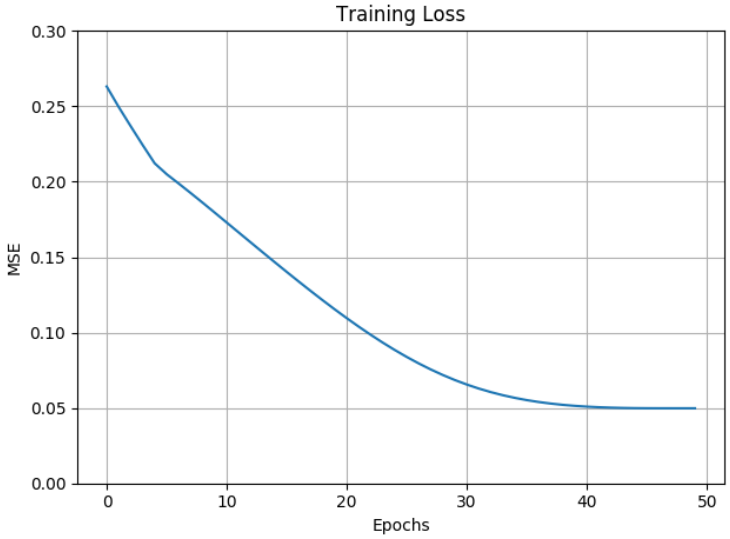


Figure 4.7: Training Loss of the model through epochs for the imperfect classification experiment.

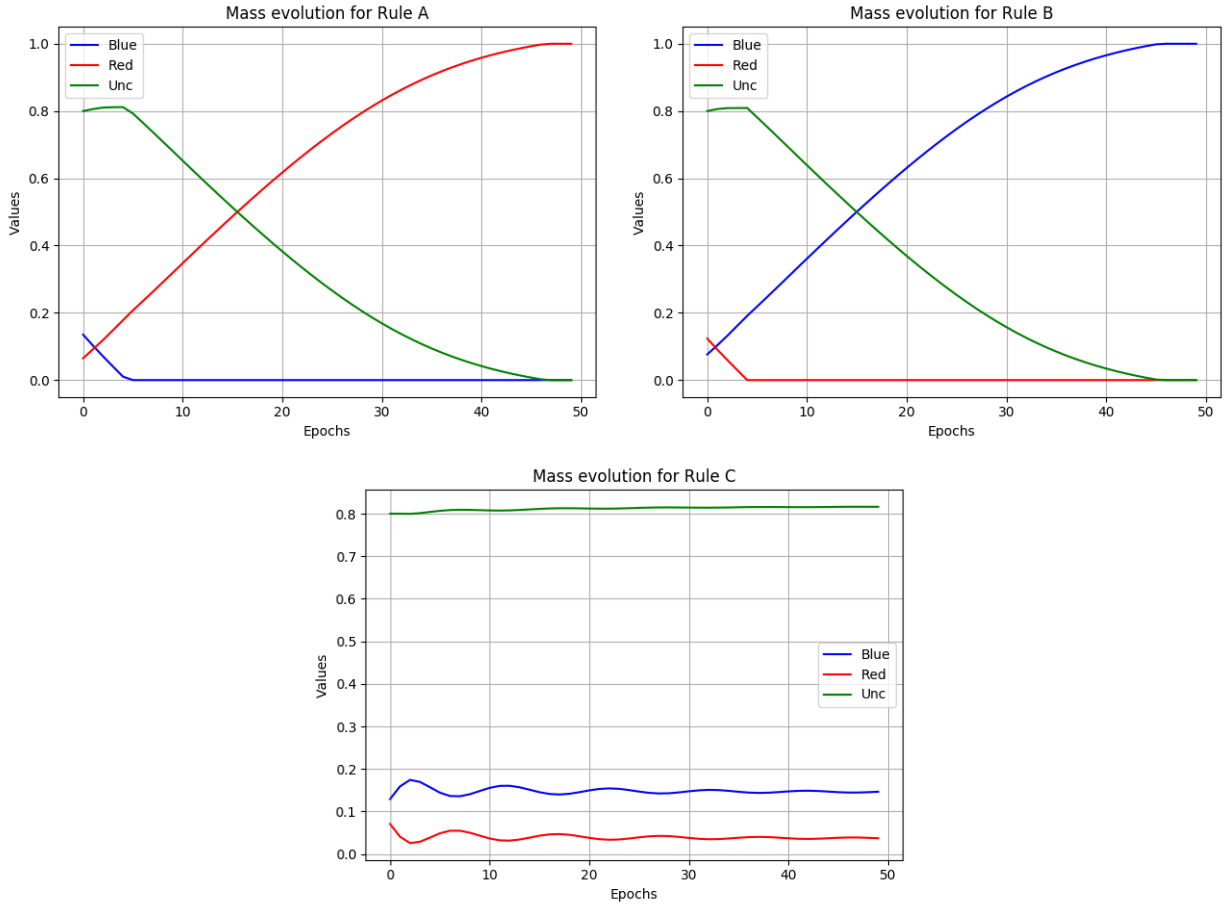


Figure 4.8: Mass values through epochs for rules A, B and C for the imperfect classification experiment.

4.1.4 2D distribution

The next experiments will test the model with more than one attribute. First, we will generate datasets using two attributes; these datasets have the property that can be plotted and therefore easier to visualize.

The first two-attribute dataset, called A1, contains 500 points, which are random uniformly distributed in the rectangle $[-1, 1] \times [-1, 1]$ the sign of the y component determines the class of each point. A point that has $y < 0$ belongs to the "blue" class, and the rest belong to the "red" class. Figure 4.9 shows the distribution of the dataset.

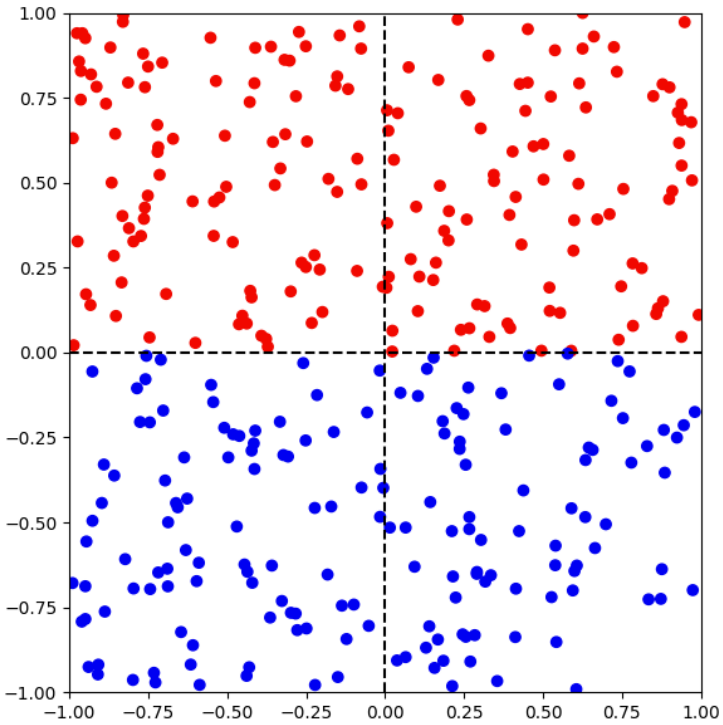


Figure 4.9: Distribution of the dataset A1.

Note that in this dataset, the variable y is informative since its sign defines the class of all records. In contrast, the variable x is useless because if we consider only that variable, we cannot perform the classification correctly. In fact, note that if the dataset is projected to the y-axis, then the distribution obtained is similar to the first 1D experiment. If we project the dataset to the x-axis, we get a dataset similar to the second experiment concerning a random classification. Then, it is expected that the rules regarding the x variable behave like the random experiment, and the rules regarding the y variable should act like our first experiment.

The model was configured using MSE as loss function and ADAM as the optimizer with a learning rate of 0.025. For validation, the dataset is split into a training set (70%) and a testing set (30%). The rules for the model are the following:

1. Rule A, which has the statement $x \geq 0$.
2. Rule B, which has the statement $x < 0$.
3. Rule C, which has the statement $y \geq 0$.
4. Rule D, which has the statement $y < 0$.

The model was tested over this dataset during 50 epochs. Figure 4.10 shows the loss of the classifier and Figure 4.11 shows the evolution of the mass values for all rules.

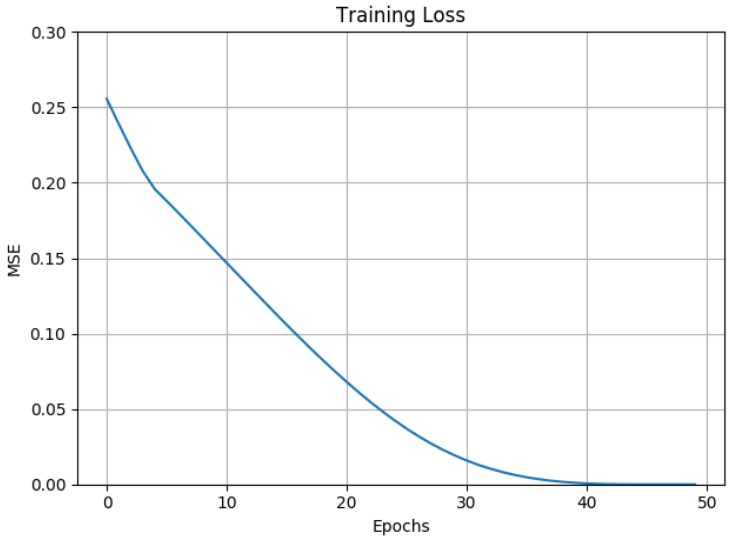


Figure 4.10: Training Loss of the model through epochs for the dataset A1.

In this case, we can observe that the model performs as expected. When testing the model over the validation set, it achieves an accuracy of 100%.

The experiments presented so far always used custom (or user) defined rules. In many cases, like this one, these rules help the model to achieve perfect classification. However, this user intervention is not possible acceptable if our aim is to develop a model that can perform automatic classification without any interference. In sections 3.1.2 to 3.1.5, this problem was discussed, and several rule generators methods were proposed which find the rules that help in the automatic classification process. In order to test them we will perform another experiment, which will use the same configuration and dataset A1, but instead of user defined rules we will let the method generate them using the statistical one-attribute rule generator with three breaks (see 3.1.4).

The rules the model generated in this experiment are the following:

1. Rule 1 with condition $x < -0.390$
2. Rule 2 with condition $-0.390 < x < -0.010$
3. Rule 3 with condition $-0.010 < x < 0.370$
4. Rule 4 with condition $x > 0.370$
5. Rule 5 with condition $y < -0.368$
6. Rule 6 with condition $-0.368 < y < 0.012$

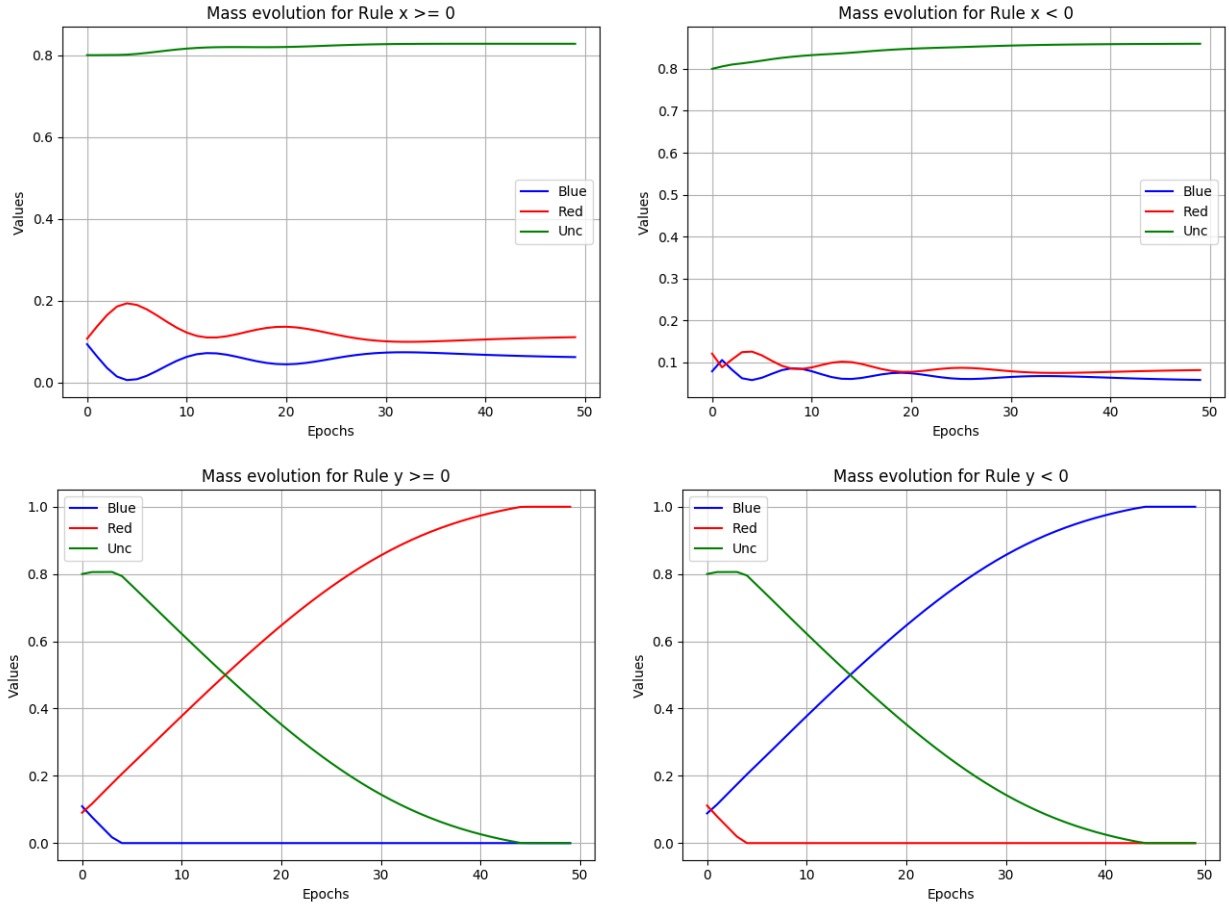


Figure 4.11: Mass values through epochs for all rules for the dataset A1.

7. Rule 7 with condition $0.012 < y < 0.393$
8. Rule 8 with condition $y > 0.39$

In addition to the configuration presented, we will not force the model to perform a certain amount of epochs. Instead, we will let the model control the iterations using the convergence conditions explained in section 3.2.4 using an ε of 0.0001.

The results for this variation of the experiment are presented in Figure 4.12 and 4.13. The first one shows the loss over the epochs; note that in this case, the algorithm decided to stop the training in the epoch 48, reaching a loss of 0.018. The second chart shows the result of the model prediction. In this chart, each dot is a record from the validation set, where their color indicates their class. The background color is the prediction of the model in that region. Darker colors mean higher certainty in prediction, and lighter means a high level of uncertainty. Although the rules of the model cannot capture exactly the definition of the class, the model still achieves an accuracy of 100% on the validation set.

Furthermore, Table 4.1 presents the final optimized rules (i.e., the values after the last epoch) the model obtains for the datasets A1. In this table, the bold values are the higher within the column, meaning that these rules are the most contributory to the prediction of that class.

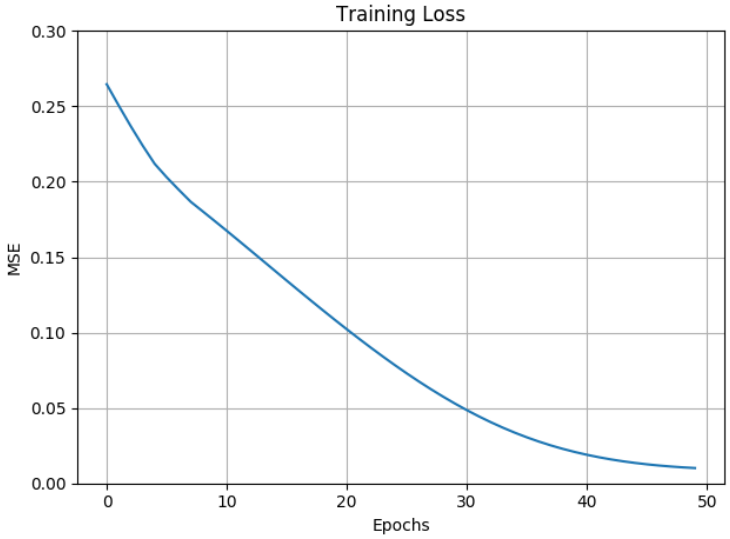


Figure 4.12: Training Loss of the model through epochs for the dataset A1 using generated rules.

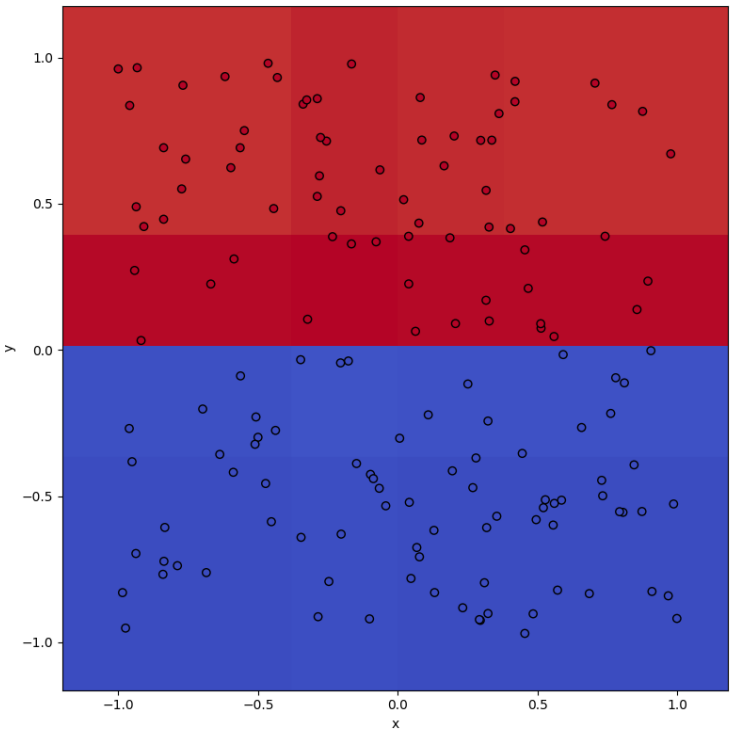


Figure 4.13: Model prediction results for dataset A1 using generated rules.

From this result, we can observe that the model performs a correct classification using rule generators. The most contributory rules where the ones that concern the y variable, while the rules concerning x variable had the highest uncertainty. However, in some cases, the generated rules cannot capture the real boundary that separates the classes, this is mainly due to the training data.

Rule	Mass Blue	Mass Red	Uncertainty
$x < -0.390$	0.000	0.291	0.708
$-0.390 < x < -0.010$	0.000	0.421	0.578
$-0.010 < x < 0.370$	0.000	0.343	0.656
$x > 0.370$	0.000	0.317	0.682
$y < -0.368$	**0.976**	0.000	0.023
$-0.368 < y < 0.012$	**0.969**	0.000	0.030
$0.012 < y < 0.393$	0.000	**0.945**	0.054
$y > 0.39$	0.000	**0.887**	0.112

Table 4.1: Resultant rules after model training for dataset A1

We replicated this experiment five times. Four of these tests obtain very similar results like the ones reported above, achieving an accuracy of 100%. In the other test, the model fails to classify one instance, getting an accuracy of 99.3%. This failure is due to the imprecision in finding the boundary for the y-attribute using the training set.

4.1.5 2D Gaussian distributions

The second two-attribute dataset to be tested aims at representing a more realistic scenario. In this case, instead of using uniform random distributions like in the previous cases, we will use two 2D Gaussian distribution to generate the dataset, which will be called A2. The dataset will be generated by the random function of the numpy library of Python. This distributions will have their mean value in different quadrants, for example, at the points $(-0.6, 0.1)$ and $(0.2, -0.8)$, and they will have a standard deviation of 0.25. The class of each record corresponds to the distribution that generates it. Figure 4.14 shows the two gaussian distributions with 250 samples each.

For this experiment, we will use the same configuration we used for the dataset A1, including the automatic rule creation by the one-attribute generator with three breaks, and the convergence condition with $\varepsilon = 0.0001$.

Similarly to previous cases, Figure 4.15 shows the loss function through epochs, and Figure 4.16 shows the results of the prediction of the model within the whole domain of the records. In this case, the model obtains an accuracy of 98.8% in the validation set (on average of five tests).

Table 4.2 shows the final values for the mass functions of all rules after the training process.

In the Figure 4.16 and Table 4.2, we can observe that all the rules that were important for the prediction of a class have a meaningful condition that captures points of the class they are contributory.

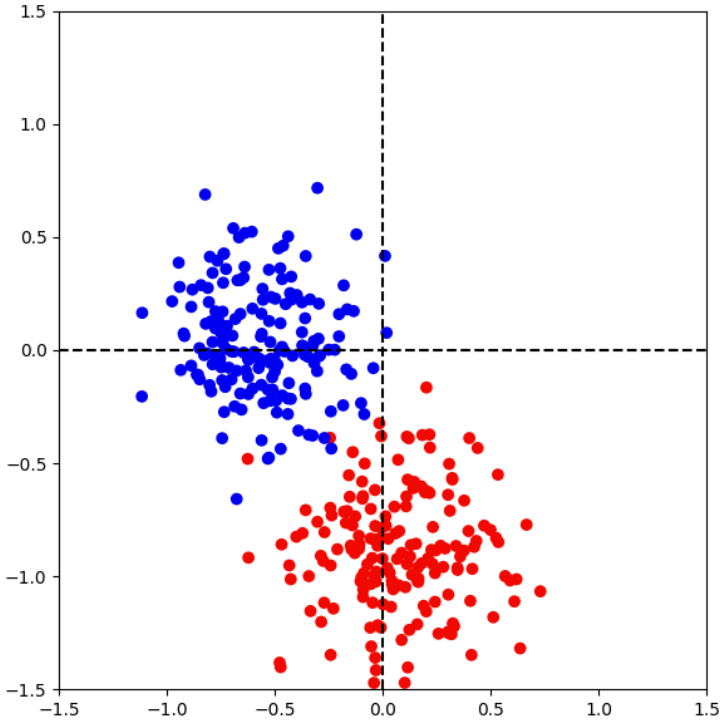


Figure 4.14: Distribution of the dataset A2.

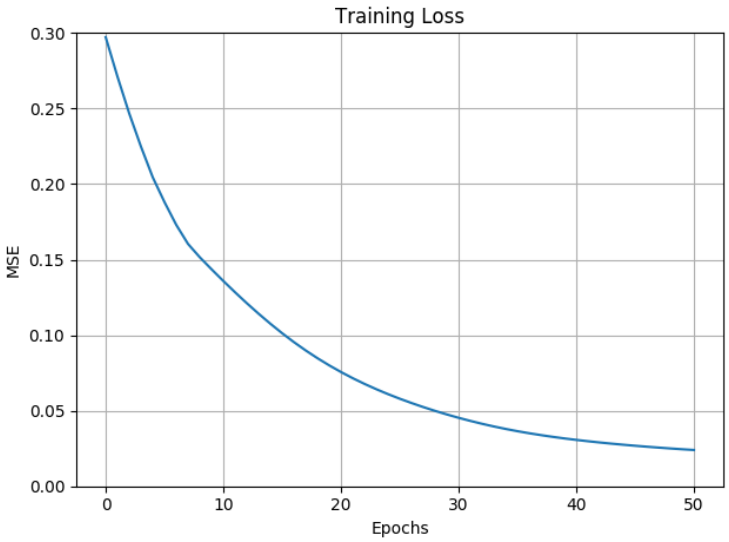


Figure 4.15: Training Loss of the model through epochs for the dataset A2.

4.1.6 Multi-class classification

All experiments presented so far were binary classification tasks. Our model can also handle multi-class classification problems. In this section, we present results for multi-class datasets. For that purpose, we generate a dataset using 2D Gaussian distributions, similar to the way A2 was generated, but changing the number of distributions. For example, we can build a

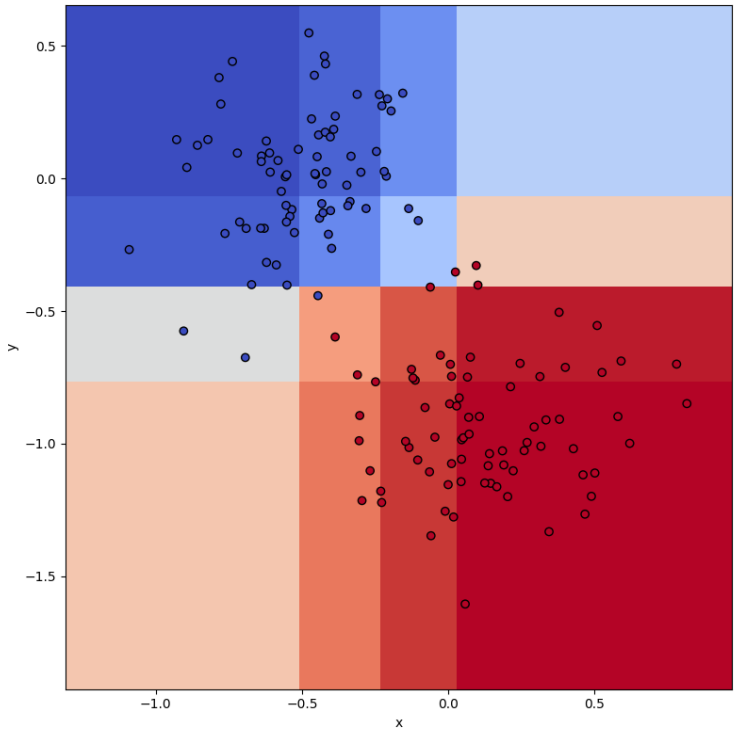


Figure 4.16: Model prediction results for dataset A2. Again darker colors are used to show regions with lower uncertainty.

Rule	Mass Blue	Mass Red	Uncertainty
$x < -0.518$	0.695	0.000	0.305
$-0.518 < x < -0.248$	0.288	0.104	0.607
$-0.248 < x < 0.023$	0.036	0.518	0.445
$x > 0.023$	0.000	**0.816**	0.184
$y < -0.782$	0.000	**0.802**	0.198
$-0.782 < y < -0.424$	0.000	0.693	0.307
$-0.424 < y < -0.067$	**0.753**	0.001	0.246
$y > -0.067$	**0.886**	0.000	0.114

Table 4.2: Resultant rules after model training for dataset A2

three-class dataset, called A3, using sampling from three Gaussian distributions with their centers at distant points. Figure 4.17 shows the dataset distribution and the class of each point.

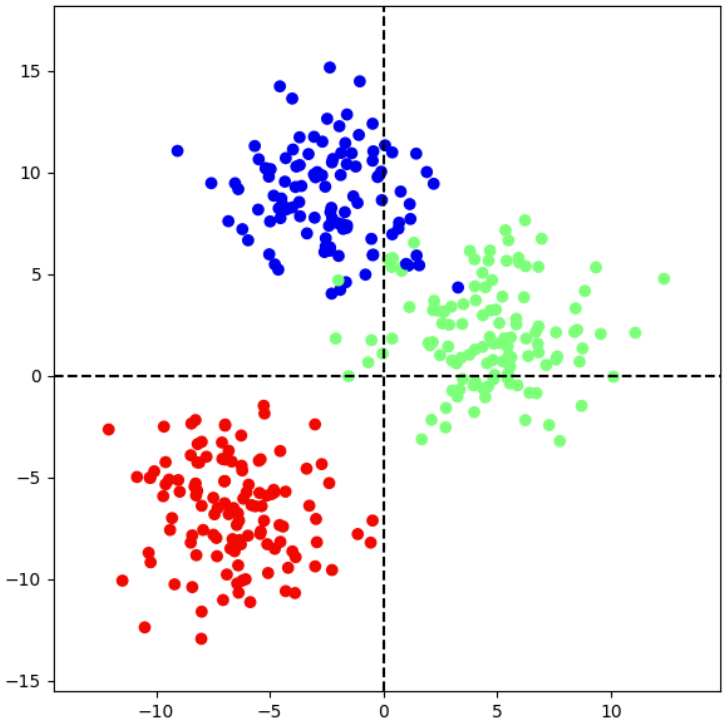


Figure 4.17: Distribution of the dataset A3 containing three point classes.

The model was tested using the same configuration utilized in the previous examples, but using five breaks instead of three. Note that all of these parameters are independent of the number of classes, so we do not need to make any change to the definition of the model nor its implementation.

Figure 4.18 shows the training loss over the epochs. The model converges after 91 iterations and reaches a loss value of 0.034.

When requesting the model to predict the validation set, it achieves an accuracy of 98.1% (on average of five tests). Figure 4.19 shows the distribution of the mass values for each class on the domain of the dataset and the validation records. Darker colors mean higher values for the mass.

Analysing the results for this experiment, we can observe that our model performs multi-class prediction without any problem. A particular challenge presented in this dataset was the fact of having more classes than attributes. Note that we need to have at least one rule for each class; otherwise, the model will not be able to distinguish all of them. Besides that, having only one rule per class is not recommended since the only way to perform a correct classification in this configuration is that the condition of each rule defines completely the definition for each class, which is usually not the case. For that reason, when having more classes than attributes, it is better to have more rules, which explains why we used five breaks instead of three in this dataset. The drawback of having more breaks is that more training data is needed to fulfill the mass values of these rules. If few data records satisfy the condition of a rule, then the support for this rule is low, and the model could overfit as

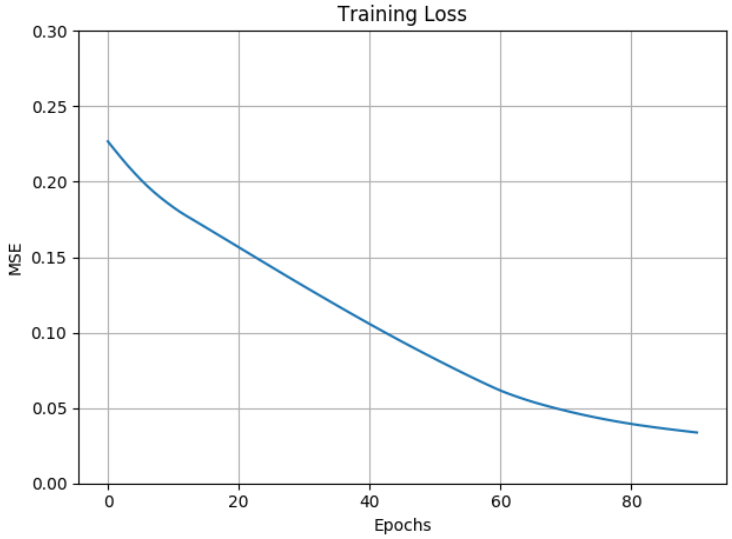


Figure 4.18: Training Loss of the model through epochs for the dataset A3.

explained in section 3.4.

4.1.7 Commonality transformation

Our model uses the Dempster Rule, which is a very costly operation, hence alternatives to reduce its cost were explored. In particular, the commonality transformation was presented and implemented in the model, which is another representation for mass assignment function that makes Dempster Rule a simple multiplication of commonalities. In sections 3.4.3, 3.4.4 and 3.4.5, we explain the advantages of using the commonality to reduce several of the operational costs of the method. However, it is not completely evident that this reduction in computational cost compensates the overhead of the computation of the transformation.

In this subsection, we will test these two implementations in order to know whether the transformation reduces the computational effort empirically. The A2 dataset from the previous section will be used since it is simple, and we know beforehand how the model behaves working with it. The same rules, initial learning rate, optimizer, and loss function will be used for the two variations of the model to avoid the differences that these variables could introduce. We will also force both methods to perform 100 epochs within the training phase.

For each method, we will measure the training time using the same hardware and conditions. This time will be split into the four main parts of the model, which are: the computation of the predicted class, the calculation of the gradients, the update of the mass values, and the normalization of them.

Figure 4.20 shows the results for both methods for five repetitions. The chart also shows the standard deviation of each time measurement.

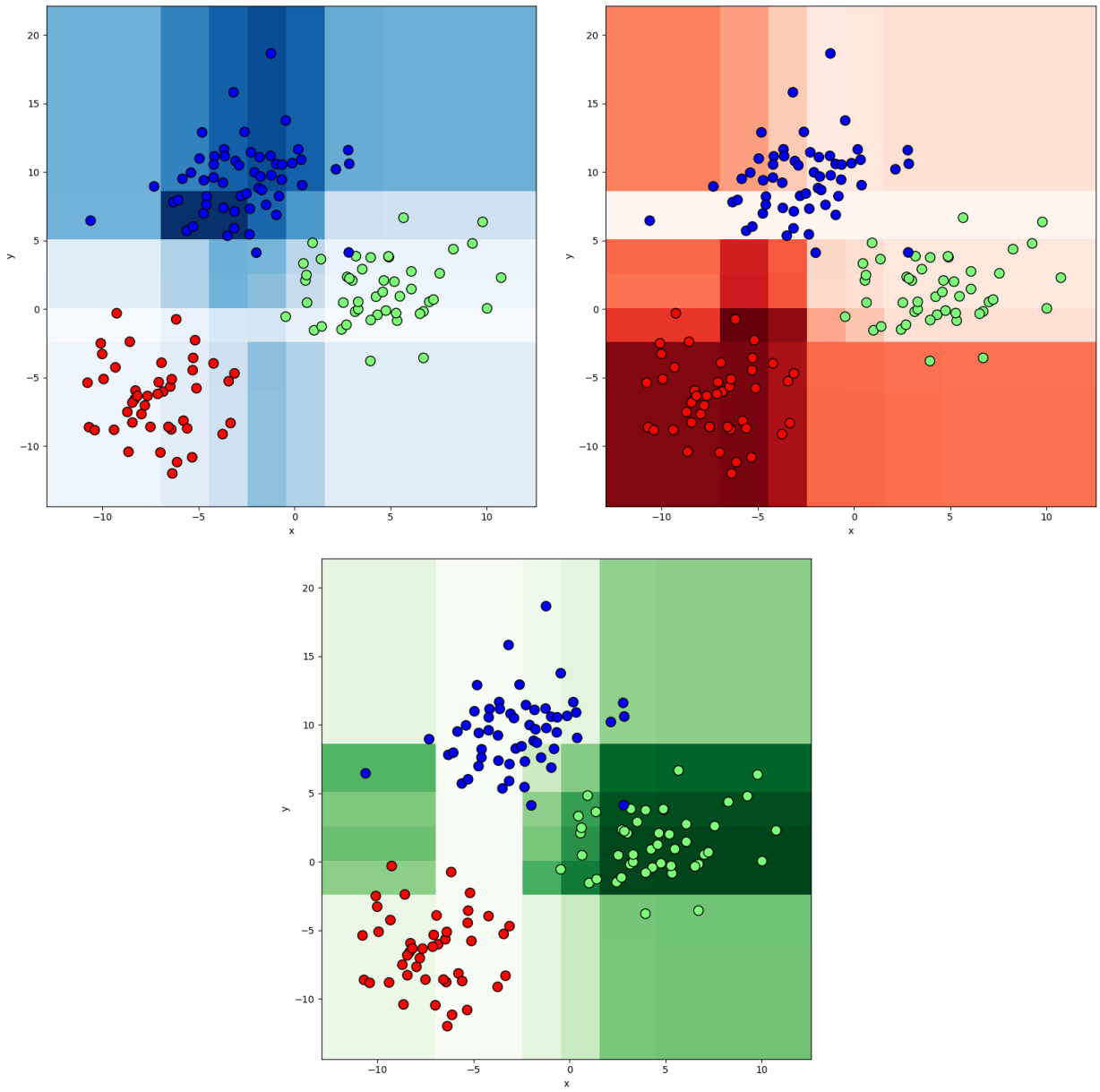


Figure 4.19: Model prediction results for dataset A3.

As we can notice, the commonality transformation effectively reduces the computational time of the training process in about 23% for this dataset. We can also see that the most costly subprocesses of the training phase are the class prediction and the computation of the gradients, where a significant cost reduction is observed. The normalization and update of the masses takes very little time compared to the other subprocesses, and using the transformation does not affect these times.

Finally, from this experiment, we can also observe that the commonality transformation may be useful for decreasing the complexity of computation and hence speeds up the model. All subsequent tests will use the commonality transformation.

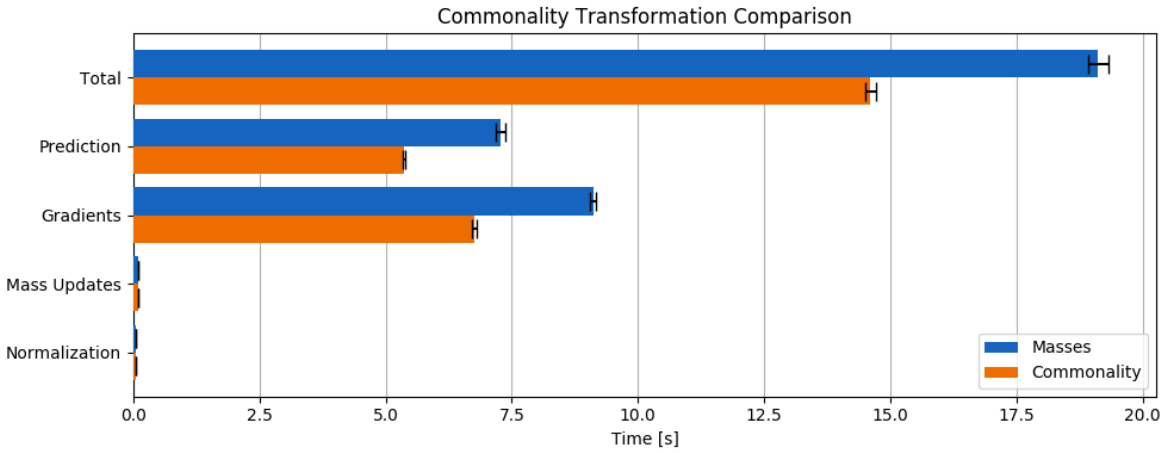


Figure 4.20: Training times for the methods based on masses and commonalities.

4.1.8 Multiple kinds of attributes

The next experiment will be similar to the previous one, but we will use more attributes, and they will have certain information. For example, we can build a dataset with four features, where only two of them are informative (i.e., these two correlate with the instance class), and the other two attributes are random noise. In these datasets, the model should learn that the informative attributes are the most important to make the classification. On the same line of this idea, another experiment is to have redundant attributes, i.e., attributes that are combination of other attributes. In this case, the model should learn which are the informative ones again.

We present three experiments varying these kinds of attributes. Dataset B1 has two informative attributes and two random attributes; dataset B2 has two informative and two redundant attributes, and dataset B3 has two informative, two random and two redundant attributes.

Informative attributes are similar to those presented in the first cases, where their sign correlate directly to the class. Random attributes are generated by sampling random uniform distributions. And redundant attributes are generated by scaling and adding a constant to an informative attribute.

Table 4.3 presents the results for the model using the same configuration as in the previous case for the scenarios B1, B2, and B3. From the table, it is possible to note that in all cases, the model achieves perfect accuracy.

Exp.	Informative	Random	Redundant	Accuracy
B1	2	2	0	1.000
B2	2	0	2	1.000
B3	2	2	2	1.000

Table 4.3: Results of controlled scenarios

The results of Table 4.3 show that the model can perform correct classifications when informative attributes are present in datasets. Also, we showed that this feature is not affected by the presence of noisy or redundant attributes.

4.1.9 Scikit learn classifier comparison datasets

Scikit Learn [63] is a python library that implements many of the most known machine learning methods, such as Multilayer perceptrons, Support Vector Machines, and Random Forest. The documentation includes user guides and tutorials to teach users how to use the library. One particular document called "Classifier comparison", which is available on this URL `https://scikit-learn.org/stable/auto_examples/classification/plot_classifier_comparison.html`, aims to compare all the classification methods the library provides using three 2D synthetic datasets. Figure 4.21 shows the results of the article. This figure uses the same notation that we used in the previous examples, and the number which appears in the bottom right corner is the accuracy of the method for the validation set. In this subsection we would like to compare our model with the ones implemented by the Scikit Learn library.

For easing further analysis we name the datasets in the following way: the one presented in the first row will be called "moons". The one in the second row will be called "circles", and the one in the last row "linear" dataset.

The model will be tested using MSE as loss function, ADAM as optimizer with an initial learning rate of 0.02, an ε of 0.0001 for convergence, and automatic generated rules using the single attribute generator with six breaks, and the two-attribute rule generator.

Figure 4.22 shows the result of the model over these three datasets.

For the case of the "moons" dataset, our model achieves an accuracy of 88.7%. This performance positions the model as the 7th best classifier, obtaining better results than Linear SVM, Naive Bayes, and Quadratic Discriminant Analysis.

For the case of the "circles" dataset, our model reaches an accuracy of 86.7%. In this case, our model is the 5th best model for this dataset, outperforming Linear SVM, Decision Trees, Random Forest, Ada Boost, Naive Bayes, and QDA.

Finally, for the case of the "linear" dataset, the model achieves an accuracy of 95%, which is the same accuracy that the best models for this dataset obtain.

Another interesting aspect to analyze is the shape of the boundaries generated by the models. As shown, the proposed model generates boundaries that are delimited by straight lines which are parallel to the axes. This behavior is typical of methods that decide according to a particular property of one attribute, for example, being in a specific range. It is important to emphasize that all rule-based methods tend to form these kinds of boundaries. In particular, if we look at our model, the results of Decision Trees and Random Forest, we will see this pattern.

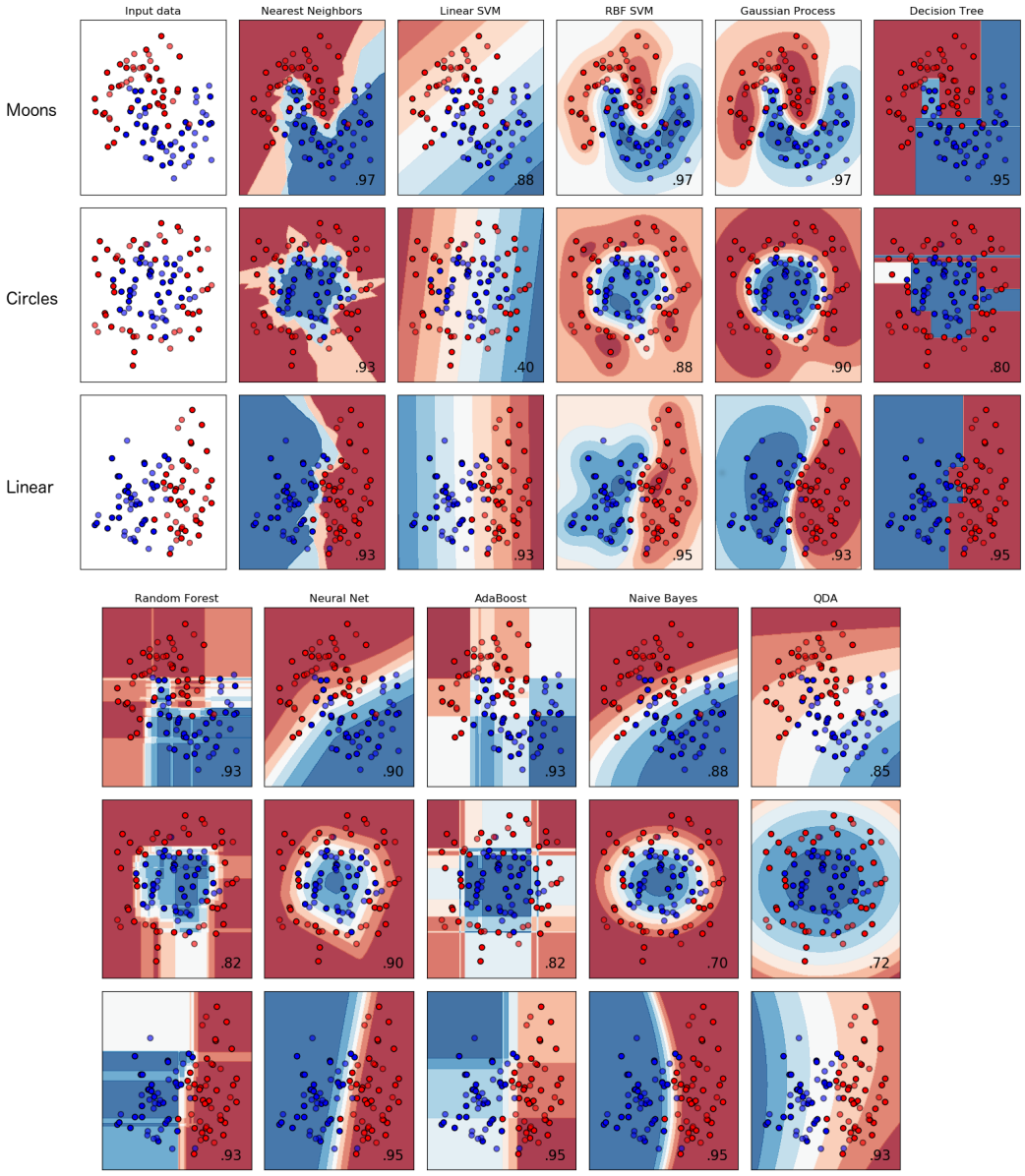


Figure 4.21: Classifier comparison results from scikit-learn website

In summary, the model behaves like an average classifier. Methods like KNN and SVM with RBF kernel always perform equal or better than our model, but also, there were models performing worse in every case. Moreover, as the authors also explain in the article, the conclusion over these experiments may not apply to real datasets. The main reason is because realistic datasets have more dimensions and less tricky distributions than the ones presented in these experiments, especially the "moons" dataset where our model has the highest error. On realistic datasets, more uncomplicated relationships may lead to a better generalization, and thus, better prediction for unseen data.

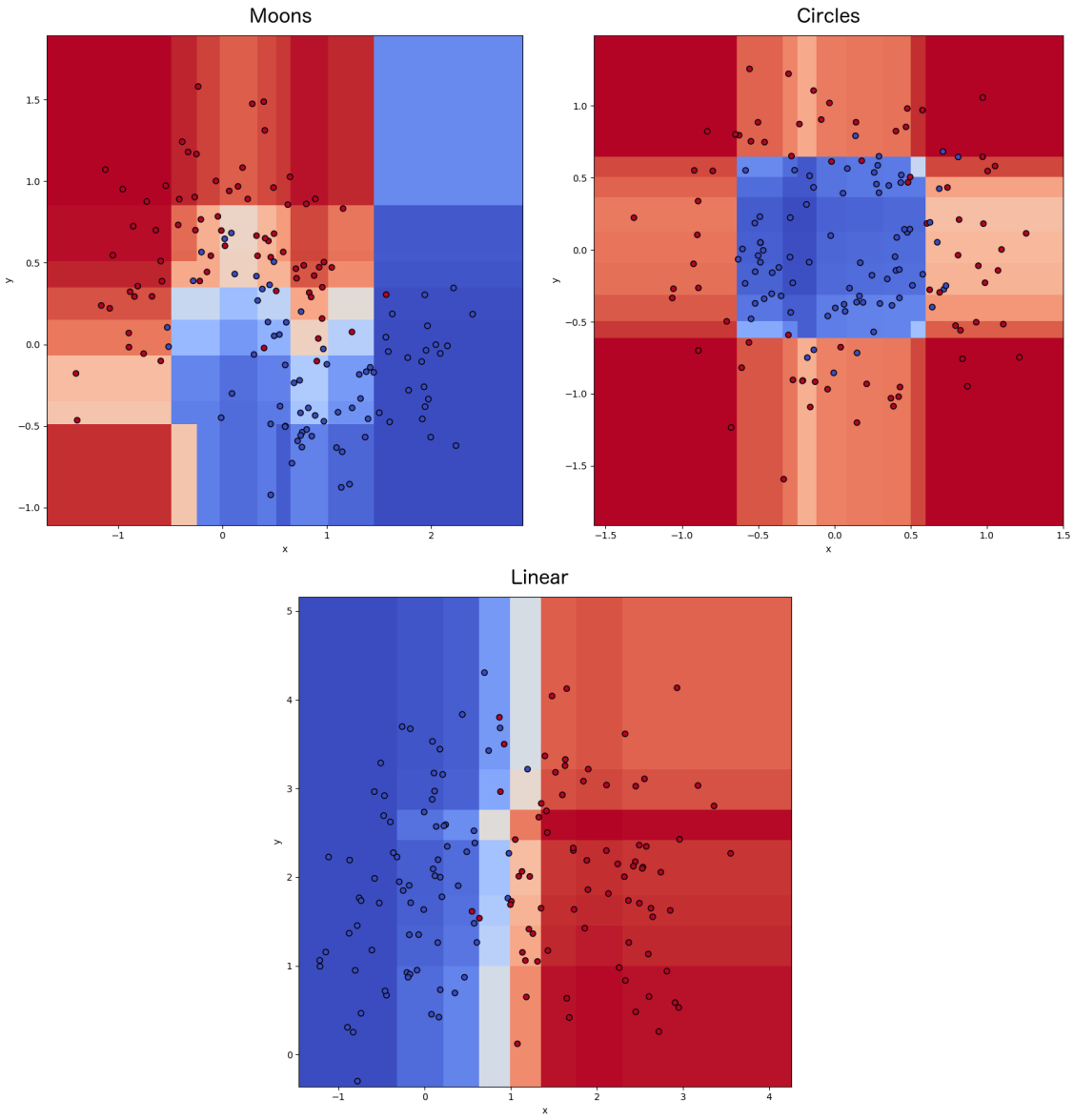


Figure 4.22: Model results over the dataset from the sklearn experiment.

4.2 Traditional Datasets

Until now, the model has been tested using fictional controlled scenarios, obtaining excellent results in terms of prediction and interpretability for all the experiments performed.

In order to continue the testing of the model in more realistic scenarios, in this section, the model will be tested in several traditional datasets. These datasets are the ones that appear in machine learning books and courses as example datasets to start using classification models, and they define typical experiments to compare any model with other classification methods. These datasets are publicly available, and their authors grant the right to use them for research purposes.

Specifically, the datasets we will use to test our model in this section will be the following.

1. Fischer Iris dataset
2. Wine quality
3. Heart disease
4. Wisconsin breast cancer
5. Handwritten digits
6. Gas sensor array drift

All of them were obtained from the UCI repository [64]. All the datasets presented in this section, except handwritten digits, are tabular, i.e., their data records have structured by attributes, and they are usually presented as tables. For the case of handwritten digits, our model will consider the images as attributes of a table as well, and we will discuss the implications of this decision later.

In the next subsections, each dataset will be described briefly, and then we will present the model configuration and results.

The result obtained with our model, using Dempster Shafer with Gradient Descend (DSGD) will be compared with the results of the following classification algorithms:

1. Majority classifier (Majority).
2. Random Forest with 100 trees (RF).
3. Naive Bayes with Gaussian estimation of probability distributions (NB).
4. K-Nearest Neighbors with $k = 5$ (KNN).
5. Multi Layer Perceptron with a single hidden layer of 100 neurons (MLP).
6. SVM with RBF kernel (SVM).

All of these methods are widely known and used in classification tasks using tabular data.

For the performance metrics, we will use k-fold cross validation [65], where the value of k will be set according to the dataset size. For small dataset k will be 3, and for large datasets, k will be 5. The classification instruments and metrics to be measured will be the following:

1. The **accuracy**, which is defined as the quotient between the correctly predicted instances and the number of cases. It is the most widely used indicator for classification tasks.
2. The F_1 **macro score**, which is defined as the average of the F1 score of each class. The F1 score of a class is computed using the following formula:

$$F1 = 2\,\frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \tag{4.1}$$

3. For binary datasets (i.e., the ones which have two classes), the **Receiver Operating Characteristic** (ROC) curve will be presented. The ROC curve corresponds to a chart that illustrates the rate of true positives to false positives when moving the acceptance threshold. The area under this curve will also be computed.

4.2.1 Iris Dataset

Fischer Iris dataset [66] is a multivariate data set introduced by the British statistician and biologist Ronald Fisher. The data set consists of 50 samples from each of the three species of Iris flower (Iris setosa, Iris virginica, and Iris versicolor). Four features were measured from each sample: the length and the width of the sepals and petals, in centimeters.

Iris dataset has become one of the most typical datasets for machine learning and statistical clustering and classification tasks.

Model	Accuracy	F1 Macro	ROC AUC
DSGD	0.960	0.957	0.990
RF	0.947	0.947	0.993
NB	0.940	0.940	0.994
KNN	0.967	0.967	0.991
MLP	0.960	0.960	0.997
SVM	0.973	0.973	0.998
Majority	0.320	0.162	0.500

Table 4.4: Results for Iris dataset.

4.2.2 Wine quality dataset

Wine quality dataset [67] is another binary dataset that contains information about red and white variants of the Portuguese "Vinho Verde" wine. This dataset contains 4898 records, where the input variables are based on physicochemical tests. They are fixed acidity, volatile acidity, citric acid, residual sugar, chlorides, free sulfur dioxide, total sulfur dioxide, density, pH, sulfates, alcohol. Besides these variables, the dataset also includes information about the color of the wine (red or white) and the quality (value from 0 to 10).

This dataset can be viewed as a regression or classification tasks. For regression, a model could predict the value of the quality based on the physicochemical variables. As a classification task, a model could predict the color of the wine, or the discrete quality, e.g., good if quality $\geq$ five or poor otherwise.

This dataset differs from the two previous ones with regard to the number of records: it contains many more.

4.2.3 Heart Disease Dataset

The Heart Disease Dataset [68] presents the results of clinical reports of heart disease. A total of 76 attributes are presented in the original work. However, many of these attributes are incomplete, or they do not provide information that contributes to prediction such as the

Model	Accuracy	F1 Macro	ROC AUC
DSGD	0.988	0.984	0.996
RF	0.995	0.993	0.998
NB	0.970	0.961	0.987
KNN	0.941	0.918	0.958
MLP	0.982	0.975	0.994
SVM	0.947	0.925	0.973
Majority	0.754	0.430	0.500

Table 4.5: Results for Wine Quality dataset.

patient id, and the national identification number. For this reason, many of these attributes were dropped, and the standard dataset now contains 14 attributes.

Model	Accuracy	F1 Macro	ROC AUC
DSGD	0.662	0.597	0.707
RF	0.662	0.613	0.698
NB	0.701	0.670	0.744
KNN	0.610	0.513	0.613
MLP	0.647	0.546	0.716
SVM	0.654	0.395	0.527
Majority	0.654	0.395	0.500

Table 4.6: Results for Heart Disease dataset.

4.2.4 Breast Cancer Dataset

Breast Cancer dataset [69] is another typical dataset. This dataset contains 699 the clinical reports of breast cells, detecting whether they are benign or malignant. Each record contains nine useful attributes about the sampled cells: Clump Thickness, Uniformity of Cell Size, Uniformity of Cell Shape, Marginal Adhesion, Single Epithelial Cell Size, Bare Nuclei, Bland Chromatin, Normal Nucleoli, and Mitosis. Unlike the previous case, this is a binary classification problem.

Model	Accuracy	F1 Macro	ROC AUC
DSGD	0.964	0.961	0.988
RF	0.963	0.959	0.987
NB	0.755	0.706	0.844
KNN	0.607	0.535	0.567
MLP	0.555	0.366	0.653
SVM	0.661	0.414	0.580
Majority	0.655	0.396	0.500

Table 4.7: Results for Breast Cancer dataset.

Breast Cancer dataset also offers an interesting case to look in detail to exemplify the advantages of our model. This dataset is one of that our model performs the best among the traditional datasets tested, achieving an accuracy of 96.4% . Moreover, after the training, the top 7 most important rules for detecting malignant cells (i.e., the ones having higher mass in malignant singleton) are obtained as a "sub-product". They are presented in table 4.8.

Rule	Mass Benign	Mass Malignant	Uncertainty
clump thickness > 6.34	0.000	0.702	0.298
bare nucleoli > 5.97	0.038	0.673	0.289
$3.2 <$ epithelial size < 4.71	0.000	0.687	0.313
size uniformity > 5.16	0.000	0.658	0.342
marginal adhesion > 4.74	0.000	0.608	0.392
$1.65 <$ mitoses < 2.84	0.002	0.597	0.401
epithelial size > 4.71	0.017	0.580	0.403

Table 4.8: Most important rules for Malignant class for Breast Cancer problem

In order to show the importance of having interpretability results, we can compare the rules obtained with the medical knowledge. From Table 4.8, one of the essential rules is the value of epithelial size, which appears in two of the most important rules. For this variable, higher values are associated with malignant cells. Checking this result with literature, Doyle et al. [70] present a study about the differentiation of normal and malignant breast cells. They show that the malignant cells exhibit larger cell and epithelial nucleus sizes as compared to the healthy cells. This statement matches exactly the rules the model obtain from the interpretability, showing that it can verify this kind of knowledge.

As our proposed model is also rule-based, we can adapt the FRBS indicators for interpretability presented in section 2.3 to our case. Also, the authors that proposed these interpretability indicators tested them using this dataset, then we can compare the result our model obtained to the ones that they reported. The indicators Q_{RANT} and Q_{ANT} do not need adaptation since we can count the number of antecedents directly from the statements. The indicator Q_{FS} needs to be re-interpreted to comply with our model. In FRBS n_{FS} refers to the numbers of active fuzzy sets using linguistic terms. We can note that fuzzy sets are similar to our definition of rule. Thus the most straightforward way to define n_{FS} in our case is to count the average number of rules an instance uses when the model performs a classification.

In order to test different configurations of our model, we can only use the top n rules that contribute most to each class (as explained in section 3.5) for the classification, and all the other rules can be dropped. Applying this procedure, we can obtain a simpler model but still accurate. Table 4.9 shows the results for accuracy and interpretability measures for different configurations of the model.

Figure 4.23 shows the values of Q_{CPLX} and Error from the Table 4.9. Blue dots show configurations of the model. The red dot represents the most interpretable and accurate configuration, which is achieved taking the top 6 rules of each class. The dashed line represents all configurations that are equally accurate and interpretable to the red dot (assuming

Top n	N. Rules	Q_{CPLX}	Accuracy	Error
all	45	0.3672	96.4%	0.0382
18	36	0.2195	96.4%	0.0382
9	18	0.1062	95.2%	0.0476
6	12	0.0522	94.8%	0.0524
3	6	0.0363	92.4%	0.0762
2	4	0.0155	89.1%	0.1095

Table 4.9: Interpretability and accuracy measures for Breast Cancer problem

that they have the same importance). This line is presented to compare with the other dots clearly.

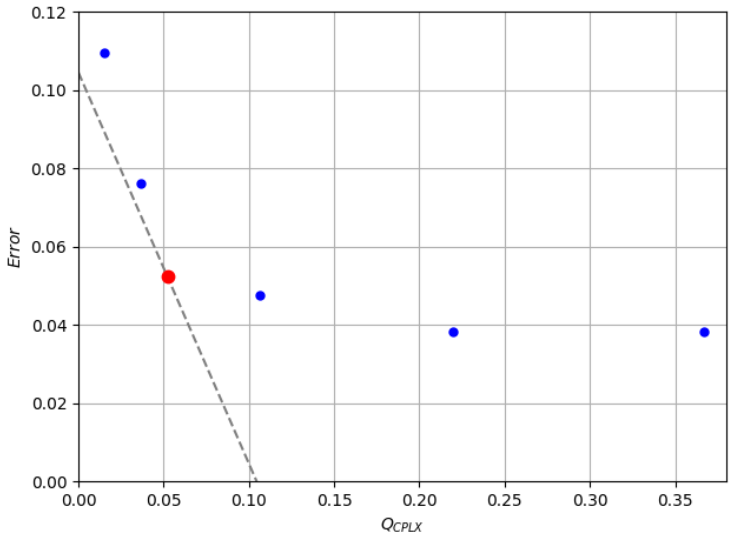


Figure 4.23: Complexity and Classification Error measures for Breast Cancer problem

The best configuration for the model was obtained when using the top 6 rules for each class. From the chart, we can observe that the model can drop many rules and still getting good accuracy, note that the best configuration has only the 27% of the rules of the original model. Comparing our results to the results presented by Gorzalczany and Rudzinski [21], we can see that our results for the best configuration are comparable. Our method has a slightly higher error of 0.0524 vs. 0.0365, but we have a slightly lower complexity of 0.0522 vs. 0.0568, thus better interpretability.

4.2.5 Handwritten Digits Dataset

Handwritten Digit dataset [71] contains information about normalized bitmaps of handwritten digits from a preprinted form. The bitmaps are presented in their original resolution after processing of 32x32 pixels or a rescale of these bitmaps in a smaller resolution of 8x8, which reduces dimensionality without affecting performance.

This dataset was widely used as the example dataset for convolutional neural networks or methods concerning image processing and classification.

Although this dataset is clearly an image dataset and thus is multimedia, we will tabulate the value of each pixel in a column. Then we will have a 64-attribute dataset (using the 8x8 resolution images).

The dataset contains 5620 different hand drawings and ten classes from digit 0 to digit 9.

Model	Accuracy	F1 Macro	ROC AUC
DSGD	0.896	0.896	0.988
RF	0.974	0.974	0.999
NB	0.846	0.847	0.978
KNN	0.982	0.982	0.998
MLP	0.967	0.967	0.999
SVM	0.572	0.627	0.993
Majority	0.102	0.018	0.500

Table 4.10: Results for Digits dataset.

Digits dataset presents a compelling case for the analysis of interpretability. This dataset contains handwritten digits in an 8x8 pixel box. The larger value means a mark is in the pixel; this information is flattened into a 64-length vector and passed as input. Although for the model this is like any other dataset, we can extract the importance of each rule, and since rules are related to single attributes, we can know exactly which of these 64 pixels are contributory for the model to predict a digit. This result is presented in Figure 4.24.

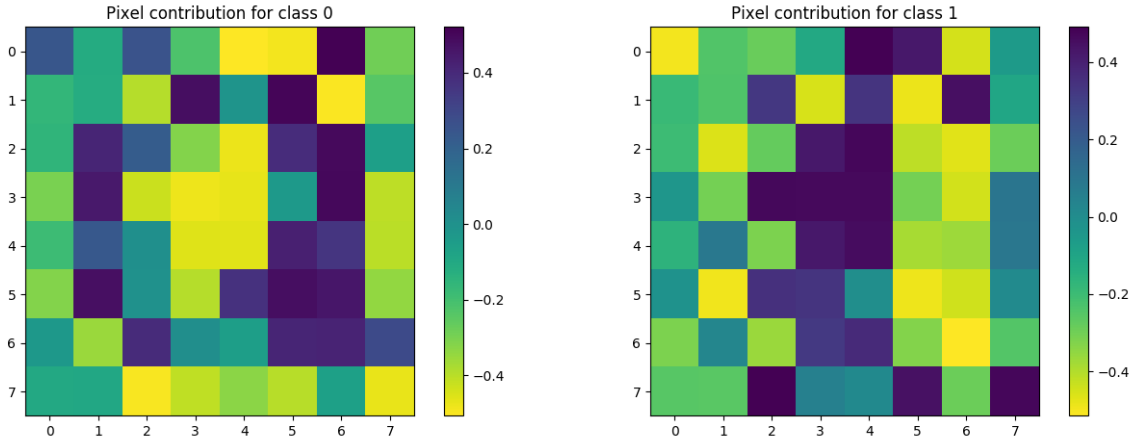


Figure 4.24: Pixel importance for the prediction of the classes 0 (left) and 1 (right) in the Digits dataset according to the interpretation of the rules.

The results of Figure 4.24 shows that for the case of 0 digit darkest blue pixel, i.e., the ones that contribute most to this prediction, are distributed in the extreme of the image, and they tend to form a circle; for the case of 1 digit, the darkest blue pixels tends to form a straight vertical line in the middle of the image. In both cases, these results make sense with typical digit drawings.

4.2.6 Gas sensor array drift

The gas sensor array drifts dataset [72] is another dataset from the UCI repository provided for research purposes. This dataset corresponds to a large dataset to test models. It has 128 features and 13910 records without missing values. The data comes from 16 chemical sensors exposed to 6 gases at different concentration levels. The goal is to predict the gas analyte according to the values of the sensors.

Model	Accuracy	F1 Macro	ROC AUC
DSGD	0.885	0.871	0.983
RF	0.993	0.993	0.999
NB	0.554	0.555	0.839
KNN	0.978	0.978	0.996
MLP	0.508	0.417	0.779
SVM	0.216	0.059	0.499
Majority	0.216	0.059	0.500

Table 4.11: Results for Gas sensor array drift dataset.

4.2.7 Summary

A summary of the results obtained in the previous datasets is presented in Table 4.12. This table shows the number of attributes (Attr), the number of records (Size) and the number of classes (Cls), how many rules the model uses in each case, and the accuracy on the test set.

Dataset	Attr	Size	Cls	Rules	Accuracy	F1 Macro	ROC AUC
Iris	4	150	3	28	0.960	0.957	0.990
Breast Cancer	9	700	2	108	0.964	0.961	0.988
Wine quality	13	6497	2	50	0.988	0.984	0.996
Heart Disease	9	462	2	34	0.662	0.597	0.707
Digits	64	1796	10	168	0.896	0.896	0.988
Gas sensor array drift	128	13910	6	512	0.885	0.871	0.983

Table 4.12: Results in traditional datasets

The application of the model to traditional datasets helps validate the previous results about accuracy and prediction power. Results from Figure 4.12 show that in most cases, accuracy is over 85%, which means that the classifier can perform operations with less than 15% of error. Another remarkable result is that the model performs well in multi-class classification tasks, from our experiments Iris and Digits are both multi-class datasets with 3 and 10 classes respectively, the metrics show that the model can also handle these kinds of data.

Chapter 5

Application to Stroke risk prediction

This chapter describes the study case of the proposed implementation, which is the stroke risk prediction problem. First, the data is presented, analyzed, and explored, then the problem is defined, and finally, the result of our model applied to this problem is presented.

5.1 Motivation

Cardiovascular disease (CD) is the leading global cause of death, accounting for more than 17.3 million deaths per year in 2013, a number that is expected to grow to more than 23.6 million by 2030 [73]. In 2015, globally, an estimated 7.4 million deaths were due to coronary heart disease, and another 6.7 million deaths were due to stroke. In 2010 the estimated global cost of CD was calculated to be $863 billion, and it is expected to rise to $1044 billion by 2030. Only in Japan, in 2012, around 55000 under 70 years old died because of CD [74].

The World Health Organization's Global Strategy for the Prevention and Control of non-communicable Diseases (ND) recommends the evaluation of various strategies for early detection and screening of ND to promote research on prevention and control of these diseases [74]. Being aware of the risk of developing these illnesses in early stages has proved to be beneficial not only for prevention but also for the treatment of patients [75].

It is known that many countries gather electronic records of patients' medical checkups from over the years. These records usually include symptoms, examination results, and the history of diseases of each patient. In the case of Japan, all active workers have to undergo every year a medical checkup to monitor their health, data that is also electronically recorded. This results in large databases of health information that can potentially be used in many ways to train classification or prediction models and a compelling case to test our model.

Technically, this case is attractive for our model since it combines several factors that contrast our model from the other solutions. For example, interpretability is required to prevent patient discrimination. Data is structured (tabular), but it may has many missing values. And, expert knowledge of factors that may correlate to stroke occurrence can be added to the model.

5.2 Data Description

This data for the study case was retrieved from electronic health records (EHR) and receipts from the Tsuyoyama Hospital, Japan. The information was provided by the Japanese company Allm in cooperation with Universidad de Chile for research purposes and it is not publicly available due to the potentially sensitive data it contains.

In Japan, all active workers have to undergo every year a medical checkup to monitor their health, which is mainly the source of the EHR and receipt data. All personal information such as names, telephones, and addresses were omitted and not considered as part of the dataset. Data were first processed for parsing, filtering, and transforming the data to store it as a relational database.

The data model is very complex, there are more than 20 entities, and some tables have more than 100 attributes. Also, the data model lacks normalization, so many of the attributes are entirely missing and others are redundant. For the purpose of this study, only certain entities and characteristics were used. A simplified version of the entity-relation diagram of the database that considers only the information we used for this work is presented in Figure 5.1.

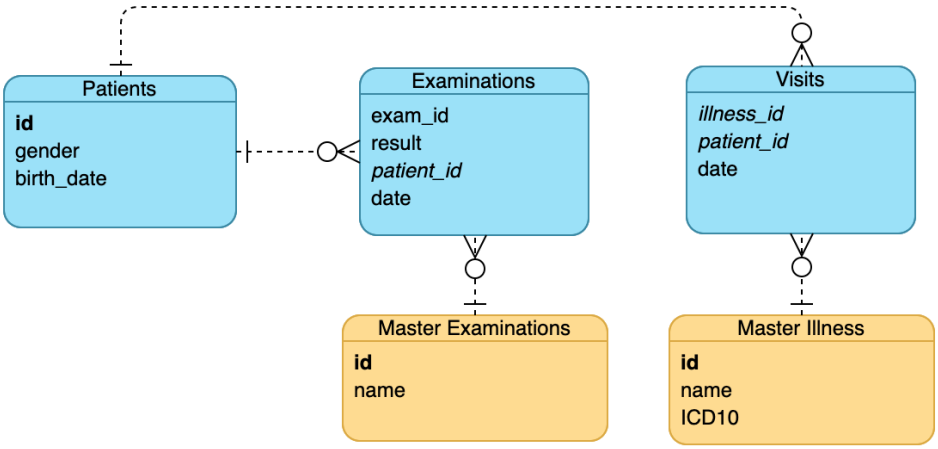


Figure 5.1: Simplified Entity-Relation Diagram of the database

A brief description of each entity is presented below.

1. **Patients** is the table that contains basic information about patients, such as gender and birth date. Personal information was omitted to keep patient's privacy.
2. **Examinations** is the table where the details of exams taken are stored. It includes the date when the exam was collected, the exam code, the patient id, and the result of the exam.
3. **Master Examinations** is a join table that provides more information about exams like the exam name.
4. **Visits** is the table used to store the visits or encounters of a patient. It includes

information like the date of the meeting, the diagnosed illness id and the patient id.

5. **Master Illness** is a join table that provides more information about illness like their name, and their ICD-10 code [76].

Besides the data model presented above, we can analyze the information semantically. In this context, we can observe three data sources described below.

1. **Patient Checkups**. This data source contains all the information about results of exams that Japanese workers have to undergo annually. This data is directly obtained from the Examination entity. Examples of exams are LChO exam and blood pressure. There are 23 different types of exams.
2. **Patient demographics.** It contains general information about patients such as height, weight, body fat, age, gender, and waist measurement. This data is obtained from Patient and Examination entities. For data that is permanent such as gender and birth, it is obtained from the Patient entity. The data that is variable in time like height, weight, body fat, and waist measurement is represented in the data model presented above as Examination with special exam codes.
3. **Patient disease history.** This data source contains information about the diagnoses of patients. It is directly retrieved from the Visit entity. The diseases are indexed by ICD-10 codes [76], which allows having a structured and hierarchical way to represent diseases. This is the data source we used to check whether a patient has a stroke.

Once the data is defined and available, it is necessary to study the quality of the data in terms of missing values, duplication, and check if tables are joinable. The data presents many missing values, mainly in Examinations and Visits entities. Also, in many cases joining entries for a patient is not possible (e.g., there patient ids in Examinations entity that does not exists in Patients entity). There is no data duplication.

If we exclude entirely the records with missing data the around 95% of the dataset will be lost. In order to conduct a valid study losing that amount of data is not permissible. Therefore a method to handle missing values is required. However, data that cannot be joined is not possible to use because we will lose almost all information about the patient and because one of the goals is to relate exam results with stroke risk.

After applying all filters, a total of 27876 patient records remained.

5.3 Data Exploration

Before performing the prediction using the data, it is important to understand and visualize the data in order to find relations that could help further prediction. This process is called data exploration.

Patient demographics are a good starting point for data exploration. Our first chart presented in Figure 5.2 shows the distribution of the patient's age separated by gender. This chart shows that all patients are adults, which coincides with the fact that our data comes

from the annual checkups of Japanese workforce. In fact, they tend to be elderly with a mean value of 69 years. We also observe that there are slightly more men than women in the dataset, with a proportion of 54% to 46%.

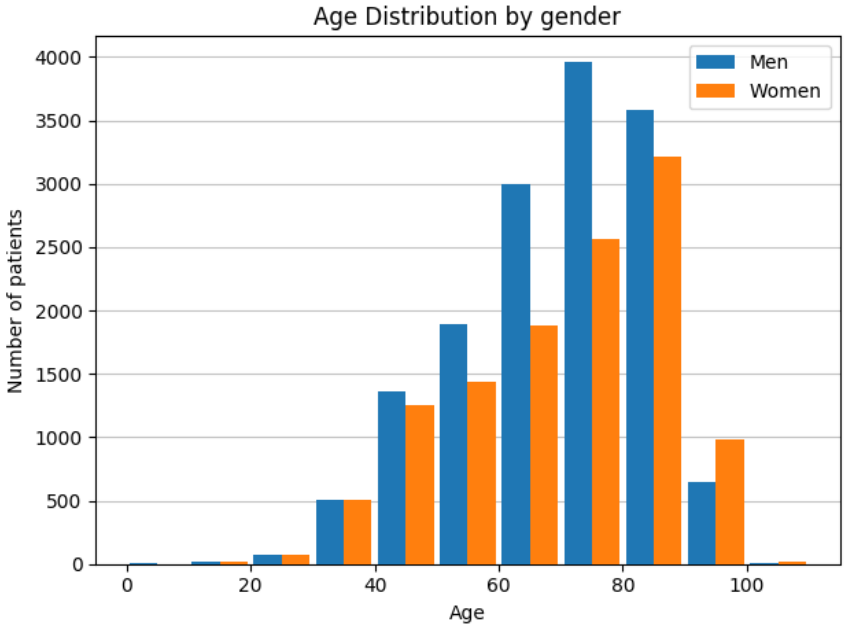


Figure 5.2: Age distribution by gender of patients in the dataset

The next demographic variables explored are the height and body weight of patients. Figure 5.3 shows the distribution of patient's height (left) and body weight (right) separated by gender. In this case, we observe that both variables behave like a normal distribution. However, the mean of the distribution for men and women is different. In both variables, women have a lower mean than men.

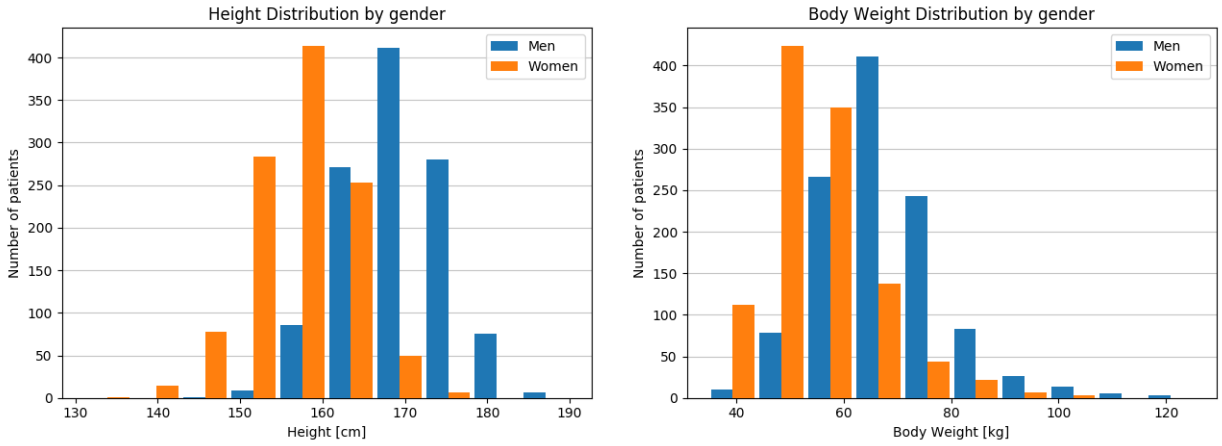


Figure 5.3: Height (left) and body weight (right) by gender of patients in the dataset

Another important clinical indicator is the Body Mass Index (BMI), which is computed as the body weight in kilograms divided by the square of height in meters of a patient

($BMI = w/h^2$). BMI can be used to screen for weight categories that may lead to health problems. Commonly accepted BMI ranges are underweight: under 18.5 kg/m2, normal weight: 18.5 to 25, overweight: 25 to 30, obese: over 30 [77].

Figure 5.4 shows the number of patients that belong to each BMI category separated by gender. The chart shows that most of the patients are in the normal range of BMI. Unlike height and weight presented above, BMI is gender agnostic since we can note that the distribution for men and women among ranges is very similar.

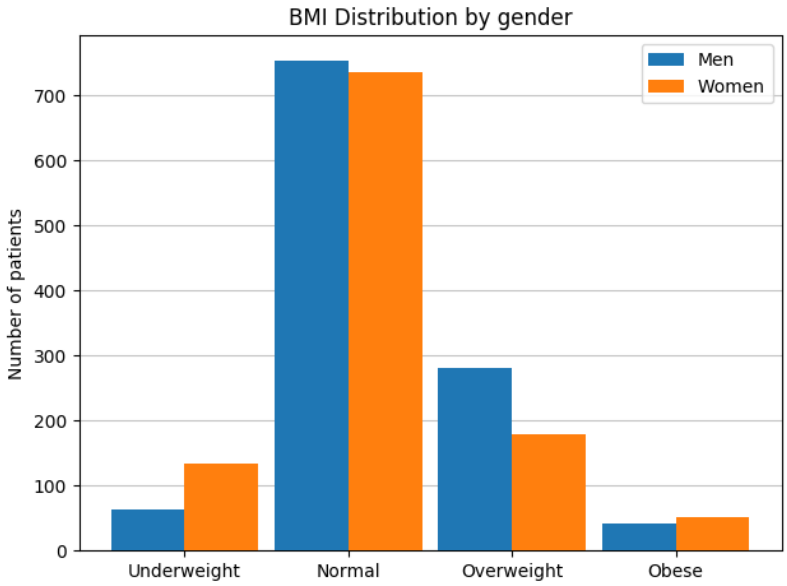


Figure 5.4: Body Mass Index distribution by gender of patients in the dataset

The following data source to be explored is exam result data. As mentioned above, the data has many missing values, the chart of Figure 5.5 shows the 23 different types of exams and how many of the patients of the dataset has at least one result for each exam.

From Figure 5.5, we can observe that most people do have results for most of the exams recorded. Also, we note a number of exams with similar number of people having results for them. That is explained by the fact that many exam results are obtained from the same procedure. For example, exams from MCHC to CREA are obtained from a blood test. Similarly, HDL-C and LDL-C are derived from a Lipid profile.

Using the patient disease history data, we can extract all diseases a patient has been diagnosed using the corresponding ICD10 code for the disease. In particular, we can search for patients with code I63, which stands for Cerebral infarction (stroke). The Figure 5.6 shows the number of patients that have had a stroke according to information in the dataset separated by gender.

In this figure, 22140 of the patients do not have a stroke (79.4%), and 5736 have stroke (20.6%).

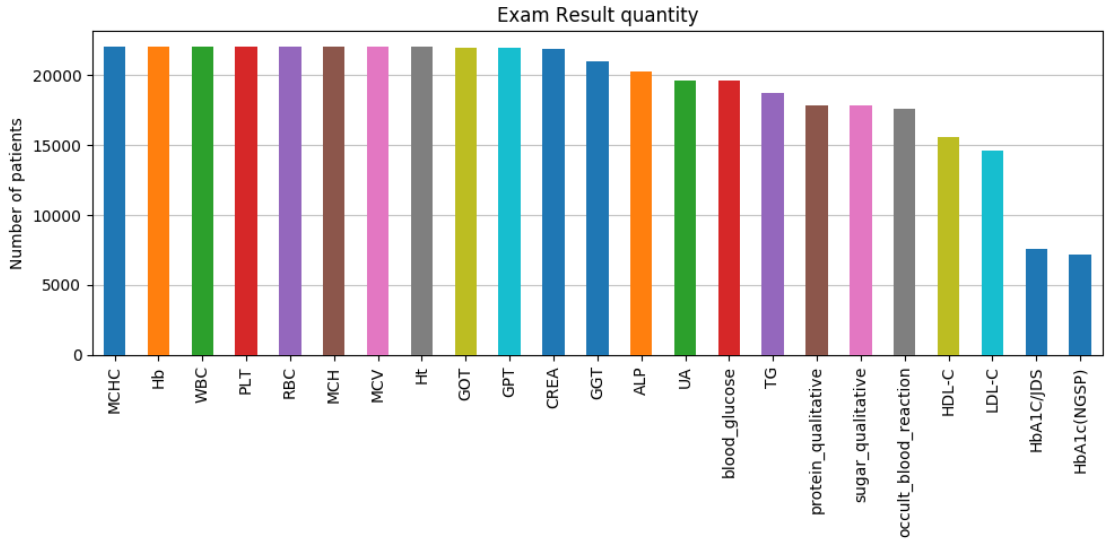


Figure 5.5: Types of exams and amount of patient that have results for that exam

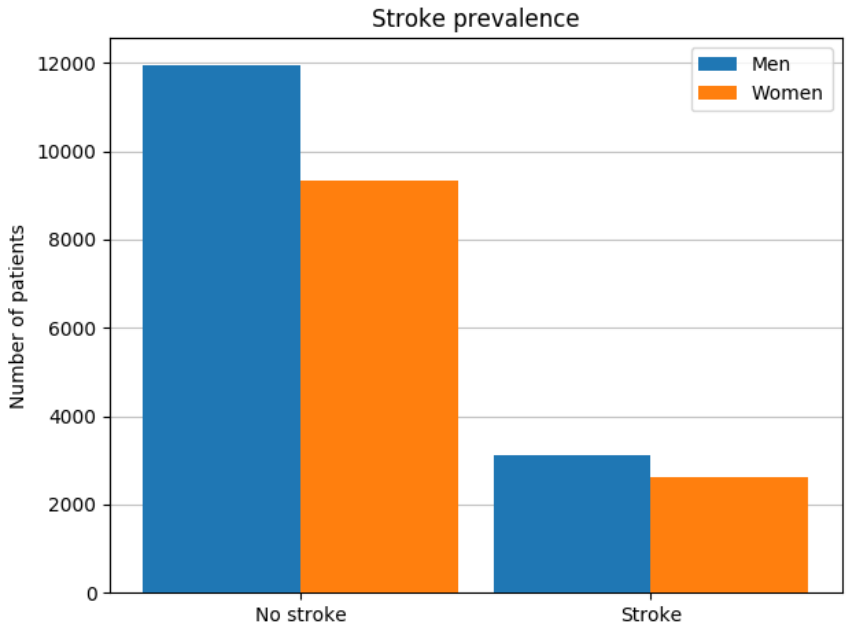


Figure 5.6: Stroke prevalence in the dataset.

5.4 Embedding

Embedding is the process of representing an observation or record as structured data, commonly as a vector. Considering the data sources presented below, a straightforward embedding is to consider each patient as an individual vector, and their columns are their attributes of all sources, i.e., their exam results, their past diseases, and their demographics.

The problem with this representation is that for one patient, there are many records of exam results, diseases, and demographics within the history of data. In order to solve this for

the case of exam results and demographics, we propose to use the most recent result, since most recent results should be more representative of the current condition of the patient.

For the case of disease history, the data is presented like a log showing the date when a patient was diagnosed with a particular disease. This representation cannot be inputted directly into the vector. Instead, we can consider to have a column for each illness and fill it with a 1 if the patient has had the disease in past or with a 0 if not. A drawback of this approach comes from the fact that ICD-10 defines more than 14000 different codes, and then the dimensionality of the feature vector will be very high, which slows down the computations of all involved method method. Moreover, this representation is very sparse because certain specific diseases occur only in a few patients, that behavior eases models to overfit their predictions. To solve this, using expert knowledge we select a group of diseases that are likely to be related with stroke occurrence, which are: Type 2 diabetes mellitus (ICD-10: E11), Cerebrovascular diseases (ICD-10: I60-I69), Ischemic heart diseases (ICD-10: I20-I25) and Diseases of arteries, arterioles and capillaries (ICD-10: I70-I79).

As a summary, Table 5.1 shows the details of each column of the embedding used.

5.5 Rules and Configuration

For this case the rules of the model were defined manually using expert knowledge instead of generating them automatically. Below is the description of how they were defined for all attributes presented in the previous section.

For exams that are qualitative or semi-quantitative such as protein qualitative, sugar qualitative, and occult blood reaction, for the past history of diseases such as diabetes, cerebrovascular diseases, cardiovascular diseases, and arteries-related diseases; and for gender; we create a rule for each possible outcome of these attributes.

For quantitative exams and demographics, we used expert knowledge to define the cutoff values between typical values and abnormal values. These cutoff values are also called reference intervals, and they are regularly used in the medical field, for example, the ones used internationally by Medscape [78]. The values we used are presented in Table 5.2.

Using these cutoff values we create a rule for each interval these values defines, for example for LDL-C there is a rule activated when LDL-C is smaller than 120, another rule is activated when LDL-C is between 120 and 160, and finally, another rule that is activated when LDL-C is greater than 120.

Finally, for quantitative measures, we also create rules to be activated when pairs of attributes are outside their normal ranges in order to search for more complex relationships in data. An example of these kinds of rules is a rule activated when UA is greater than 7 (above normal range), and PLT is smaller than 10 (below normal range).

This process generates a total of 558 different rules. The classifier was trained using Mean Squared Error as the loss function and using Adam optimizer with an initial learning rate of

Attribute	Description
Age	Age of the patient as of 2019
Gender	Gender of the patient
BMI	Body mass index
Body Fat	Body fat index in percentage
Weight	Weight of the patient in kilograms
Height	Height of the patient in centimeters
Waist Measurements	Waist Measurements of the patient in centimeters
Blood Glucose	Blood Glucose Test (latest value)
HDL-C	High-Density Lipoprotein Cholesterol Test (latest value)
LDL-C	Low-Density Lipoprotein Cholesterol Test (latest value)
Hb	Percentage of Glycosylated Hemoglobin (latest value)
Ht	Hematocrit Blood Test (latest value)
WBC	White blood cells count per microliter (latest value)
RBC	Red blood cells count per microliter (latest value)
PLT	Platelet count per microliter (latest value)
MCV	Mean Corpuscular Volume Test (latest value)
MCH	Mean Corpuscular Hemoglobin Test (latest value)
MCHC	Mean Corpuscular Hemoglobin Xoncentration Test (letest value)
TG	Triglycerides Test (latest value)
CREA	Creatinine Blood Test (latest value)
UA	Uric Acid blood test (latest value)
GOT	Glutamic-Oxaloacetic Transaminase Test (latest value)
GPT	Glutamic-Pyruvic Transaminase Test (latest value)
ALP	Alkaline Phosphatase Test (latest value)
GGT	Gamma-glutamyl Transpeptidase Test (latest value)
Sugar Qualitative	Sugar in Urine Semiquantitative Test (latest value)
Protein Qualitative	Protein in Urine Semiquantitative Test (latest value)
Occult Blood Reaction	Urine Occult Blood Semiquantitative Test (latest value)
HD Diabetes	Whether the patient has type 2 diabetes mellitus (binary value)
HD Cerebrovascullar	Whether the patient had Cerebrovascular diseases in the past (binary value)
HD Cardiovascullar	Whether the patient had Ischemic heart diseases in the past (binary value)
HD Arteries	Whether the patient had diseases of arteries, arterioles and capillaries in the past (binary value)

Table 5.1: Embedding for the Stroke risk assessment problem

0.001.

Attribute	Cutoff values
WBC	4000, 10000
Ht	38
MCV	88, 102
MCH	27, 32
MCHC	30, 35
PLT	10, 40
UA	2, 7
LDL-C	120, 160
CREA	0.5, 1.2
TG	100, 200
HDL-C	40, 60
GOT	30, 100
GPT	30, 100
ALP	200, 1400
GGT	30, 100
Blood Glucose	100, 126, 200
Hb	7, 10, 13
BMI	18, 25, 30, 35, 40
Age	80
Body Fat	11, 22
Waist Measurement	95

Table 5.2: Cutoff values for attributes based on normal medical ranges

5.6 Model Results

After defining the embedding and configuration for our model in this problem, we tested the model performance using a 5-fold experiment over the dataset. In other words, we split the dataset into a training set (80%) and validation set (20%) randomly. We repeat the process five times, variating which data records belong to the training and validation sets. For each fold, the following indicators were calculated: accuracy, sensitivity, specificity, F_1 macro, and the area under the ROC curve. Table 5.3 shows the average of these indicators among all folds.

Indicator	Value
Accuracy	0.854
Sensitivity	0.595
Specificity	0.878
F_1 macro	0.451
AUC ROC	0.875

Table 5.3: Performance results for stroke prediction problem

Figure 5.7 presents the results of the model as a Receiver Operating Characteristic (ROC) curve and a Precision-Recall curve. These are two commonly used methods to assess the performance in binary classification. The ROC curve illustrates the rate of true positives and false positives results when moving the decision threshold. The Precision-Recall curve shows the performance of the classifier over a specific class; in our case, the interested class is the "Stroke" class. Consequently, it shows the variation of the true positive rate with the total number of positive predictions.

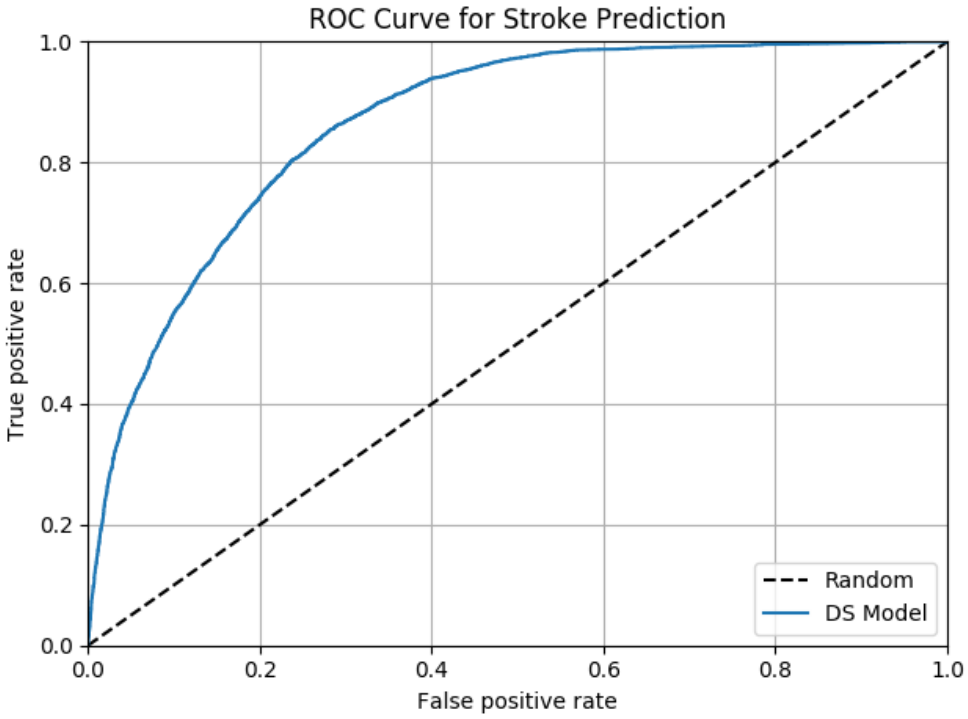


Figure 5.7: ROC curve for stroke prediction.

These first results hints that our model performs well in this scenario, being able to identify correctly the patients who are more likely to have a stroke from the healthy ones.

In order to validate these results, these results model will be compared to others' in two cases, which are detailed in the following sections. In the first case, we will compare our methods to other machine learning techniques using the same embedding. In the second case will compare our model to current methods for stroke risk prediction from different sources such as clinical procedures and other data science proposed solutions.

5.6.1 Comparison with other Machine Learning Models

The model was compared to other machine learning methods to validate the performance result obtained.

We will use the same methods and configuration presented in section 4.2 about traditional datasets and we will also use the accuracy and the area under the ROC curve as indicators

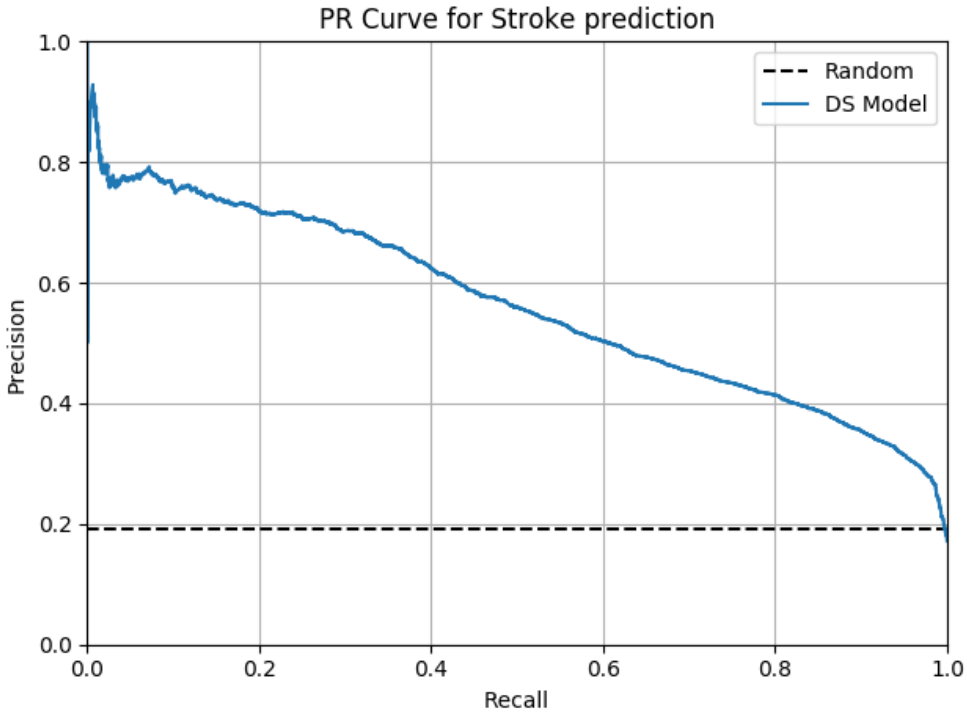


Figure 5.8: Precision-Recall curve for the class "Stroke".

for all methods.

These other benchmark methods do not support missing values on data. This could be mentioned as an advantage of our model over the others. However, to compare accuracy, we need to fill the empty data with some values or dropping these entries. We opted for the first option since removing data records change the dataset, and then results became incomparable. Moreover, in a real scenario, many patients have incomplete information, and any screening method for stroke risk should be able to assess this risk for these patients as well.

For filling the missing data, we will use the mean imputation strategy. This process consists of assuming that the best estimation for a missing value is the mean of the available data. Therefore within an attribute, we compute the average for the available information and use that value to fulfill the missing ones.

Table 5.4 shows the results of the different models and our model for the validation set. Furthermore, Figure 5.9 shows the ROC curve for all methods in the same chart to compare them easily.

From the analysis of these results, we can observe that our model is the one that achieves the best performance for this problem in both accuracy and area under the ROC curve. In the ROC curve chart, we can see that our model (blue line) is always over the other models showing that it is more accurate in the prediction of both classes.

We identified two main reasons why our model outperforms the other models. First, our

Model	Accuracy	AUC ROC
DSGD	**0.854**	**0.875**
RF	0.849	0.861
NB	0.618	0.838
KNN	0.794	0.717
MLP	0.821	0.813
SVM	0.820	0.574
Majority	0.793	0.500

Table 5.4: Result of different method for the stroke risk problem.

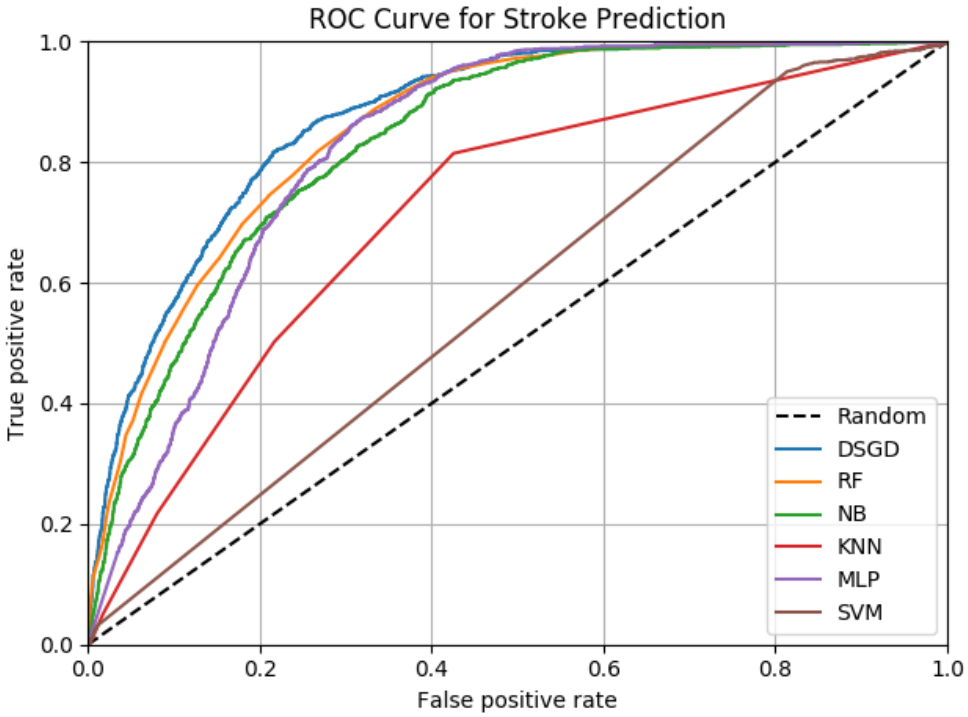


Figure 5.9: ROC curve for different method in the stroke risk problem.

model can handle missing values directly, which avoid the errors introduced by the mean imputation. In fact, we can see that the models that perform worst are KNN and SVM, which are based on geometry. In these cases, mean imputation causes many records to have a distance of 0 or colinearities in some dimensions, which introduces difficulties to the methods for performing correctly.

The second reason why we believe our method works better is that the rules we used to construct the model are meaningful and validated by medical studies. Every rule statement represents an actual condition that patients can have. This property helps the model to distinguish different patients better than the rest of the models.

5.6.2 Comparison with Deep Learning methods

To illustrate that the proposed model is in between of high interpretability and high accuracy, in this section we will compare it to a deep learning model which aims to be the most accurate and to a decision tree with aims to be the most interpretable and simple.

As we mentioned in the introduction, current deep learning models outperform many other machine learning models in terms of accuracy especially in computer vision and natural language processing. However, deep learning models usually operate with unstructured data like images, videos or free text. In our case, the data has a well defined structure since it comes from exam results and disease history. For this reason common deep learning techniques like convolutional neural networks and recurrent neural networks cannot be applied because they process unstructured data. We can use feed-forward neural networks because they work with tabular data, but in that case we will not be applying deep learning.

To apply deep learning techniques to our dataset, we will have to look into each attribute. For continuous attributes, we cannot transform this number since it is already the best representation of the variable. But for categorical attributes we can represent them as vectors, i.e., for each category we assign a vector of a fixed length that represents this category. This technique is called categorical feature embedding [79], and the aim of the model is to learn the optimal representation of each category that produces the best results for the classification problem. After converting all the categorical attributes to vectors they are appended to a single vector along with the continuous variables and this is the new input for a feed-forward network. Using this structure, the back-propagation algorithm can go back up to the feature embedding parameters, changing their values according to the data and the error in the prediction. Note that using categorical feature embedding we can represent missing information as a new category, so the neural network gains the ability to handle absent of data for categorical attributes.

Having categorical attributes, we can then build a neural network model using categorical feature embedding for all these attributes, we call this model "CatEncoder". We will map each categorical attribute to a vector of length equal to the ceiling of the half of the amount of categories. For example the attribute "the patient has diabetes" can be "true", "false" or "null", then it will be mapped to a 2-length vector, since 2 is the ceiling of 3/2.

Note that continuous variables cannot handle missing values even when categorical attributes can work with them using this technique. Therefore, we need to use the mean imputation strategy to fill the missing information among these continuous attributes as explained in the previous subsection.

Another approach to use categorical feature embedding is to discretize all the continuous attributes into ranges, then to consider these ranges as categories and finally apply categorical embedding to these variables. We will call this model "AttrEncoder". Applying this strategy allows us to transform every attribute into a vector that our neural network will learn how to represent. In addition, we can represent null values as vectors without needing to put values to fill the missing information. To choose which ranges to use for each attribute, we can use the same cutoff values our model uses presented in Table 5.2 which come from the

medical literature. Using these cutoff values, the comparison between our proposed model and "AttrEncoder" is fair since both can handle missing values and both include medical knowledge in their representations.

The models "CatEncoder" and "AttrEncoder" were tested over the dataset. Table 5.5 shows the performance of our model (DSGD) and these other two models, and the Figure 5.10 shows the ROC curve for the three methods.

Model	Num. Parameters	Accuracy	AUC ROC
DSGD	1674	0.854	0.875
CatEncoder	4628	0.897	0.912
AttrEncoder	5892	**0.910**	**0.922**
Majority	—	0.793	0.500

Table 5.5: Comparison of results of deep learning method for the stroke risk problem.

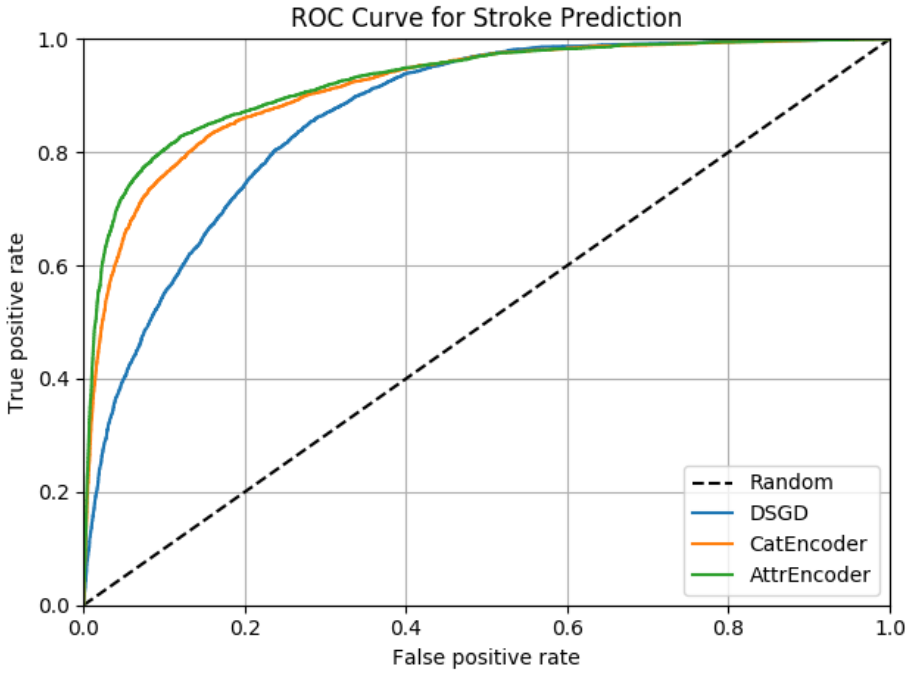


Figure 5.10: ROC curve for Deep Learning methods in the stroke risk problem.

Table 5.5 also shows a new column for the number of parameter of each model. Our model and neural-network based model have inner parameters that are adjusted to the optimum values in the training phase. In our case these are the mass values, and in the other methods these are the weights of each connection between neurons and the encoding values for the feature embedding. From this column we can see that both "CatEncoder" and "AttrEncoder" are much more complex models that have more than 2.5 and 3.5 times the number of parameters our model uses respectively.

Analysing the results presented, we can note that these Deep Learning techniques outperforms our model in terms of accuracy, which is the expected results since they learn to

represent the categories more accurately than our model. However, the difference in accuracy is about 5% which implies that our model does not lose a lot of performance compared to these state-of-the-art models. Another remarkable result is that "AttrEncoder" performs better than "CatEncoder" which again supports the hypothesis that adding expert knowledge to models may increase performance.

From Figure 5.10 we can see exactly where these models perform better that our method. The greatest difference between the curves is presented for low values of False positive rate (FPR). For example, at the value of $FPR = 0.1$ our model reaches a True positive rate (TPR) of almost 0.6, while the other models are close to 0.8. In other words, if we let the model to misclassify 10% of the "No Stroke" cases as "Stroke" ones, then our model is able to find the 60% of all "Stroke" cases, while the Deep Learning methods are able to find the 80% of these cases. This result give us a more clear interpretation of the

Finally, it is important to say that even though these methods undoubtedly reach higher precision, they do not provide any interpretation of their decisions which is one of the aim of this thesis. We can hypothesize that the values of the resulting trained embeddings are related to the interpretation of the predictions. As en exploratory example, Figure 5.11 shows the resulting embeddings for the attributes "Body fat" and "HD diabetes".

Body Fat	Vector		
[0, 11)	1.3957	-0.4869	-0.184
[11, 22)	0.2553	0.1668	0.1232
[22, inf)	1.0255	-0.2085	0.9405
null	-0.8819	-0.7708	2.431

HD Diabetes	Vector	
0 (False)	-0.3637	-0.069
1 (True)	-0.0805	0.1667
null	-0.0547	-1.9375

Figure 5.11: Example of the resulting embeddings for two features of the dataset.

From these results we can see that there is no indication that they are related to the decisions of the model. The values seem very erratic and without any clear pattern. Also, these values are linked to the following parameters of the network (e.g., the weights of the neurons that take these values as input) so it is not possible to separate the embeddings completely for this analysis. This is characteristic of black-box models, thus getting interpretability from the model itself is difficult.

5.6.3 Comparison with other Stroke risk assessment methods

Many models have been proposed to evaluate the risk a patient has of getting a stroke. These methods offer baselines for our model since they are created with the same purpose. In this section, we present a subset of these methods. Some of them are very basic, but they are widely applied by clinicians, while others are complex and use state-of-the-art techniques of data science.

CHADS_2 is a score metric to evaluate stroke risk prediction proposed by Gage et al. [80]. The name CHADS_2 is an acronym for the method itself, the letter "C" assigns a point to patients who have congestive heart failure, "H" assigns a point to patients with hypertension, "A" is for patients aged 75 or older, "D" is for patients with diabetes mellitus and finally

"S_2" assigns 2 points to patients that had stroke, ischemic attack or thromboembolism in the past. CHADS_2 score is widely used in medical field to predict stroke occurrences because of its simple formulation and evaluation. A drawback of this method is that it only uses five variables for decision making.

Letham et al. [81] used Bayesian Rule Lists (BRL) to develop an interpretable model to predict stroke risk within a year for patients diagnosed with atrial fibrillation. The method uses decision lists, which consist of a series of "if . . . then . . ." statements, then a generative model called Bayesian Rule Lists that automatically produces a posterior distribution over possible decision lists allowing inferences to be made about stroke risk.

Another approach to predict stroke risk was proposed by Peñafiel et al. [47]. In their work, a model based on Dempster Shafer Theory was proposed, the model operates using rules that were built using the training data statistics, we will call this model as DS-Stat. When evaluating a new case, the corresponding rules are combined using Dempster Rule to provide a final mass assignment function, then the belief for this assignment was computed and given as the model response. This model achieves an accuracy of 61%, and an important feature of the model is that it was interpretable. The authors showed which were the essential rules associated to stroke and how they were verified with medical literature.

Teoh [82] also proposed a method to predict stroke risk. In his work, a Recurrent Neural Network (RNN) [83] was used in combination with a custom loss function. The model uses all available data for a patient, such as exam results and diagnosis structured like a time series for the RNN, then fully-connected layers are applied to predict the class. The custom loss function was tested and compared with classical ones. This function was proven to behave better since it takes into account the context of data more accurately. Interpretability is also covered in this work by eliminating specific attributes from the feature vector and observing the change in the accuracy indicator. The best result of the model achieves an area under the ROC score of 0.669.

Our model was compared to other existing or proposed model presented in section 2.1 for screening stroke. We used the area under the receiver operator curve (AUC) as a comparison metric because it is a standard metric used when comparing binary classification, and also it is a useful metric when analyzing unbalanced datasets like our case where most of the patients are healthy. Table 5.6 shows the AUC score for each method; our method outperformed other methods.

Method	AUC
CHADS_2 [80]	0.721
BRL [81]	0.756
DS-Statistical [47]	0.612
RNN with custom loss function [82]	0.669
DSGDt	**0.875**

Table 5.6: Comparison of Results for different stroke risk prediction methods

From the analysis of Table 5.6, we can note that our model outperforms all the other solutions for screening stroke.

Chapter 6

Validation

In this chapter, the interpretability results of the proposed model for the stroke risk prediction problem will be presented, evaluated, and validated. Three different methodologies will be applied to these results to test whether they are useful. The first method will compare our interpretable results with the results of other explanation methods. In this case, we will show the results of app plying the partial dependency plots and LIME strategies, which are already accepted methods and compare our our approach with them. The second test will be checking if current medical works support the most contributory rules obtained by our model. Furthermore, we will present the most contributory rules to medical experts in a survey asking whether or not they think these rules are correct for stroke risk prediction.

6.1 Interpretability Results

Besides the model performance and the reached accuracy, our model is able to extract the most important rules while predicting whether a patient will have a stroke.

In order to do this, we will follow the process explained in section 3.3. In other words, after the training phase, all masses have been adjusted to find the values that minimize the error when predicting stroke. That values, according to Dempster Shafer Theory, are assignments to every subset of possible outcomes; in our case, they will be the null set, the singleton for "No stroke" class, the singleton for "Stroke" class and the complete set. The mass of a singleton measures the contribution to a particular outcome, whereas the mass of the complete set measures the uncertainty to any outcome.

Having all these mass values, the procedure to extract the most contributory rules for a specific class is to compute the contribution score γ defined in section 3.3 and then sort all rules by this indicator.

Table 6.1 and Table 6.2 show the top 7 most important rules for the prediction of the classes "Stroke" and "No Stroke" respectively according to γ indicator.

#	Rule	Mass No stroke	Mass Stroke	Uncertainty	γ value
1	HD cerebrovascullar = 1	0.000	0.755	0.245	0.755
2	10 < PLT < 40	0.002	0.410	0.589	0.411
3	Hb > 13	0.007	0.318	0.674	0.322
4	HD diabetes = 1	0.003	0.253	0.743	0.255
5	Body fat > 24	0.002	0.236	0.762	0.237
6	HDL-C > 60	0.000	0.213	0.787	0.213
7	Waist measurement > 95	0.007	0.207	0.786	0.210

Table 6.1: Most important rules for class "Stroke"

#	Rule	Mass No stroke	Mass Stroke	Uncertainty	γ value
1	PLT > 40	0.853	0.000	0.147	0.853
2	GOT > 100	0.733	0.000	0.267	0.733
3	Hb < 7	0.676	0.003	0.321	0.677
4	GPT > 100	0.560	0.000	0.440	0.560
5	ALP > 1400	0.495	0.003	0.502	0.496
6	7 < Hb < 10	0.465	0.002	0.533	0.465
7	WBC > 10000	0.457	0.002	0.541	0.458

Table 6.2: Most important rules for class "No Stroke"

From these tables, we can observe that the model can produce meaningful explanations for the whole problem and clearly distinguish and sort the contributory rules from the useless rules for each class.

The values for the γ indicator are higher in the "No Stroke" classification than the "Stroke" class. This consequence is likely to happen because the data is heavily unbalanced; in our exploratory analysis presented in section 5.3, we showed that 79.4% of the patients did not have a stroke.

For the case of the "Stroke" class, the rule "HD cerebrovascular = 1", which means "the patient had a cerebrovascular disease in the past" is by far the most important rule while predicting this class with a γ score of 0.755. The second most contributory rule which relates the count of platelets in the blood (PLT) to low range of values has a contribution of 0.411, which is almost the half of the contribution of the first rule. The following rules decreases its contribution more slowly. For the case of "No Stroke" rules, there is not a clear rule that defines that class. Alternatively, many rules achieve high contribution values, meaning that there exist many more distinct configurations for these variables that determine a healthy patient.

Furthermore, many of the rules that appear for the "Stroke" class also appear in their contrary form for the "No Stroke" class. This is the case of the count of platelets (PLT), which for lower values relates to the occurrence of strokes and for higher values related to healthy patients. Also, the higher values of Hemoglobin concentration (Hb) are related to stroke, whereas lower and average values are related to healthy patients.

Although the obtained rules that explain the decisions made by the model seem to be meaningful and coherent, we need to prove that. We will test these interpretability results in two ways. The first one is to show that the rules obtained by this process are the most contributory to the classification. In order to do that, we will compare our rules to several explanation models, such as partial dependency plots and LIME. The second way these interpretability results will be tested will be their applicability and correct meaning. In this case, we will contrast the rules with current medical knowledge in order to see whether they are in accordance by presenting these rules to physicians and experts and ask them if they consider them correct.

6.2 Explanation models

6.2.1 Decision Trees

The first benchmark model to compare our interpretability results is a Decision Tree. As we mentioned in the introduction, Decision Trees is one of the simplest and more interpretable models to make inferences and predictions. Also, they are widely used in non-computer science fields such as medicine, where many of the procedures to apply in certain cases are obtained from the result of a Decision Tree.

Decision Trees are binary trees in which each inner node represents a condition that can be evaluated using the input data, each branch represents if the condition is true (left branch) or false (right branch). In a classification problem, leaves nodes show the predicted class.

Decision Trees can be built by using expert knowledge or using a certain algorithm to generate the decision rules for each inner node inferred from the data features. These algorithms are often called tree algorithms. The most known tree algorithms are ID3, C4.5 and CART. All of them are improvements for the previous versions, but they remain the basic principles for split generation. ID3 was one of the first tree algorithms proposed by Ross Quinlan [84]. The greed algorithm creates a multi-way tree, finding for each node the categorical feature that will yield the largest information gain for categorical targets. After that a pruning step is applied to remove nodes with low support, avoiding overfitting. The C4.5 algorithm extends the ID3 algorithm, and the same author proposed it [85]. C4.5 algorithm removed the restriction that features must be categorical by dynamically partitioning continuous variables into discrete ranges. The C4.5 algorithm uses information gain as the target metric to find the best splits. Finally, the CART algorithm is an extension of C4.5 that allows to create decision trees for classification or regression [86]. It uses the Gini impurity as metric for splits instead of information gain or entropy as the previous algorithms.

We will explain CART algorithm in more detail because this will be the strategy we will use to compare our results. To create a new node based on the training data with certain attributes, the algorithm looks through all the attributes testing all possible splits. For categorical attributes, it tests making a split using all categories. For continuous attributes, the algorithm sorts their values and tests all breaks between two different values. To measure

how good is a split, the algorithm separates the training data according to this split, using this separation it computes the Gini impurity of each children using the following formula:

$$I = 1 - \sum_{i=1}^{K} p_i^2 \tag{6.1}$$

where K is the number of classes, and p_i is the proportion of the samples that belongs to the class i. Then the Gini impurity for the split is defined as the weighted average of the Gini impurity of the children weighted by the number of samples in each node. After computing the Gini impurity of all splits the algorithm selects greedily the split which has the least impurity and uses it as the split for the node. The process continues recursively in each child node, these nodes update their training data to use only the data that satisfy the parent node, thus every time we descend in the tree less data is available for testing new splits. We can define several base cases or conditions to stop recursion. For example, the algorithm should stop when the impurity is 0 (i.e. there is a perfect split to separate the classes), when the tree has reached a certain maximum depth or when the remaining data for testing the split is low (e.g. the 5% of the original dataset size). When one of the above conditions is met, we do not create a split of that node and instead we create a leaf node with the predicted class corresponding to the majority of the samples in this node (i.e. $argmaxp_i$). Note that because of the greedy behavior of the algorithm, the resulting tree may not be the optimum Decision Tree for the problem.

We created a Decision Tree using the CART strategy to find the splits. We separated the data in a 80% for building the tree (training) and the other 20% of the samples for validation. We also impose the condition that all nodes must have at least 5% of the samples of the dataset, and the maximum depth is 4.

The Figure 6.1 shows the resulting Decision Tree. In this figure, each inner node present the following information: the decision rule of the node, the Gini impurity of the split, the proportion of the samples that were used to generate the split, the proportion of samples of each class (in the form $[p_{\text{No Stroke}}, p_{\text{Stroke}}]$), and the predicted class if the process is terminated up to this node. Leave nodes present the same information except for the decision rule. The color of the nodes are mapped to the classes. Green is for "No Stroke" class and red is for "Stroke" class, darker colors mean more certainty in the prediction.

From the figure we can clearly understand the classification process the tree is performing proving that it is one of the most interpretable methods.

Note that because of the class imbalance of the dataset, there is only one leaf node in the tree with the outcome "Stroke". All the others predict "No Stroke". However, as in many machine learning processes, we can also predict a probability of belonging to the classes as the proportion of the samples in each leaf node.

This Decision Tree reaches an accuracy of 83.9% and an area under the ROC curve of 79.9% when tested over the validation set. Compared to the results presented in section 5.4 our method and Random Forest achieve better accuracy and AUC ROC. However, it is

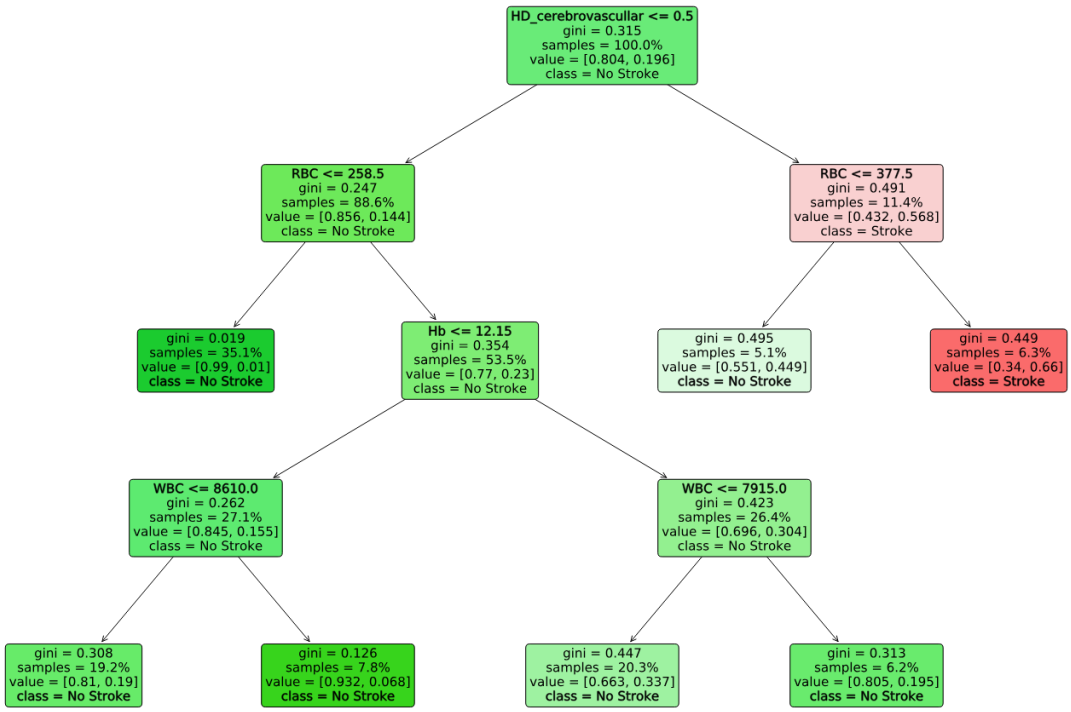


Figure 6.1: Decision Tree for stroke risk problem

important to notice that this simple Decision Tree outperforms other methods such as SVM and KNN for this particular classification task.

Comparing the interpretability, we can see that the first node of the Decision Tree has a decision rule concerning the "HD cerebrovascular" attribute which separates patients who had a stroke before from those who not. This rule coincides with the most important rule to predict the stroke class according to our model (see Table 6.1). This result shows that our method can find the same most important attribute as the CART algorithm. Then, the next level of the tree shows two nodes with the "RBC" attribute for the rule. According to the Decision Tree, this is the second most important attribute for prediction, but in this case this attribute does not appear in our top 7 most important rules of either "Stroke" and "No Stroke" classes. Thus, this result reveals that our model and the Decision Tree differ. One explanation for this difference is that our model does not separate the data, it applies all rules evenly to patients that had stroke before or not. Then it may be more difficult to find a rule regarding the "RBC" attribute because of that restriction. Finally, analysing the deeper levels, we see that the features that appear are "Hb" and "WBC". The "Hb" also appears as the third most important rule for both classes in our method. The "WBC"' appears as the seventh most important rule for the "No Stroke" class.

From this experiment we showed that the interpretability of Decision Trees are more simple to understand than our model. However, most of the attributes that appear as nodes in the tree also appear as important attributes in our model, showing that our interpretability is related to the one found in a Decision Tree. In terms of performance, our method outperforms the presented Decision Tree having over 7% more in AUC ROC metric. In this experiment we see the trade-off of interpretability and accuracy, having a harder to interpret method

but more accurate than Decision Trees. This is the opposite case from the one presented in section 5.6.2 about comparing with deep learning methods.

6.2.2 Partial dependency plots

Partial dependence plots (PDP) is a strategy to discover the marginal dependency a prediction or classification has with respect to a particular attribute [87]. PDPs were initially defined for regression tasks since they show the relationship between an attribute and the target variable by modifying the input like a sensitive analysis. However, this idea can be extended easily to binary classification since we can use the probability of belonging to the positive class as the target function. In this probability function, the value 1 means that the classification is accurate for the class, while the value of 0 means that the classification supports the other class. In our case, we will use the probability of belonging to the "Stroke" class as our target function.

In order to build a PDP for a particular feature, we require the trained model and a validation set. The first step is to obtain the domain of the target attribute. After that, we divide the domain into several ranges; in our case, we opted for ranges with an equal number of samples. For each of the cutoff values, the process requests the model to perform the prediction modifying the target attribute and forcing to have the cutoff value for all the record on the validation set. For each of these altered samples, we observe the change produced on the target function, which is called as the dependency or marginal value. If this change is positive, then the difference contributes to the prediction of the "Stroke" class, and a negative value contributes to the "No Stroke" class. If the change is close to 0, then the difference in that variable does not affect the prediction, and thus, we can state that this feature is worthless for prediction.

After computing all the changes in the target function produced by the alteration of the feature, we can plot the dependency on the y-axis and the cutoff values on the x-axis and connect the same samples. The result is a chart that shows the contribution of the variable to prediction while moving value of the attribute. Along with these lines, the average of the dependency in cutoff value is presented to visualize better the trend.

The PDPs for the features PLT, Hb, Body fat, HDL-C, GOT, and CREA, are presented in Figure 6.2. We used eight cutoff values with an equal number of samples to build the plots, and the x-axis was altered to display better these quantiles.

From the analysis of the PDP for PLT, we can observe that for lower values, this variable has a positive dependency on the stroke occurrence, whereas, for higher values, the dependence is strongly negative. This behavior is what we exactly found on the Tables 6.1 and 6.2 because for the "Stroke" class, our model declares that lower values of PLT affect the prediction, while for the "No Stroke" case higher values are important.

Observing the PDPs for Hb, Body fat, and HDL-C, we note that higher values have a positive dependency on the stroke risk. Once again, this result is the same as our model reported in Table 6.1. Unlike the previous case, the dependence for these three variables is

Figure 6.2: Partial Dependency Plots for the features PLT, Hb, Body fat, HDL-C, GOT, and CREA.

lower, which matches with the result of the model since these variables are less influential than PLT.

The case of the PDP for GOT is similar but for the "No Stroke" case. In this chart, we see that higher values generate negative dependency, which means that contributes to the prediction of the "No Stroke" class. Our model anticipated this result since it appears in Table 6.2.

Finally, the PDP for CREA was presented to show the dependency of a variable that our model reports to be is useless. Note that the feature CREA does not appear in either of the Tables 6.1 and 6.2, then the model tells its contribution is negligible. When we observe the PDP, we can conclude the same, note that even when the process says that exists a positive dependency, the average value of this dependency is 0.05, which is a value very close to 0, indicating that the dependency is weak.

From this experiment, we can conclude that PDPs obtains in general the same explanations for the prediction than the proposed methodology for the model. An advantage of our methods is that it is part of the model itself, then the computational cost for producing that explanations are much lower. On the other hand, PDPs require the model to perform many predictions over the validation set, which makes them very expensive.

6.2.3 LIME

This sections compares the explanations of the model with the ones obtained using LIME. Since LIME gives an explanation for single instances instead of the entire system, some data records (i.e., patients) will be selected to produce the explanations. Then they will be compared to our interpretability results. Also, as these data records are part of our validation set, we know the class they belong to, thus several experiments can be performed. For example, explaining a true-positive or true-negative or failure cases, i.e., the false-positive and false-negative cases.

The first variant to be compared will be true-positive cases for the class stroke, i.e., patients who had a stroke within the next year, and the model classifies them correctly. We have selected randomly four true-positive instances, and we request LIME to explain that instances using our trained model. These explanations are presented in Figure 6.3. In each chart, the y-axis shows the top 10 most important rule attributes according to LIME, and the x-axis shows their contribution to each class, positive values are rules that contribute to the Stroke class.

From the analysis of these results, we can observe that in all cases, the most important attribute is "HD cerebrovascular" which is the attribute that indicates whether the patient had a stroke in the past. When this attribute has a value of 1, LIME assigns a high contribution to the prediction of the Stroke class. The second most contributory attribute in all cases was "PLT", which is the count of platelets in the blood. In these cases, lower values of PLT support the prediction of the Stroke class.

This result is identical to the one observed in Table 6.1 since the most important rules considers the same attributes, namely the HD cerebrovascular, PLT, and Body fat. The main difference with our results is that PLT seems to have an importance score slightly lower than HD cerebrovascular; in our case, this importance is about half.

The next case to be tested is a true-negative patient, i.e., a healthy patient that the model detects that he or she will not have a stroke within the next year but did not.

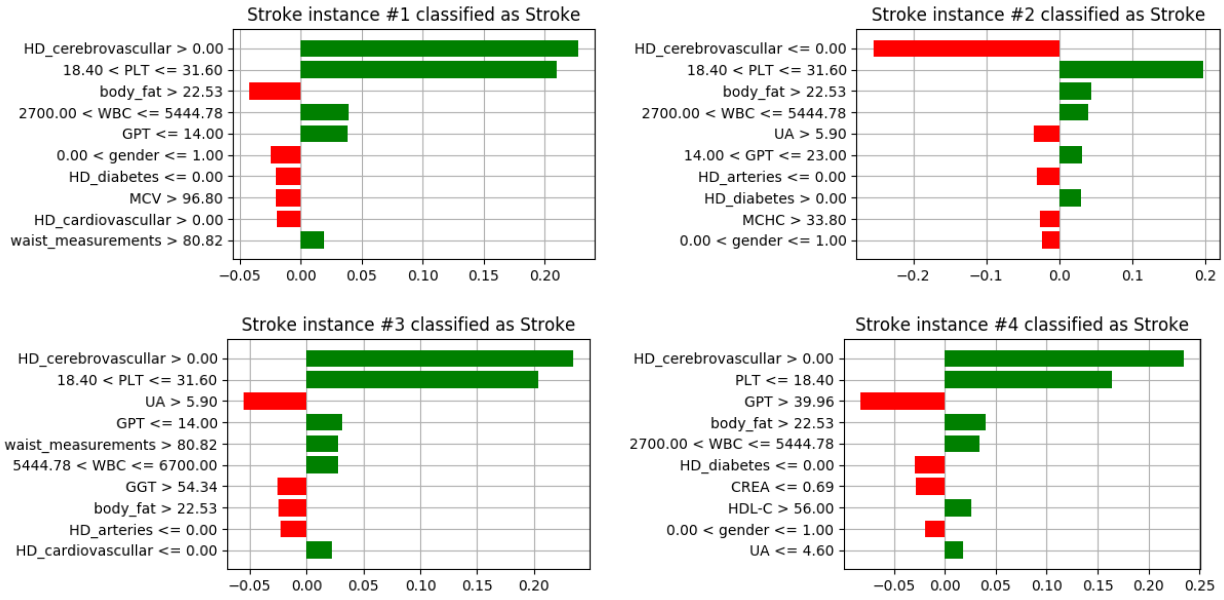


Figure 6.3: LIME explanations for Stroke instances classified as Stroke

Similar to the previous case, Figure 6.4 shows the results of the explanations models for four patients sampled randomly. In these charts, positive values indicate contribution to the "No Stroke" class. However, for better comparison among the experiments, we will keep the colors of the bars unchanged, which means that red bars are associated with "No Stroke" and the green ones are for the "Stroke" class.

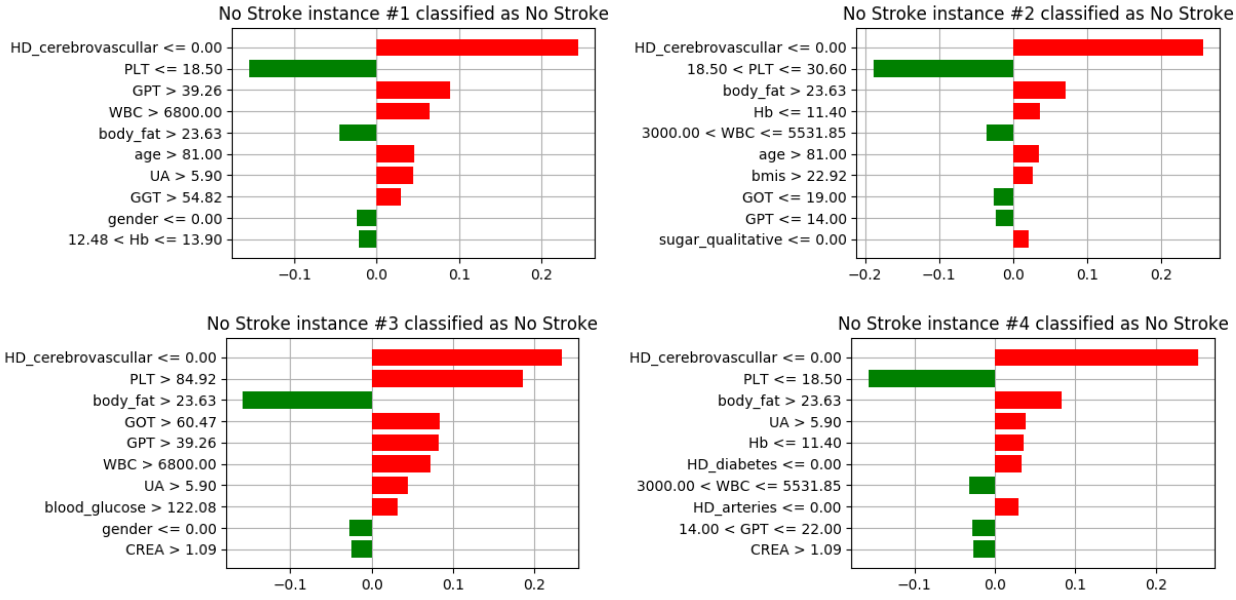


Figure 6.4: LIME explanations for No Stroke instances classified as No Stroke

In this case, the most useful feature for prediction is "HD cerebrovascular" similar to the previous case. However, in this case, a value of 0 indicates a low stroke risk, being the opposite statement compared to the last example. Although this is not a rule our model

obtains from Table 6.2 for the "No Stroke" class, it can be explained by the fact that it is the opposite of the most important rule for the other class. Moreover, we can observe in the explanations that high values for PLT, GOT, and GPT, are related to the prediction of the "No Stroke" class, similar to the rules obtained by our model.

One of the most remarkable result from the analysis of these explanations is the case of the attribute Body fat. If we look in detail, the statement "body fat > 23.63" appears in all four cases. However, for the instances #1 and #3, LIME assigns importance to the "Stroke" class, and for the instances #2 and #4, the same rule has importance on the prediction of "No Stroke" class. This result seems to be contradictory, but we have to recall that LIME finds its explanations locally in the proximity of the instances. Then, in the proximity of instances #1 and #3 this variable is related with the stroke occurrence, whereas, in the vicinity of the other two, the opposite happens.

Even though we can understand why the above happens, this can be considered a big blunder for the interpretability. This is because the next time a human or expert see a patient with a high value of body fat, they would not be able to make any inference about the stroke risk of the patient. This example shows one of the main, if not the most important, drawbacks of local explanations. This kind of contradictions does not happen in explanation methods for the entire model, like is our case.

As explained above, the selected patients to be tested are sampled from our dataset, thus we know the real class they belong to, and therefore, we know whether the classifier predicts them correctly. Then, in addition to the previous examples, we can present cases where our model fails and observe the explanations for those instances.

Figure 6.5 and 6.6 present the explanation for false-positive and false-negative cases respectively.

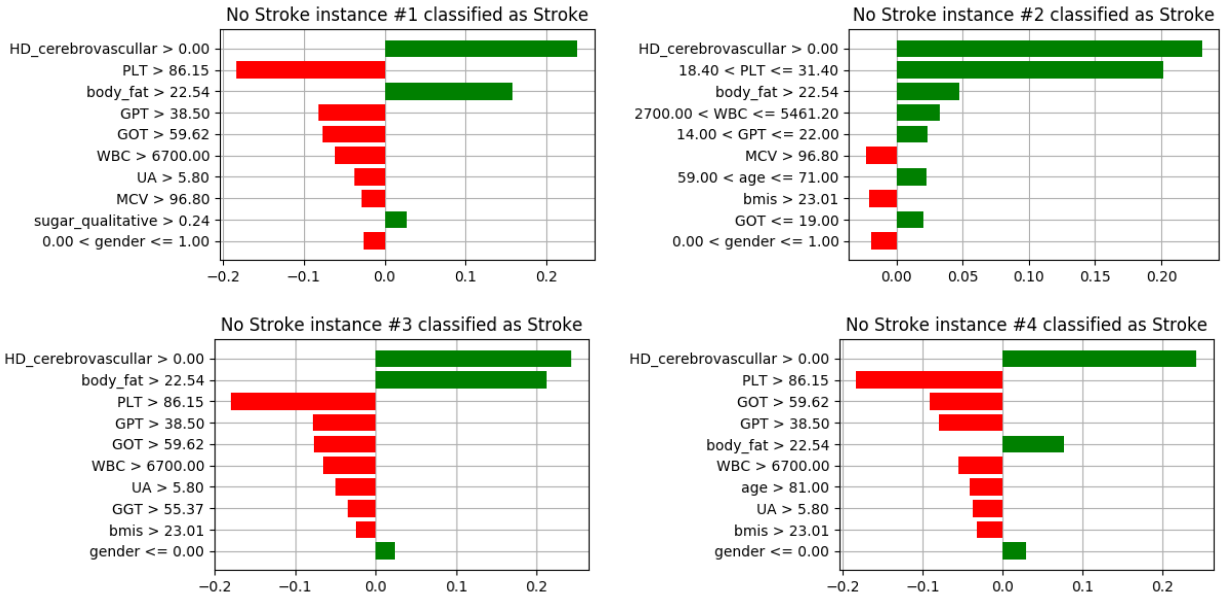


Figure 6.5: LIME explanations for No Stroke instances classified as Stroke

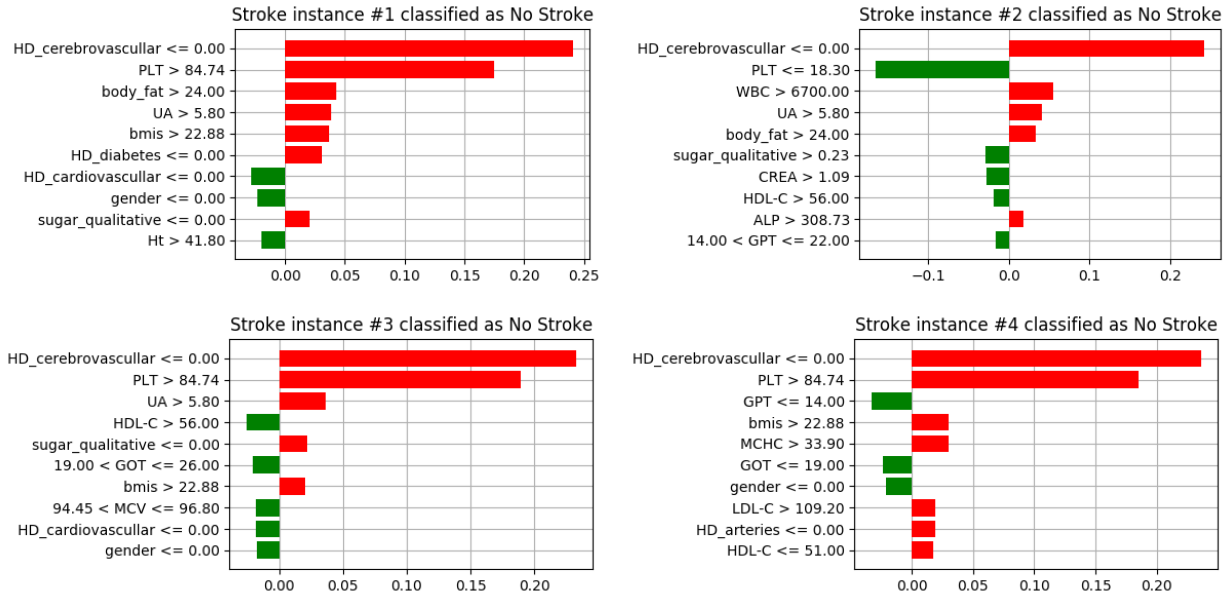


Figure 6.6: LIME explanations for Stroke instances classified as No Stroke

In these failure cases, we observe that the most representative rules, for example, the ones regarding HD cerebrovascular and PLT, are preserved. Moreover, we can notice that there are two groups of wrong classifications. First, we have cases where the model supports both classes almost equally, like the instances #1, #3, and #4 for Figure 6.5. In all these cases, the contribution of the rules for the two classes is more balanced than the other cases, which suggests a high uncertainty in this prediction. In the rest of the examples, we find a clear tendency to a specific class, especially in false-negative cases, which implies the "Stroke" class is hard to detect using these simple rules and the available data.

In summary, LIME explanations, in most samples, coincide with the explanations given by our model. We observe that local explanations are worse than explanations for the whole model since the first ones can provide us with information about contradictions while explaining instances. Finally, the instance-level explanations help us to recognize and analyze the cases where the model fails.

6.3 Contrasting the Model with Medical Literature

Besides comparing our interpretability results with other computational-based explanation models, we can also evaluate these results by their real meaning. In other words, each one of the rules concern attributes that have a meaning, since they are exam results or conditions of the patients. Then, the rules of our model describe medical statements or hypotheses that can be verified or refuted based on real medical works reflecting the current knowledge of stroke risk in the medical field.

In this section, we will analyze from the medical point of view each one of the rules presented in Table 6.1 reported as the most important for predicting the "Stroke" class.

6.3.1 Cerebrovascular disease in the past

The first and most influential rule for classifying stroke is "HD cerebrovascular = 1". This rule states that patients who had suffered a cerebrovascular disease, including stroke before, are more likely to have another.

In the medical field, a recurrence of a disease happens when a patient had a disease, and then he or she has it again. Clinicians and experts study recurrence for most lethal diseases, including stroke. They also define the recurrence rate as the fraction of patients that suffered the illness again within a specified period of time.

Stroke recurrence is a well-known and documented clinical effect [88, 89]. Within two years, the recurrence rate among these patients is about 14% [90], and within five years, it increases to approximately 25%, which is at least five times more likely compared to patients who have not experienced a stroke before. Mortality also increases after a first stroke episode. In ten years, patients with stroke have a relative risk of death of 1.7 compared to healthy population [91].

All these facts point that knowing a patient has suffered a stroke is a significant piece of evidence for increasing their risk and then for the classification itself.

6.3.2 Low values of platelets

Rule 2 considers that a low platelet counting is related to the stroke risk. Platelets play an important role after a stroke occurs because experts believe that they participate in the thromboembolic creation that may initiate stroke falls. Although the relationship between the count of platelets in blood and stroke is still unclear, several studies have found that after the stroke occurred, the platelet count tends to decrease significantly. For example, D'Erasmo et al. find an inverse correlation of -0.41 between platelets count and Hyperfibrinogenemia, which is related directly to the stroke event [92].

6.3.3 High values of Hemoglobin concentration

Rule 3 states that high values of Glycosylated Hemoglobin concentration are associated with a higher risk of stroke.

Similar to previous cases, there are prospective studies that validate this conclusion. For example, Rocco et al. [93] present a work that declares that a high concentration of Glycosylated Hemoglobin (HbA_{1c}) is an important factor for prediction of symptomatic intracerebral hemorrhage and acute stroke. The study also reports that Glycosylated Hemoglobin is a better predictor for stroke than blood glucose or a history of diabetes mellitus.

6.3.4 Diabetes condition

Rule 4 is the next most influential rule for screening stroke according to our model. This rule states that patients who have been diagnosed with diabetes are more likely to have a stroke. Similarly to other rules, several studies validate it.

In particular, Abbott et al. [94] present a 12-year follow-up study of 690 patients with diabetes and 6908 nondiabetic subjects. They conclude that the relative risk of thromboembolic stroke for those with diabetes compared with those without diabetes was 2.0 (95% confidence limits, 1.4 to 3.0).

6.3.5 High level of body fat

Rule 5 relates a high level of body fat with a high risk of stroke. High levels of fat in blood and their accumulation is often designated as a strong predictor for acute stroke. For example, Walker et al. present a study relating obesity to stroke risk among US men [95]. They found that the relative risk of obesity patients is 1.29 relative to healthy patients.

As another example, Folsom et al. [96] performed a prospective study with 191 patients in order to determine associations of several factors, including fat distribution to ischemic stroke. They conclude that there is a relative risk of 1.74 when incrementing the body fat distribution. These findings validate our rule.

6.3.6 High rates of HDL-C

Rule 6 states that higher values of HDL-C are related to the risk of getting a stroke. This statement is the most controversial of the ones presented since most medical studies conclude that there is no evidence pointing that high levels of HDL-C are related to cerebrovascular diseases. Instead, several prospective studies found that low concentration of HDL-C correlates with a higher risk of atherosclerotic diseases, even when the mechanism of how this low concentration eases the disease occurrence is still unclear [97]. Therefore, these findings invalidate our rule.

However, let us recall that the patients we are analyzing are Japanese workers. Many medical findings are subject to the population where the study takes place. Thus some of these conclusions may not apply directly to other populations. The two studies that we presented before are about western people, which have different genetics and habits to eastern people. This fact of no representativeness could explain why our model finds this rule as necessary. For example, Saito et al. [98] say that despite low HDL-C is an established risk factor, evidence regarding stroke and stroke subtypes is very limited for Asian population. They hint that the relationship is inverse, which coincides with the rule obtained by our method.

Even when we can explain why this contradiction occurred, we still need to perform further

studies to understand this causality correctly.

Finally, this example shows one of the most significant characteristics of the interpretability proposed by our model because any of the statements that do not have a clear interpretation in the expert field of the problem can guide future new researches on that topic. This feature makes our model a helpful tool for knowledge discovery.

6.3.7 High values of waist measurements

The last of the top 7 rules for predicting stroke class, relates high values of waist measurements with a high risk of stroke.

Unlike the previous cases, waist measurement is a metric that helps to build or find problems with other indicators such as body fat, cholesterol, and triglycerides. Hence, waist measurement is not a condition that directly affects the risk of cerebrovascular diseases.

For example, adiposity is a condition derived from the abnormality of several indicators such as waist measurements, and body mass index. Studies have confirmed that adiposity was associated with a higher risk of total and ischemic stroke in men [99].

Moreover, it is valid to assume that high values waist measurements are related to other conditions such as high weight, obesity, and high levels of body fat [100]. Then, as we explained in the fifth rule, high levels of body fat are indeed related to high stroke risk.

6.4 Expert Survey

The final test to assess the interpretability results of our model will be an expert survey. In this survey, we will present medical experts, mainly neurologists, the rules that increase stroke risk according to the model results. For each of these rules, we will ask whether they consider their statements are true, false, or if there is no correlation (NA) to stroke risk, according to their experience and knowledge. Some of the conditions were intentionally inverted to prevent biased answers to one alternative (e.g., an expert that responds all questions as true).

Table 6.3 shows the questions of the test and our expected results according to the interpretability results obtained. The questionnaire was built using Google Forms, which is widely used for this purpose and allows us to share it easily. The survey is in Spanish because we requested Chilean physicians and neurologists to respond to it, given our possibilities to conduct this test. The exact design of the questions can be found in the appendix section.

Moreover, we add two other rules that are less contributory to the estimation of stroke risk to check the "no correlation" option and to verify that the most important rules discovered by the model are also the most accepted statements among experts.

Alongside the questions about rules, we ask the Chilean national identification number

#	Rule	Question	Expected Result
1	HD cerebrovascullar = 1	A patient who had a stroke in the past is more likely to have a stroke again.	True
3	Hb > 13	A patient who has a percentage of glycosylated hemoglobin (HbA1c) greater than 13% is more likely to have a stroke.	True
4	HD diabetes = 1	A patient who has type 2 diabetes mellitus is less likely to have a stroke.	False
5	Body fat > 24	A patient with a body fat index greater than 24% is more likely to have a stroke.	True
6	HDL-C > 60	A patient who has a high-density cholesterol (HDL-C) level greater than 60 mg/dL is more likely to have a stroke.	True
10	BMI < 18	A patient with malnutrition (BMI under 18) is less likely to have a stroke.	False and NA
11	WBC < 4000	A patient who has a white blood cell count (WBC) less than 4000 per microliter is less likely to have a stroke.	False and NA

Table 6.3: Expert Survey Questions

(RUT) to check that people who participated in the survey are medical practitioners. We verified this information using the official database of medical practitioners from the Chilean health ministry. We also ask experts for acknowledgment or anonymity in their answers and additional comments about the experiment.

A total of 16 participants correctly answered the survey. We appreciate the interest of these experts in answering the survey. Eight of these participants preferred to share their answers anonymously, whereas the other seven are acknowledged below.

- Alvaro Riquelme
- Oscar Loureiro
- Claudia Paris
- Andres Baloian
- Víctor Navia
- Violeta Diaz
- Mario Vergara Rojas

Table 6.4 presents the results of the relative frequency of each option for every rule. We

include an extra column called "validation rate" which is the result of the expected choice for each rule according to the column "Expected Result" of Table 6.3. For rules where the expected result is a single value, for example, Rule 1 has "True" as expected value, we use the result for this value as the validation rate. For rules which have more than one expected value (e.g., Rules 10 and 11), we sum the results of all expected outcomes for the validation. The validation rate column is added in order to compare these results quickly. Also, raw answers are in the appendix section.

#	Rule	True	False	NA	Validation Rate
1	HD cerebrovascular = 1	**100%**	0%	0%	100%
3	Hb > 13	**94%**	0%	6%	94%
4	HD diabetes = 1	0%	**100%**	0%	100%
5	Body fat > 24	**88%**	0%	12%	88%
6	HDL-C > 60	31%	**63%**	6%	31%
10	BMI < 18	12%	**63%**	25%	88%
11	WBC < 4000	6%	31%	**63%**	94%

Table 6.4: Expert Survey Results

From these results, we can observe that all of the experts verify Rule 1 regarding stroke recurrence and Rule 4 about diabetes condition, since they all agree that these factors increase the stroke risk.

Almost all experts also confirm that Rule 3 concerning high levels of Hemoglobin raises the risk of getting a stroke, only one of them chose the "no correlation" option.

Likewise, almost all specialists consider Rule 5 concerning high values of body fat, as significant for the prediction of stroke. In this case, two of them selected the "no correlation" choice.

The results about Rule 6 are the most controversial since only one expert said that there is "no correlation" between high levels of HDL-C and stroke occurrence. However, the other experts have divided opinions about whether this variable increases or decreases the risk of getting a stroke. 62.5% of them said that the statement is false, thus invalidating our result, and 31.25% said that the result obtained by our method is valid. As we anticipated in the previous section, HDL-C is usually known as the "good cholesterol", then many clinicians consider that having a high value is healthy, which explains why most of them invalidate our finding.

Finally, Rules 10 and 11 concerning malnutrition and low levels of white blood cells are included for validation. These are rules that our model considers that are less important to the classification, and our expected result for these cases is that most experts choose the NA or False option. Looking at the results for these rules, we can verify our expected results. These are the rules with more NA choices, and in both cases, the number of False answers is higher than True option.

Chapter 7

Conclusion

In this work, we presented a new rule-based interpretable classification model using the Dempster-Shafer Theory and Gradient-Descent techniques aiming to solve tabular classification tasks. We showed that the proposed model has many interesting features that current solutions do not have. The most important feature is the interpretability. The model can give explanations for the decisions made after training, and we also propose a new representation for interpretable results that allow us to know which rules are essential for the classification.

We validate our hypothesis concerning that our model solves classification problems with an accuracy similar to classical classification methods. The proposed model proved to be a valid classifier; it was able to predict outcomes by adjusting rules from data in both controlled and traditional datasets. The results obtained in controlled scenarios were as expected, and in most cases, the model performed perfect classifications. Comparing our model to other classification methods such as KNN and SVM, we reported that our model could achieve accuracy similar to those methods. In all cases, the accuracy was not lower than 10% of the best model for all the datasets.

We also tested our model in a real case problem of stroke prediction. In this case, our model outperforms all the other classification methods in accuracy and area under the ROC curve. Moreover, our model outperforms several of the proposed solutions for the same problem.

Our hypothesis also considers the interpretability of the results obtained by our model, stating that the model can provide meaningful explanations for the prediction made. For the case of the stroke prediction problem, we showed three different methods to compare our interpretability results. In all of the tests performed, we were able to validate most of our interpretability results with the expected results. The most remarkable test was the expert survey, where we asked medical experts whether they consider our results were valid. The answers of these experts, along with the literature review, help us to validate almost all rules.

From the experiments executed and the results obtained, we can conclude that our hypothesis was verified.

7.1 Future Work

Although the model was proven to be able to classify correctly in most of the cases, several times, mostly with the traditional datasets, the model cannot achieve accuracy levels as high as other models such as an SVM with an RBF kernel.

As a final section, we will discuss the drawbacks of our model still has and how they can be improved as future works. Also, we will propose other scenarios and variations of our model where it can be applied.

A very early assumption that we made in our model is that we do not use the uncertainty of multi-class set except for the complete set (e.g., in a three-class problem A, B, C, the mass for the subset $\{A, B\}$ is omitted). We decided that because using all subsets involves exponential-length computations. However, we may lose valuable information due to the omission of these subsets. To deal with exponential complexity, several work proposed approximation algorithms for these computations that could be explored [40].

One of the most significant drawbacks of the current model is that data must be discretized. This process makes the model loose accuracy, resulting in worse predictions because the model cannot act differently for feature vectors that satisfy the same rules even if they are different. This disadvantage can be seen in the experiments using traditional datasets, whose scores obtained were acceptable, but lower than some other classifiers. Another significant addition is to get rid of the discretization of attributes by having a score that indicates the degree of belonging to a specific rule. This score can be computed using the attribute value without any discretization.

To retain the achieved interpretability for this new representation of rules, the functions that indicate the degree of belonging must be simple to understand. For example, we can use a sigmoid function to suggest that higher values of an attribute relate to a higher validity of the rule. Then, if the interpretability result says that this rule is important to the prediction of a class, we can interpret that higher values of this attribute as a support for the prediction of that class. Similarly, we can use a radial basis function for rules that activate closes to a specific value.

As another future work, the model can be extended to handle multimedia data. Unlike tabular data, multimedia data has many more attributes, and they relate each other forming meaningful structures. For example, in an image of a face, a group of pixels may define an eye. As another example, in an audio, the presence of cyclic low frequencies may define a genre. An alternative to tackle this problem is to rebuild the proposed model so that rules can capture these high-end structures and then making inferences using this information. Interpretability must be preserved in this variation, and that is the reason why we use these meaningful structures instead of just using the attributes.

7.2 Contributions

Besides this thesis, our work has contributed to several articles listed below.

- Peñafiel, S., Baloian, N., Sanson, H., & Pino, J. A. (2020). Applying Dempster–Shafer theory for developing a flexible, accurate and interpretable classifier. Expert Systems with Applications, 148, 113262.
- Baloian, N., Pino, J. A., Peñafiel, S., Quiteros, J., Riquelme, A., & Sanson, H. (2018). Data Science in e-Health: Two Examples from Japan. Collaborative Technologies and Data Science in Smart City Applications, 89.
- Peñafiel, S., Baloian, N., Pino, J. A., Quinteros, J., Riquelme, Á., Sanson, H., & Teoh, D. (2018, February). Associating risks of getting strokes with data from health checkup records using Dempster-Shafer Theory. In 2018 20th International Conference on Advanced Communication Technology (ICACT) (pp. 239-246). IEEE.

Bibliography

[1] M. I. Jordan and T. M. Mitchell, "Machine learning: Trends, perspectives, and prospects," *Science*, vol. 349, no. 6245, pp. 255–260, 2015.

[2] A. Tharwat, "Classification assessment methods," *Applied Computing and Informatics*, 2018.

[3] R. Caruana, Y. Lou, J. Gehrke, P. Koch, M. Sturm, and N. Elhadad, "Intelligible models for healthcare: Predicting pneumonia risk and hospital 30-day readmission," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1721–1730, ACM, 2015.

[4] S. García, A. Fernández, J. Luengo, and F. Herrera, "A study of statistical techniques and performance measures for genetics-based machine learning: accuracy and interpretability," *Soft Computing*, vol. 13, no. 10, p. 959, 2009.

[5] J. Casillas, O. Cordón, F. Herrera, and L. Magdalena, "Interpretability improvements to find the balance interpretability-accuracy in fuzzy modeling: an overview," in *Interpretability issues in fuzzy modeling*, pp. 3–22, Springer, 2003.

[6] E. Štrumbelj and I. Kononenko, "Explaining prediction models and individual predictions with feature contributions," *Knowledge and information systems*, vol. 41, no. 3, pp. 647–665, 2014.

[7] G. Shafer *et al.*, *A mathematical theory of evidence*, vol. 1. Princeton university press Princeton, 1976.

[8] G. Shafer, "Dempster's rule of combination," *International Journal of Approximate Reasoning*, vol. 79, pp. 26–40, 2016.

[9] S. J. Russell and P. Norvig, *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited,, 2016.

[10] P. Domingos and M. Pazzani, "On the optimality of the simple bayesian classifier under zero-one loss," *Machine learning*, vol. 29, no. 2-3, pp. 103–130, 1997.

[11] N. Friedman, D. Geiger, and M. Goldszmidt, "Bayesian network classifiers," *Machine learning*, vol. 29, no. 2-3, pp. 131–163, 1997.

[12] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[13] N. S. Altman, "An introduction to kernel and nearest-neighbor nonparametric regression," *The American Statistician*, vol. 46, no. 3, pp. 175–185, 1992.

[14] P. A. Lachenbruch and M. Goldstein, "Discriminant analysis," *Biometrics*, pp. 69–85, 1979.

[15] R. Olshen and C. Stone, "Classification and regression trees," *Belmont, CA: The Wadsworth and Brook*, 1984.

[16] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[17] S. H. Walker and D. B. Duncan, "Estimation of the probability of an event as a function of several independent variables," *Biometrika*, vol. 54, no. 1-2, pp. 167–179, 1967.

[18] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *nature*, vol. 323, no. 6088, p. 533, 1986.

[19] N. Chater, J. B. Tenenbaum, and A. Yuille, "Probabilistic models of cognition: Conceptual foundations," 2006.

[20] H. Ishibuchi and Y. Nojima, "Analysis of interpretability-accuracy tradeoff of fuzzy systems by multiobjective fuzzy genetics-based machine learning," *International Journal of Approximate Reasoning*, vol. 44, no. 1, pp. 4–31, 2007.

[21] M. B. Gorzałczany and F. Rudziński, "Interpretable and accurate medical data classification–a multi-objective genetic-fuzzy optimization approach," *Expert Systems with Applications*, vol. 71, pp. 26–39, 2017.

[22] Y. Gal, "Uncertainty in deep learning," *University of Cambridge*, 2016.

[23] J. D. Olden and D. A. Jackson, "Illuminating the "black box": a randomization approach for understanding variable contributions in artificial neural networks," *Ecological modelling*, vol. 154, no. 1-2, pp. 135–150, 2002.

[24] M. T. Ribeiro, S. Singh, and C. Guestrin, "Model-agnostic interpretability of machine learning," *arXiv preprint arXiv:1606.05386*, 2016.

[25] A. Shrikumar, P. Greenside, and A. Kundaje, "Learning important features through propagating activation differences," *arXiv preprint arXiv:1704.02685*, 2017.

[26] A. Datta, S. Sen, and Y. Zick, "Algorithmic transparency via quantitative input influence: Theory and experiments with learning systems," in *Security and Privacy (SP), 2016 IEEE Symposium on*, pp. 598–617, IEEE, 2016.

[27] S. Bach, A. Binder, G. Montavon, F. Klauschen, K.-R. Müller, and W. Samek, "On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance prop-

agation," *PloS one*, vol. 10, no. 7, p. e0130140, 2015.

[28] S. Lipovetsky and M. Conklin, "Analysis of regression in game theory approach," *Applied Stochastic Models in Business and Industry*, vol. 17, no. 4, pp. 319–330, 2001.

[29] M. T. Ribeiro, S. Singh, and C. Guestrin, "Why should i trust you?: Explaining the predictions of any classifier," in *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 1135–1144, ACM, 2016.

[30] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in *Advances in Neural Information Processing Systems*, pp. 4765–4774, 2017.

[31] A. P. Dempster, "Upper and lower probabilities induced by a multivalued mapping," *The annals of mathematical statistics*, pp. 325–339, 1967.

[32] P. Smets, "Decision making in a context where uncertainty is represented by belief functions," in *Belief functions in business decisions*, pp. 17–61, Springer, 2002.

[33] H. M. Thoma, "Factorization of belief functions," 1990.

[34] P. Smets, "Decision making in the tbm: the necessity of the pignistic transformation," *International Journal of Approximate Reasoning*, vol. 38, no. 2, pp. 133–147, 2005.

[35] B. R. Cobb and P. P. Shenoy, "On the plausibility transformation method for translating belief function models to probability models," *International journal of approximate reasoning*, vol. 41, no. 3, pp. 314–330, 2006.

[36] F. Voorbraak, "A computationally efficient approximation of dempster-shafer theory," *International Journal of Man-Machine Studies*, vol. 30, no. 5, pp. 525–536, 1989.

[37] P. Orponen, "Dempster's rule of combination is# p-complete," *Artificial Intelligence*, vol. 44, no. 1-2, pp. 245–253, 1990.

[38] D. Dubois and H. Prade, "A set-theoretic view of belief functions logical operations and approximations by fuzzy sets," *International Journal Of General System*, vol. 12, no. 3, pp. 193–226, 1986.

[39] R. R. Yager, "On the dempster-shafer framework and new combination rules," *Information sciences*, vol. 41, no. 2, pp. 93–137, 1987.

[40] T. Reineking, *Belief functions: Theory and algorithms*. PhD thesis, Staats-und Universitätsbibliothek Bremen, 2014.

[41] L. A. Klein and L. A. Klein, *Sensor and data fusion: a tool for information assessment and decision making*, vol. 324. SPIE press Bellingham^ eWA WA, 2004.

[42] Y. Mulyani, E. F. Rahman, L. S. Riza, *et al.*, "A new approach on prediction of fever disease by using a combination of dempster shafer and naïve bayes," in *Science in Information Technology (ICSITech), 2016 2nd International Conference on*, pp. 367–

371, IEEE, 2016.

[43] T. Denoeux, "A k-nearest neighbor classification rule based on dempster-shafer theory," *IEEE transactions on systems, man, and cybernetics*, vol. 25, no. 5, pp. 804–813, 1995.

[44] T. Denoeux, "A neural network classifier based on dempster-shafer theory," *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, vol. 30, no. 2, pp. 131–150, 2000.

[45] Q. Chen, A. Whitbrook, U. Aickelin, and C. Roadknight, "Data classification using the dempster–shafer method," *Journal of Experimental & Theoretical Artificial Intelligence*, vol. 26, no. 4, pp. 493–517, 2014.

[46] D. Fixsen and R. P. Mahler, "The modified dempster-shafer approach to classification," *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, vol. 27, no. 1, pp. 96–104, 1997.

[47] S. Peñafiel, N. Baloian, J. A. Pino, J. Quinteros, Á. Riquelme, H. Sanson, and D. Teoh, "Associating risks of getting strokes with data from health checkup records using dempster-shafer theory," in *Advanced Communication Technology (ICACT), 2018 20th International Conference on*, pp. 239–246, IEEE, 2018.

[48] V. Torczon, "On the convergence of pattern search algorithms," *SIAM Journal on optimization*, vol. 7, no. 1, pp. 1–25, 1997.

[49] J. Kennedy, "Particle swarm optimization," *Encyclopedia of machine learning*, pp. 760–766, 2010.

[50] L. Davis, "Handbook of genetic algorithms," 1991.

[51] C. T. Kelley, *Iterative methods for optimization*. SIAM, 1999.

[52] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[53] T. Zhang, "Solving large scale linear prediction problems using stochastic gradient descent algorithms," in *Proceedings of the twenty-first international conference on Machine learning*, p. 116, ACM, 2004.

[54] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, no. Jul, pp. 2121–2159, 2011.

[55] T. Tieleman and G. Hinton, "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude," *COURSERA: Neural networks for machine learning*, vol. 4, no. 2, pp. 26–31, 2012.

[56] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[57] S. Linnainmaa, "Taylor expansion of the accumulated rounding error," *BIT Numerical Mathematics*, vol. 16, no. 2, pp. 146–160, 1976.

[58] A. Griewank, "Who invented the reverse mode of differentiation," *Documenta Mathematica, Extra Volume ISMP*, pp. 389–400, 2012.

[59] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind, "Automatic differentiation in machine learning: a survey," *Journal of machine learning research*, vol. 18, no. 153, 2018.

[60] N. Baloian, J. Frez, J. A. Pino, and G. Zurita, "Supporting collaborative preparation of emergency plans," in *Multidisciplinary Digital Publishing Institute Proceedings*, vol. 2, p. 1254, 2018.

[61] P.-T. De Boer, D. P. Kroese, S. Mannor, and R. Y. Rubinstein, "A tutorial on the cross-entropy method," *Annals of operations research*, vol. 134, no. 1, pp. 19–67, 2005.

[62] R. Correa and C. Lemaréchal, "Convergence of some algorithms for convex minimization," *Mathematical Programming*, vol. 62, no. 1-3, pp. 261–275, 1993.

[63] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[64] A. Asuncion and D. J. Newman, "Uci machine learning repository [http://www. ics. uci. edu/˜ mlearn/mlrepository. html]. irvine, ca: University of california," *School of Information and Computer Science*, vol. 12, 2007.

[65] L. Breiman and P. Spector, "Submodel selection and evaluation in regression. the x-random case," *International statistical review/revue internationale de Statistique*, pp. 291–319, 1992.

[66] R. Fisher and M. Marshall, "Iris data set," *RA Fisher, UC Irvine Machine Learning Repository*, vol. 440, 1936.

[67] P. Cortez, A. Cerdeira, F. Almeida, T. Matos, and J. Reis, "Modeling wine preferences by data mining from physicochemical properties," *Decision Support Systems*, vol. 47, no. 4, pp. 547–553, 2009.

[68] R. Detrano, A. Janosi, W. Steinbrunn, M. Pfisterer, J.-J. Schmid, S. Sandhu, K. H. Guppy, S. Lee, and V. Froelicher, "International application of a new probability algorithm for the diagnosis of coronary artery disease," *The American journal of cardiology*, vol. 64, no. 5, pp. 304–310, 1989.

[69] W. H. Wolberg and O. L. Mangasarian, "Multisurface method of pattern separation for medical diagnosis applied to breast cytology.," *Proceedings of the national academy of sciences*, vol. 87, no. 23, pp. 9193–9196, 1990.

[70] T. E. Doyle, J. B. Goodrich, B. J. Ambrose, H. Patel, S. Kwon, and L. H. Pearson, "Ultrasonic differentiation of normal versus malignant breast epithelial cells in monolayer cultures," *The Journal of the Acoustical Society of America*, vol. 128, no. 5, pp. EL229–EL235, 2010.

[71] L. Xu, A. Krzyzak, and C. Y. Suen, "Methods of combining multiple classifiers and their applications to handwriting recognition," *IEEE transactions on systems, man, and cybernetics*, vol. 22, no. 3, pp. 418–435, 1992.

[72] A. Vergara, S. Vembu, T. Ayhan, M. A. Ryan, M. L. Homer, and R. Huerta, "Chemical gas sensor drift compensation using classifier ensembles," *Sensors and Actuators B: Chemical*, vol. 166, pp. 320–329, 2012.

[73] W. H. Organization *et al.*, "2008-2013 action plan for the global strategy for the prevention and control of noncommunicable diseases: prevent and control cardiovascular diseases, cancers, chronic respiratory diseases and diabetes," 2009.

[74] E. J. Benjamin, M. J. Blaha, S. E. Chiuve, M. Cushman, S. R. Das, R. Deo, S. D. de Ferranti, J. Floyd, M. Fornage, C. Gillespie, *et al.*, "Heart disease and stroke statistics—2017 update: a report from the american heart association," *Circulation*, vol. 135, no. 10, pp. e146–e603, 2017.

[75] W. H. Organization *et al.*, "Noncommunicable diseases country profiles 2014," 2014.

[76] World Health Organization, *International Classification of Functioning, Disability and Health: ICF*. World Health Organization, 2001.

[77] D. Gallagher, S. B. Heymsfield, M. Heo, S. A. Jebb, P. R. Murgatroyd, and Y. Sakamoto, "Healthy percentage body fat ranges: an approach for developing guidelines based on body mass index," *The American journal of clinical nutrition*, vol. 72, no. 3, pp. 694–701, 2000.

[78] A. Farinde, "Lab values, normal adult: Laboratory reference ranges in healthy adults.," tech. rep., Medscape, 2019.

[79] C. Guo and F. Berkhahn, "Entity embeddings of categorical variables," *arXiv preprint arXiv:1604.06737*, 2016.

[80] B. F. Gage, A. D. Waterman, W. Shannon, M. Boechler, M. W. Rich, and M. J. Radford, "Validation of clinical classification schemes for predicting stroke: results from the national registry of atrial fibrillation," *Jama*, vol. 285, no. 22, pp. 2864–2870, 2001.

[81] B. Letham, C. Rudin, T. H. McCormick, D. Madigan, *et al.*, "Interpretable classifiers using rules and bayesian analysis: Building a better stroke prediction model," *The Annals of Applied Statistics*, vol. 9, no. 3, pp. 1350–1371, 2015.

[82] D. Teoh, "Towards stroke prediction using electronic health records," *BMC medical informatics and decision making*, vol. 18, no. 1, p. 127, 2018.

[83] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[84] J. Quinlan, "Induction of decision trees. mach. learn," 1986.

[85] J. R. Quinlan, *C4. 5: programs for machine learning*. Elsevier, 2014.

[86] L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen, *Classification and regression trees*. CRC press, 1984.

[87] J. H. Friedman, "Greedy function approximation: a gradient boosting machine," *Annals of statistics*, pp. 1189–1232, 2001.

[88] H. Jerrgensen, H. Nakayama, J. Reith, H. Raaschou, and T. Olsen, "Stroke recurrence: Predictors, severity, and prognosis. the copenhagen stroke study," *Neurology*, vol. 48, no. 4, pp. 891–895, 1997.

[89] R. L. Sacco, P. A. Wolf, W. Kannel, and P. McNamara, "Survival and recurrence following stroke. the framingham study.," *Stroke*, vol. 13, no. 3, pp. 290–295, 1982.

[90] D. Hier, M. Foulkes, M. Swiontoniowski, R. L. Sacco, P. Gorelick, J. Mohr, T. Price, and P. Wolf, "Stroke recurrence within 2 years after ischemic infarction.," *Stroke*, vol. 22, no. 2, pp. 155–161, 1991.

[91] M. Prencipe, F. Culasso, M. Rasura, A. Anzini, M. Beccia, M. Cao, F. Giubilei, and C. Fieschi, "Long-term prognosis after a minor stroke: 10-year mortality and major stroke recurrence rates in a hospital-based cohort," *Stroke*, vol. 29, no. 1, pp. 126–132, 1998.

[92] E. D'erasmo, M. Acca, F. Celi, F. Medici, T. Palmerini, and D. Pisani, "Plasma fibrinogen and platelet count in stroke.," *Journal of medicine*, vol. 24, no. 2-3, pp. 185–191, 1993.

[93] A. Rocco, P. U. Heuschmann, P. D. Schellinger, M. Köhrmann, J. Diedler, M. Sykora, C. H. Nolte, P. Ringleb, W. Hacke, and E. Jüttler, "Glycosylated hemoglobin a1 predicts risk for symptomatic hemorrhage after thrombolysis for acute stroke," *Stroke*, vol. 44, no. 8, pp. 2134–2138, 2013.

[94] R. D. Abbott, R. P. Donahue, S. W. MacMahon, D. M. Reed, and K. Yano, "Diabetes and the risk of stroke: the honolulu heart program," *Jama*, vol. 257, no. 7, pp. 949–952, 1987.

[95] S. P. Walker, E. B. Rimm, A. Ascherio, I. Kawachi, M. J. Stampfer, and W. C. Willett, "Body size and fat distribution as predictors of stroke among us men," *American Journal of Epidemiology*, vol. 144, no. 12, pp. 1143–1150, 1996.

[96] A. R. Folsom, M. L. Rasmussen, L. E. Chambless, G. Howard, L. S. Cooper, M. I. Schmidt, and G. Heiss, "Prospective associations of fasting insulin, body fat distribution, and diabetes with risk of ischemic stroke. the atherosclerosis risk in communities

(aric) study investigators.," *Diabetes care*, vol. 22, no. 7, pp. 1077–1083, 1999.

[97] A. M. Gotto and E. A. Brinton, "Assessing low levels of high-density lipoprotein cholesterol as a risk factor in coronary heart disease: a working group report and update," *Journal of the American College of Cardiology*, vol. 43, no. 5, pp. 717–724, 2004.

[98] I. Saito, K. Yamagishi, Y. Kokubo, H. Yatsuya, H. Iso, N. Sawada, M. Inoue, and S. Tsugane, "Association of high-density lipoprotein cholesterol concentration with different types of stroke and coronary heart disease: The japan public health center-based prospective (jphc) study," *Atherosclerosis*, vol. 265, pp. 147–154, 2017.

[99] G. Hu, J. Tuomilehto, K. Silventoinen, C. Sarti, S. Männistö, and P. Jousilahti, "Body mass index, waist circumference, and waist-hip ratio on the risk of total and type-specific stroke," *Archives of internal medicine*, vol. 167, no. 13, pp. 1420–1427, 2007.

[100] T. B. Harris, M. Visser, J. Everhart, J. Cauley, F. Tylavsky, T. Fuerst, M. Zamboni, D. Taaffe, H. E. Resnick, A. Scherzinger, *et al.*, "Waist circumference and sagittal diameter reflect total body fat better than visceral fat in older men and women: the health, aging and body composition study," *Annals of the New York Academy of Sciences*, vol. 904, no. 1, pp. 462–473, 2000.

Appendix

Expert survey

Referenced in section 6.4.

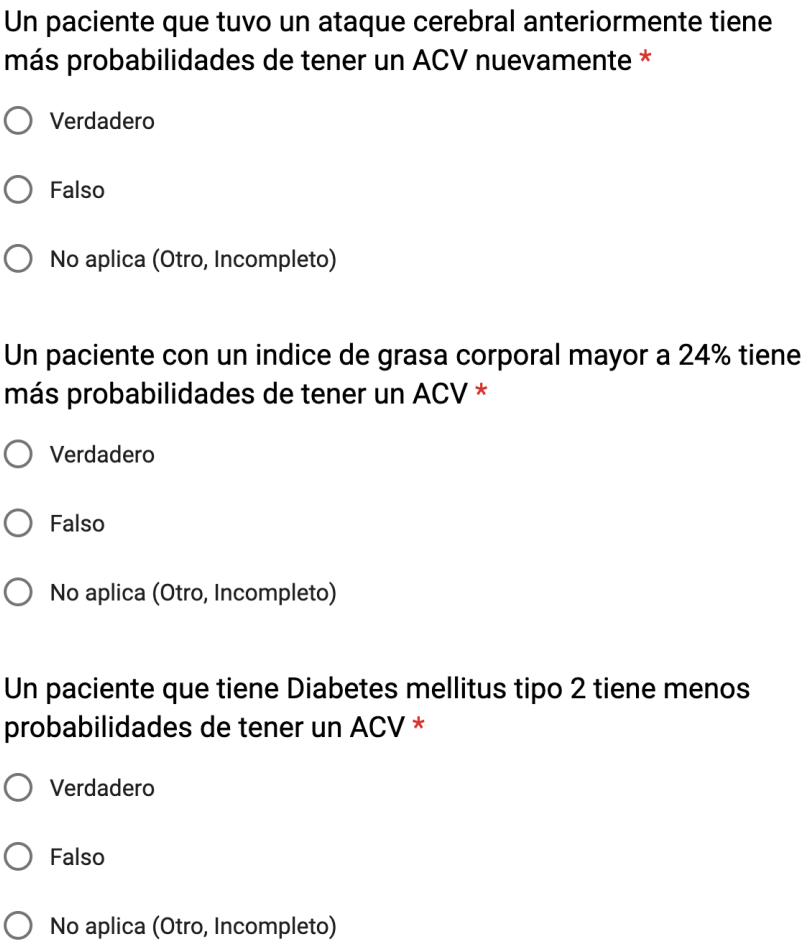


Figure 7.1: Expert survey about the rules of the model.

Un paciente que tiene un porcentaje de hemoglobina glicosilada (HbA1c) mayor al 13% tiene más probabilidades de tener un ACV *

- ◯ Verdadero
- ◯ Falso
- ◯ No aplica (Otro, Incompleto)

Un paciente que tiene un nivel de colesterol de alta densidad (HDL-C) mayor a 60 mg/dL tiene más probabilidades de tener un ACV *

- ◯ Verdadero
- ◯ Falso
- ◯ No aplica (Otro, Incompleto)

Un paciente que tiene un recuento de glóbulos blancos (WBC) menor a 4000 por microlitro tiene menos probabilidades de tener un ACV *

- ◯ Verdadero
- ◯ Falso
- ◯ No aplica (Otro, Incompleto)

Un paciente con desnutrición (IMC menor a 18) tiene menos probabilidades de tener un ACV *

- ◯ Verdadero
- ◯ Falso
- ◯ No aplica (Otro, Incompleto)

Figure 7.2: Expert survey about the rules of the model. (cont.)

Expert survey results

Referenced in section 6.4.

No.	Date	R1	R3	R4	R5	R6	R10	R11
1	2019-10-07	Verdadero	Verdadero	Falso	Verdadero	Verdadero	Falso	No aplica
2	2019-12-16	Verdadero	No aplica	Falso	Verdadero	Verdadero	No aplica	No aplica
3	2019-12-19	Verdadero	Verdadero	Falso	Verdadero	Falso	Verdadero	Falso
4	2019-12-23	Verdadero	Verdadero	Falso	Verdadero	Falso	No aplica	Falso
5	2019-12-23	Verdadero	Verdadero	Falso	Verdadero	Falso	No aplica	Falso
6	2020-01-02	Verdadero	Verdadero	Falso	Verdadero	Falso	No aplica	No aplica
7	2020-01-02	Verdadero	Verdadero	Falso	Verdadero	Verdadero	No aplica	Verdadero
8	2020-01-02	Verdadero	Verdadero	Falso	Verdadero	Verdadero	Falso	Falso
9	2020-01-02	Verdadero	Verdadero	Falso	Verdadero	Falso	No aplica	Verdadero
10	2020-01-02	Verdadero	Verdadero	Falso	Verdadero	Verdadero	No aplica	Falso
11	2020-01-02	Verdadero	No aplica	Falso	Verdadero	No aplica	Falso	Falso
12	2020-01-02	Verdadero	Verdadero	Falso	Verdadero	Falso	Falso	Falso
13	2020-01-02	Verdadero	Verdadero	Falso	Verdadero	Falso	No aplica	No aplica
14	2020-01-03	Verdadero	Verdadero	Falso	Verdadero	Falso	No aplica	Falso
15	2020-01-03	Verdadero	Verdadero	Falso	Verdadero	Falso	Falso	Falso
16	2020-01-04	Verdadero	Verdadero	Falso	No aplica	Falso	No aplica	Falso

Table 7.1: Expert Survey Raw Results