POLITECNICO DI TORINO

Department of Control and Computer Engineering

Master of Science in Computer Engineering

Master Thesis

Techniques for trustworthy artificial intelligence systems in the context of a loan approval process

Advisor
prof. Elena Baralis

Candidate
Flavio Lorenzo

December 2019

Acknowledgements

This Master Thesis marks the end of my formal education journey, as well as the beginning of my professional life. I would like to thank all the people that made this accomplishment possible.

First of all, i would like to thank my advisor, professor Elena Baralis.

A special thank goes to Blue Reply for allowing me to work on such an interesting project. In particular, i would like to thank my tutor, Oscar Pistamiglio, as well as Erasmo Purificato, for all the support that you gave to me. An acknowledgement goes also to Edoardo Ghignone, for helping me while scrambling through dozens of articles in the attempt to build a decent product.

I would like to thank my parents for allowing me to pursue this degree. It really means a lot to me.

Finally, i would like to thank my girlfriend, Cinzia, as well as all my relatives and friends for believing in me and supporting me through countless hours of studying.

Summary

On April 2019, the *High-Level Expert Group on AI*, appointed by the European Commission, presented the document “Ethics Guidelines for Trustworthy Artificial Intelligence”. In these guidelines, the group identifies seven key requirements that AI systems should meet in order to be considered trustworthy. To meet these requirements, companies need to apply a combination of both organizational and technical adjustments to their AI systems.

This work focuses on two key technical aspects of a trustworthy AI: interpretability of the underlying machine learning model and fairness in the decisions taken by the system. Model interpretability can be defined as the degree to which a human can understand the cause of a decision, while machine learning fairness refers to the property of an AI system to not base his decisions on sensitive attributes, like gender or skin colour. These concepts are extensively analysed in the first part of the thesis, and a selection of algorithms, frameworks, and tools available for supporting the processes of comprehending an AI system’s behaviour and detecting biased decisions is presented. Several solutions that can be adopted to mitigate the biases embedded in an AI system are also discussed.

In the second part of this work, the topics of interpretability and fairness are applied to the use case of a loan approval process. The algorithms and frameworks presented in the first part of the thesis are exploited to build an internet-based application that allows the user to manage the whole life cycle of a machine learning model, provide an interpretation of the model’s output, and monitor the model’s decisions to detect and react to unfair behaviours. An overview of the architecture and interface of this model management application is presented, and the most interesting components of the application are discussed in detail and compared to existing solutions.

Contents

Chapter 1

Introduction

Over the last decades, the rise of information technology has led to an increase in the amount of data that is being generated by people all over the world. According to the report "DataAge 2025" by IDC [34], the quantity of digital data available in 2018 amounted to 33 zettabyte, and this number is expected to grow up to 175 zettabyte by 2025. The increasing availability of data introduces several new possibilities for companies to extract valuable information capable of improving their decision making processes. However, for this scenario to happen, data analysis techniques must evolve accordingly to keep up with the speed to which new data is being generated, and to overcome the human limit when it comes to processing new information. The need to process increasingly large quantities of data with the goal of extracting valuable insights was one of the main reasons that led to the development and adoption of machine learning techniques. Through these strategies, an artificial intelligence (AI) system is able to automatically learn new behaviours based on previous experience without being explicitly programmed. Machine learning algorithms are being adopted in a wide variety of fields such as medicine, finance, and fraud detection.

As machine learning techniques, and the AI systems based on them, are becoming ubiquitous, concerns are starting to arise whether the development of these pieces of software, and the decisions made by them, should be based on a set of ethical principles. Indeed, artificial intelligence advances offer a unique opportunity to promote social equity, sustainability, and process optimisation. However, along with these great benefits, AI systems also threatens to empathise social injustice and provide sub-optimal, unintelligible decisions that can't be easily controverted. To minimise these risks, AI systems must be designed to be *human-centric*, that is, people must commit to use these tools "in the service of humanity and the common good, with the goal of improving human welfare and freedom" [26]. In other words, for AI systems to act on behalf of a common good, their development, deployment, and usage must be based on the principle of *trustworthiness*.

1.1 Ethics Guidelines for Trustworthy Artificial Intelligence

On April 2019, the *High-Level Expert Group on AI* (AI HLEG), appointed by the European Commission, presented the document “Ethics Guidelines for Trustworthy Artificial Intelligence” [26]. Through these guidelines, the group aims to build a horizontal foundation to promote and achieve what they refer to as “Trustworthy Artificial Intelligence”. As the guidelines’ authors note, the concept of Trustworthy AI is made of three main components: compliance with existing laws and regulations (*lawful AI*); alignment with society’s ethical principles, even in those situations in which no regulation has been developed yet (*ethical AI*); robustness both from a technical and social perspective in order to avoid incorrect behaviours that may cause unintentional harm (*robust AI*). These three components all contribute to the trustworthiness of the AI system as a whole. Nonetheless, there may be situations in which pursuing all three components at the same time is unfeasible, and a trade-off must be reached.

In the ethics guidelines, the AI HLEG group identifies four ethical principles that must be satisfied in order for an AI system to be considered trustworthy: *respect for human autonomy*, *prevention of harm* to other human beings, *fairness* of the AI system’s decisions, and *explicability* of the outcome of an AI system. These principles offer the foundation of a framework that can be used as a reference to build trustworthy AI systems. This framework, proposed by the AI HLEG group, is based on the alignment to seven key requirements that provide a practical translation of the ethical principles previously identified:

Human agency and oversight AI systems must align with the principle of *respect for human autonomy*, which means that they must act as promoters of fundamental rights, support user’s agency and allow human oversight, thus providing the users the tools to monitor and assess the way an AI system works.

Technical robustness and safety This requirement states that AI systems must be designed following a preventive approach to risk, in order to minimise the chances that the system behaves in an unpredicted and harmful way. Moreover, to satisfy technical robustness, AI systems must also provide accurate, reproducible, and reliable results.

Privacy and data governance To realise the principle of *prevention of harm*, AI systems must guarantee the privacy of their users, as well as adequate data protection measures. This requirement also highlights the importance of the quality of the data used to train the underlying machine learning model, and the risk of having a dataset which contains socially constructed biases that might influence the model’s training.

Transparency AI systems should provide clear explanations of their result, and the way they interact with the user should never be interpreted as though the decision were made by a human rather than a machine.

Diversity, non-discrimination and fairness In order to satisfy the principle of *fairness*, the user must be aware of existing biases that may lead the AI system to discriminate certain groups or individuals. Furthermore, AI systems should be accessible to people of any age, gender, and abilities.

Societal and environmental wellbeing AI systems should be designed in a way that addresses the environmental concerns and enhances social skills.

Accountability The algorithms that govern a company's AI system, as well as the data and the design processes from which this system stems, should be auditable by users internal or external to the organisation. The system should also offer the capability to monitor its actions and decisions, and should alert the user when some behaviour might negatively influence the final outcome.

These requirements affect the whole life cycle of an AI system and influence the several stakeholders that interacts with the system. This includes the developers that design the system, the deployers that exploit the AI system in their business processes, and the end users that are affected by the system's decisions. The present work addresses the problem of meeting the afore mentioned requirements from a technical perspective, and focuses on the technical factors that support the development of an AI system that satisfies the principles of *fairness* and *explicability*.

1.2 Context and scope of this work

This work is the result of a five-months full-time internship at Blue Reply s.r.l., a branch of the Reply holding which specialises on IBM technology. While the original project concerned the application of machine learning techniques to support a loan approval process, the growing need for clear explanations and the subsequent release of the Ethics Guidelines provided the perfect testing ground for evaluating the most popular solutions for model interpretability and machine learning fairness. The scope of this work is thus to provide an in depth analysis of the most recent algorithms and tools capable of providing the foundation to develop an AI system which satisfies the *fairness* and *explicability* principles identified by the AI HLEG. A selection of these solutions is then applied to the afore mentioned loan approval use case, with the goal of developing a model management tool capable of supporting the user throughout the life cycle of a machine learning model. The resulting internet-based application also provides the possibility to monitor the model's behaviour, obtain an explanation for its outcome, and asses the fairness of the model's decisions.

The work is structured as follows. Chapter 2 provides an overview of the interpretability problem. It presents the most commonly known solutions to assess a model's behaviour, and describes some more sophisticated techniques capable of providing an explanation independently of the machine learning algorithm used to train the model. It concludes by presenting some frameworks and tools that provide an easier access to the proposed strategies. Chapter 3 analyses instead the *fairness* concept, its different definitions, and the algorithms that allow to mitigate the biases that might be embedded in an AI system. Some frameworks and tools that support the end user or developer during the fairness monitoring process are also presented. In Chapter 4 the techniques previously analysed are applied to the loan approval process, and the resulting application's structure and interface are presented by focusing on the most value adding features. This tool is then compared to similar solutions analysed in the previous chapters. Finally, Chapter 5 draws the work's conclusions and describes some improvements that might be introduced in future versions of the presented model management application.

Chapter 2

Interpretability

One of the core features required for artificial intelligence systems to be trustworthy is their capability to provide the user with an explanation for why a specific prediction was made. This property is crucial for building trust in the decisions taken by a model, and is one of the key factors that influence the adoption of machine learning techniques inside high risk fields. This chapter analyses the interpretability problem by first presenting the reasons why machine learning models should provide explanations for their outputs. In the second section, the most commonly used solutions to gather insights on a model's predictions are discussed. The third chapter presents some of the most widely adopted model agnostic algorithms for model interpretability. Finally, the fourth section describes a variety of tools and frameworks available for supporting the process of understanding how a machine learning model works and for interpreting its results.

2.1 The need for interpretability

In the last decades a wide array of machine learning algorithms has been proposed to help extracting insights and making predictions from large collections of data. As the amount of data being collected grew and these algorithms became faster and more accurate, their applications have extended to several different fields. Machine learning models are used to predict stock prices, to identify the presence of cancerous tumors or, as in the use case presented in this work, to decide whether to grant a loan to bank customers. Given the specific field of application, the risk associated to the prediction being computed may vary greatly. This is why it is crucial to be able to understand the reasons why a prediction was made. There are situations in which having the capabilities to interpret a result and to understand the factors that contributed to it is far more important than having the model with the highest possible accuracy. This is one of the reasons why simpler algorithms, such as decision trees and logistic regression, are widely used despite being less accurate then other options like neural networks and support vector machines.

Linked to the interpretability of a model is the concept of fairness. This concept is discussed in more detail in Chapter 3, but it's still worth noting how the ability to understand which factors contributed the most to a prediction helps the process of detecting biases embedded in the trained model. As an example, let's consider the situation in which a black person requests a loan. The model may be biased towards white people, hence the loan may not be granted to a customer based on his skin colour alone. The capability to understand how a prediction was made can help exposing the discriminatory behaviour of the machine leaning model, thus allowing to recognise and fix the bias.

Another reason why interpretability is needed is that it provides a strategy to debug previously trained model, that is, to discover behaviours that lead to incorrect results. To provide an example of the way interpretability techniques can be used to understand how a model works and debug incorrect behaviours, a real use case is presented next. As part of an application designed to support the management of a power plant, there was the need to train a model to recognise, based on the photo of a special type of switch, whether the switch was turned on. Though the model had a really high accuracy independently on the position in which the photo was taken and the lightning conditions, it was decided to use a model agnostic interpretability algorithm to gain further insights on which parts of the image contributed the most to the prediction. For this use case, the specific algorithm chosen to perform the task was LIME, a technique described in more detail in Section 2.3. Figure 2.1 shows the result of this test. A mask was applied to the photo to highlight which parts of the image contributed positively to the prediction, and which parts had a negative influence. As expected, the interpretability algorithm was able to correctly identify the handle. However, the algorithm highlighted how the overlying plate also influenced the final decision. This insight was essential to discover that, among the photos used as the training and test set, the ones in which the switch was turned on had a plate of similar size, while the photos where the switch was turned off had a smaller plate. Therefore, this pattern was learnt by the model and used to predict the switch's status in new photos. This example shows how an interpretability algorithm allowed to both gain further understanding on how the model behaved, and with that to uncover an incorrect behaviour of the model.

The ability to explain why a prediction was made, and the possibility to debug a model and uncover discriminatory behaviours, all contribute to the trustworthiness of a machine learning model.

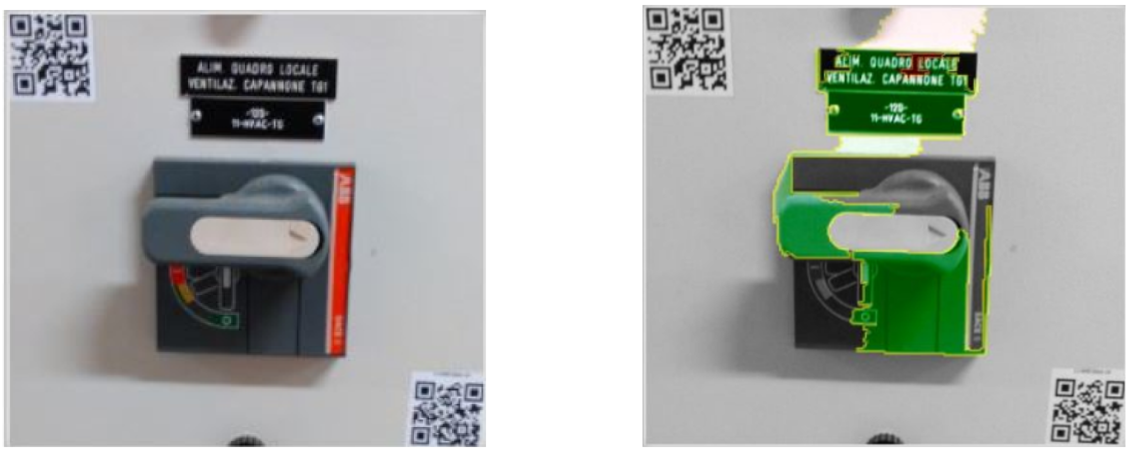


Figure 2.1: On the left, the original photo with the switch turned on. On the right, a mask has been applied to the original photo to highlight which parts of the image have a greater contribution on the model decision. The parts that positively influence the decision are shown in green, while the portions of the image that have a negative contribution are shown in red

2.2 Traditional solutions

The problem of understanding how a model works and measuring its performance has been around for a long time. Since the birth of the first machine learning algorithms, different techniques have been introduced to improve the confidence on the trained model. This section describes the most popular strategies for evaluating the goodness of a model and its predictions.

2.2.1 Interpretable models

Different machine learning algorithms are not only characterized by different performances but, depending on the algorithm used, they may also provide insights on why a specific decision was taken. In many high risk situations, the possibility to understand the inner workings of a model may be as important as the performance of the model. This section lists some of the most popular machine learning algorithms capable of training a model that can be easily interpreted.

Decision tree algorithms

Decision trees are among the most widely used classification algorithms due to their fast performances and their ease of representation. The goal of this family of algorithms is to build a tree-like graph, where each internal node, also called decision node, represents a condition on the dataset features, and the leaf nodes are the final outputs of the model. Starting from the root node, its condition is applied to the instance to predict and, based on the result of such a test, one of the available branches is traversed. Then, the same steps are repeated until a leaf node is reached.

An example of decision tree, computed from the Kaggle Titanic dataset [1], is presented in Figure 2.2. As shown by the image, it is very easy to visualize the internal structure of the model, as well as to understand the process that led to a particular decision.

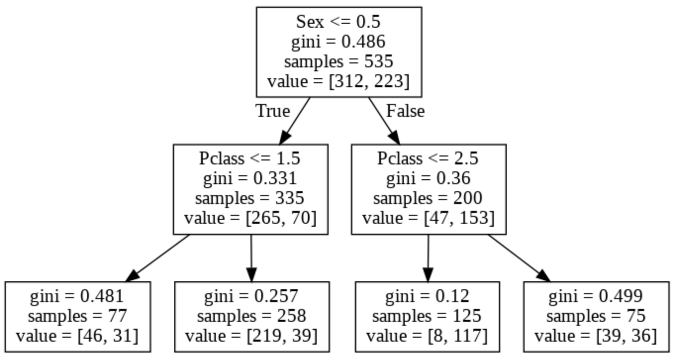


Figure 2.2: An example of a simple decision tree, built using the Titanic dataset

There are several strategies that can be followed to build decision trees. The most used ones include CART [15] and C4.5 [33], which follow a top-down greedy approach. Starting from the entire population, the algorithm selects the best condition to be used to split the data into two or more groups. Such condition varies depending on the type of attribute based on which the population will be split (i.e. nominal, ordinal or continuous), as well as the number of outgoing edges. The attribute on which the split will be applied is chosen based on the attribute that has a more homogeneous class distribution. There are several metrics available to measure the impurity of a node. The most popular ones are the Gini Index, the entropy and the classification error rate. The choice on which metric to use depends on the specific algorithm used to build the tree. The other nodes are then recursively created following the steps described above. The process continues until no split is possible (either because the records in the identified region have similar attribute values or because all the instances belong to the same class), or when a stopping condition is reached. The most common stopping conditions are when the tree has reached its maximum depth, when a split does not improve the impurity measure, and when the number of instances in the current region is below a user-defined threshold. In this situation, the class with the highest cardinality in that group is selected to characterize the leaf node.

The advantages associated with decision trees are that they are inexpensive to generate and are fast at classifying unknown records. The trained model can be fully visualized and its representation can be easily understood even by non-experts. It is also possible to grasp the motivations behind a single prediction by following the sequence of decisions taken by the model. The same procedure can be used to determine which are the attributes that are considered more relevant. However, as the tree grows in size, the model becomes harder to understand. For instance, Figure 2.3 depicts a model obtained from the same Titanic dataset [1] previously used, but without limiting the maximum depth of the tree. The size of the new decision tree prevents the trained model from being easily interpreted. Moreover, there is the risk of having an excessive data fragmentation due to the population being split across multiple nodes, which may lead to nodes in which the number of instances does not allow to make any statistically significant decision. Finally, there are several algorithms that outperform decision trees in terms of accuracy.

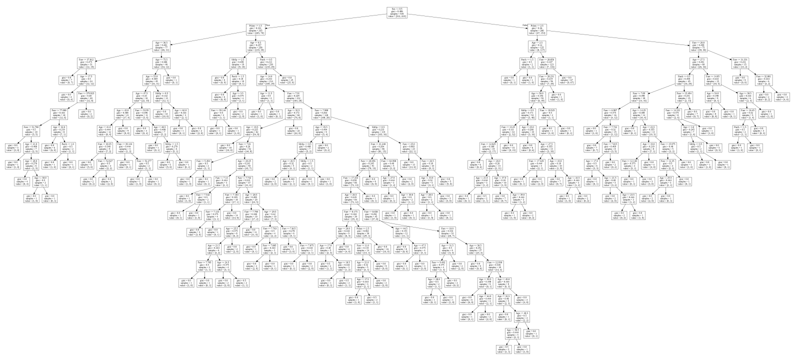


Figure 2.3: A decision tree built from the same Titanic dataset as before, without limiting the maximum depth of the tree

Nonetheless, decision trees are among the most widely used algorithms due to the advantages described above. They also provide the basis for more complex algorithms such as Random Forests and Gradient Boosting, which provide higher accuracy, but are harder to interpret.

Rule-based classification

Rule-based classifiers refers to classifiers that make use of IF-THEN rules for class prediction. A rule is usually made of two parts: the condition that must be satisfied by the instance at hand (also called antecedent) and the prediction. A prediction can also be the result of several condition being applied on the instance attributes. When a record can be described by a rule antecedent, then the instance is said to be covered by the rule.

Each rule can be evaluated by means of two main metrics: *support* and *accuracy*. The support of a rule refers to the percentage of instances to which the condition of a rule applies, while accuracy is a measure of how accurate the rule is at predicting the correct class for the instances to which the condition of the rule applies.

In order for each instance to be correctly classified, rules must have two properties: they must be both *mutually exclusive* and *exhaustive*. Mutual exclusion means that two rules can't be true at the same time, while exhaustiveness refers to the fact that each instance must be satisfied by at least one rule. By following these requirement, rules may become longer, more complex, and less readable. To overcome these limitations, the afore mentioned constraints need to be relaxed, which leads to the loss of the previously described properties. The mutual exclusion issue can be overcome either by organizing the rules in an ordered list, or by creating an unordered rule set where some rules have a higher voting power and there is a strategy to resolve potential conflicts. Instead, the loss of exhaustiveness can be solved by using a default fallback class when no rules apply.

Another key element of rule-based classifiers is the rule induction algorithm, that is, the strategy through which new rules are generated. Rules can be derived indirectly from an existing classification model, such a decision tree, or they can be extracted directly from the data by means of algorithms such as OneR, Sequential covering, or Bayesian Rule Lists.

Rule-based classifiers have the advantage of being very fast to generate. Predicting the label of an instance based on a rule is also very fast. Rules are also as expressive as decision trees, while being more compact and more readable. Rule-based classifiers also share with decision trees many of their disadvantages. They are often not as performing as other, more complex algorithms, and they also perform badly when the features being evaluated are not categorical.

Linear regression

Linear regression is a strategy to predict a continuous target value as a weighted sum of the feature inputs. Using this strategy it is possible to model the dependence of a target variable on a set of features as a linear relationship. The modelled relationship takes the following form:

$$y = \beta_0 + \beta_1 x_1 + ... + \beta_n x_n + \epsilon \tag{2.1}$$

where y represents the target variable and $x_1, ..., x_n$ are the n instance's features. $\beta_1, ..., \beta_n$ are the weights of the model, β is the intercept term, and ϵ is the error variable. Weights can be computed using several methods, such as the Ordinary Least Squares and Gradient Descent.

The main advantage of linear regression is that, once the model has been trained, estimating the target value for an unknown record is straightforward, since the only action required is to substitute the feature and weight values in Equation 2.1.

Furthermore, linear regression models can be easily interpreted by extracting the computed weights. Models trained with linear regression also come with several drawbacks. They are only suitable to model linear relationships, hence they have a tendency to oversimplify the reality, and they are sensible to outliers. Moreover, linear regression is not suitable for classification tasks, for which logistic regression should be used.

Logistic regression

Unlike linear regression, logistic regression is used to model the probabilities for binary classification problems, that is, problems in which the outcome can only take on two possible values. The main difference between the two techniques is that in logistic regression the function used to compute the probabilities is the logistic function.

$$P(y = 1) = \frac{1}{1 + exp(-(\beta_0 + \beta_1 x_1 + ... + \beta_n x_n))} \tag{2.2}$$

Logistic regression also shares many of its advantages and disadvantages with linear regression. In addition to performing classification tasks, the main output of logistic regression is a probability, which can provide additional information to the person who requested the prediction. On a different note, due to the logistic function used to compute the probability, the interpretation of the weights is more difficult with respect to the ones obtained with linear regression.

2.2.2 Model evaluation metrics

The most common way to evaluate the performance of a model, make a comparison across different models, and build confidence in the goodness of the obtained results is through model evaluation metrics. This section describes the main metrics available and the techniques used to estimate them.

Estimation methods

Metrics are estimated by first splitting the data set into two different sets: one used for training the model and a test set used to evaluate the performance of the model.

The simplest technique to perform this partitioning is called *holdout*. With this strategy, the data set is split into two disjoint groups of $p * N$ and $(1 - p) * N$ instances. Typically, p is equal to 1/2 or 2/3. The first partition obtained is the training set, while the second is the test set. This procedure can be iterated in order to compute the average performance of the model over different partitions. Holdout is especially useful when dealing with very large data sets, for which more accurate techniques might result in a longer computation time.

A finer grained strategy is *k-fold cross validation* [39]. Through this technique, the data set is split into n partitions (folds) of the same size. Then, the first $n-1$ folds are used to train the model and the remaining one is used as the test set. This process is then iterated for all partitions in order to obtain a reliable estimate of the performance of the model. With *k-fold*, there is a risk that folds might be unrepresentative of the data set as a whole. This issue can be solved by adopting a *stratified cross validation* approach instead. Through this variant of the *k-fold* technique, the data set is split into folds in which the original distribution of the instances is preserved.

Another technique frequently used is *bootstrap* [20]. In bootstrap, each training set is formed by randomly drawing instances from the initial dataset with replacement. The trained model is then tested on the remaining instances, and an estimate θ_i is computed. The process is repeated b times and a sampling distribution is built from the obtained estimates. This distribution is then exploited to make further inferences.

Accuracy, confusion matrix and ROC curve

Once the initial data set has been split into different partitions and the model has been trained on the training set, it's time to evaluate the performance of the model on the test set. This is achieved by computing a set of metrics based on the comparison between the predictions made by the model and the actual label values associated with the test set instances.

Confusion Matrix A confusion matrix is a table used to describe the performance of a classification model on the test set. Each row of the matrix represents the instances in a predicted class, while each column lists the instances in an actual class. A confusion matrix for a binary classifier can be represented like in Figure 2.4.

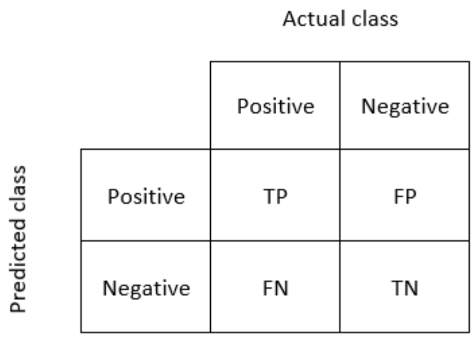
In a binary confusion matrix, the quadrants split the entire test set in four groups:

True Positives The instances in which the predicted and actual classes are both positive

False Positives The instances in which the predicted class is true, but the actual class is false

False Negatives The instances in which the predicted class is false, but the actual class is true

True Negative The instances in which the predicted and actual classes are both negative



		Actual class	
		Positive	Negative
Predicted class	Positive	TP	FP
	Negative	FN	TN

Figure 2.4: A binary confusion matrix

Accuracy Accuracy is perhaps the simplest technique to evaluate the performance of a model. It is defined as the ratio of the number of correct predictions to the total number of input samples. For binary classifiers, it can be computed as:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.3}$$

While being widely used, it is not always the best metrics to measure the performance of a model, especially when dealing with imbalanced data sets. In such situations, the interest is usually in identifying the instances with the minority class. Hence, if the model get most of the minority class instances wrong, but correctly predicts the label for the records belonging to the majority class, it will result in a high accuracy, even though the model is not able to correctly identify instances belonging to the minority class. Moreover, accuracy is not suitable when dealing with situations in which different classes have different relevance. To solve this issue other measures like recall and precision can be used.

Recall It can be described as the ratio of the number of true positives to the total number of instances with the positive class. For binary classifiers:

$$Recall = \frac{TP}{TP + FN} \tag{2.4}$$

Precision Measures the ratio of the number of true positives to the total number of instances that were predicted positive. For binary classifiers:

$$Precision = \frac{TP}{TP + FP} \tag{2.5}$$

F-Score When measuring the performance of a model, recall and precision can give opposite results. F-Score can be used to summarise both metrics and it is defined as the harmonic mean of recall and precision.

$$FScore = \frac{2 \cdot precision \cdot recall}{precision + recall} \tag{2.6}$$

ROC Curve The *Receiver Operating Characteristic* curve is a plot used to represent the performance of a classification model as its discrimination threshold is varied. It is created by plotting the true positive rate (TPR, also known as recall, see Equation 2.4) against the false positive rate (FPR, see Equation 2.7) at various threshold settings. An example of a ROC curve is shown in Figure 2.5.

$$FPRate = \frac{FP}{FP + TN} \tag{2.7}$$

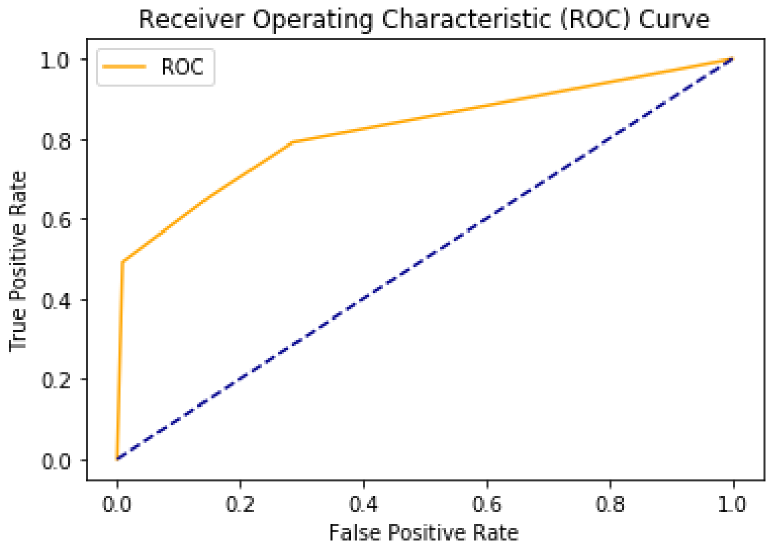


Figure 2.5: An example of a ROC curve

ROC curve can be used to compare the performance of different models, especially when dealing with imbalanced data sets. One key metric that is used for this task is the Area Under Curve (AUC), which measures the two-dimensional area underneath the ROC curve and provides a summary of the model performance across all discrimination thresholds.

2.2.3 Data visualisation techniques

Data visualisation techniques can be exploited to gain greater understanding of the data available, discover insights and hidden patterns, and understand the choices of a machine learning algorithm. As other strategies described in this chapter, this is not a new concept. Data visualisation is used on a wide variety of fields due to its effectiveness in conveying information about the data at hand. An example of process that heavily relies on graphical representation is the *exploratory data analysis* (EDA), which is one of the most important steps when approaching a new dataset to understand how the data is structured and to start recognising hidden patterns and outliers. To explore the data available, EDA makes large use of graphical data representations such as box plots, histograms, and scatter plots.

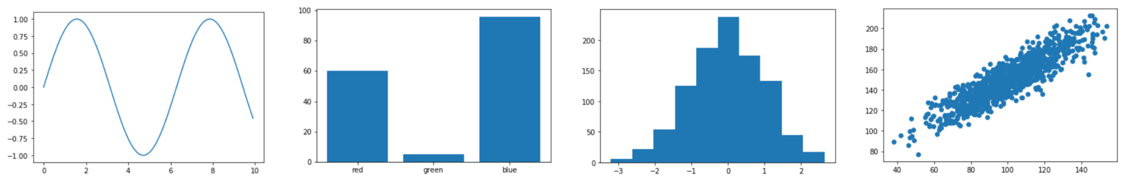
There are several plots that can be used to analyse a given dataset. The most important ones are listed below:

Line chart This plot is used to display the information related to a sequence of data points collected at a regular interval. The points are then connected using a line segment to better display the trend. (Figure 2.6a)

Bar chart The bar chart is generally used to display the relationship between a categorical (or discrete) feature and one or more numerical values. For each categorical value one or more bars are displayed. The length of each bar is defined by the numerical value associated to the selected categorical value. (Figure 2.6b)

Histogram Histograms are used to visualise the distribution of a feature. They are characterised by different bins, each of which is represented by a bar which height depends on the number of values that falls into that bin. (Figure 2.6c)

Scatter plot This plot summarizes the relationship between two features. Each point of the scatter plot represents a single observation, and its position is based on the value of the analysed attributes for that specific instance. (Figure 2.6d)



(a) Line chart (b) Bar chart (c) Histogram (d) Scatter plot

Figure 2.6: Example of different types of plots

2.3 Model-agnostic algorithms

In the previous section, the most common model evaluation strategies have been introduced. Though these techniques provide a good overview of the model performance, they do not offer any insights on why the model predicted a class instead of another. Some additional information can be extracted by providing a visual representation of the data set, but this strategy is not always the best solution. Moreover, most graphical representations can't be easily interpreted by non-expert, which is a problem when designing a solution that should encourage them to trust and critically evaluate the model decisions.

To overcome these limitations, in recent years a number of algorithms have been designed to provide insights on which features are most likely to influence the model predictions. These algorithms can be divided in two classes:

Model-specific solutions These algorithms are limited to specific model classes. They are able to provide further insights on a model prediction by exploiting the specificities of the model class of interest. An example of a model-specific interpretation is the extraction of weights from a logistic regression model. There are also solutions that deals with more complex machine learning models such as DeepLift [38], which is used to compute importance scores in deep neural networks.

Model-agnostic solutions These algorithms can be used on any machine learning model and are applied after the model has been trained. These strategies usually consider the model as a black box and they work by analysing only the feature input and the model output. They provide greater flexibility with respect to model-specific solutions, since they can be applied to a model regardless of its class.

Since the goal of this work is to build a tool to support the creation of trustworthy artificial intelligence systems regardless of the specific machine learning algorithm they're based on, in the next pages we will focus only on reviewing the most popular model-agnostic solutions.

2.3.1 LIME

LIME (*Local Interpretable Model Agnostic Explanation*) [36] is an algorithm used to access the behavior of any model (base estimator) using local interpretable surrogate models such as linear classifiers or regressors. As the authors state, the overall goal of LIME is to identify an interpretable model (e.g. decision tree, logistic regression) over the interpretable representation that is locally faithful to the classifier.

The insights derived from LIME can be used to enhance the comprehension of an otherwise black box model, hence helping experts in the process of debugging their own models and provide easy to read explanations of which features contributed the most to producing a specific output and how they influenced the final result.

The output of the algorithm can be used to generate graphs that can be easily understood by non-technical people, thus allowing to build trust in the model that was produced, identify biases embedded in the dataset and in the model, and validate people's intuitions about why a certain prediction was made.

As its name implies, LIME is model-agnostic, which means that it can work on any classification model independently of its complexity. It can be used in the field of text classification, image classification, as well as it can be applied to tabular data.

Instead of trying to fit a global surrogate model, LIME focuses on fitting a local surrogate model to explain why single predictions were made. Global surrogate models can be defined as interpretable models that have been trained to approximate the predictions of a black box model. Instead, local surrogate models do not try to approximate the whole model, but they try to learn an interpretable model only locally around the prediction of interest. This strategy overcomes the main limitation of global surrogate models, that is, the fact that they provide a simplistic view of the model, hence they are not able to fully explain the inner workings of the original model. Instead, by evaluating the model locally, Ribeiro et al. obtains a model that is *locally faithful*, i.e. it corresponds to how the model behaves in the vicinity of the instance being predicted. As the authors argue, local fidelity still does not imply global fidelity, because features that are globally important may not be important in the local context, and vice versa.

On a high level, the way *LIME* works can be described as follows: to generate its explanations, LIME tests what happens to the predictions of a black box model when it is fed with variations or perturbations of the original dataset. Typically, LIME generates a new dataset consisting of perturbed samples and the associated black box model's predictions. On this dataset LIME then trains an interpretable model weighted by the proximity of the sampled instances to the instance of interest. The model obtained through this process is then used to explain why a specific prediction was made.

As an example of what is the authors intuition for LIME, Ribeiro et al. provide a toy example that is presented in Figure 2.7. In the image, the blue/pink background represents the decision function f of the black-box model. The bold red cross is the instance being explained. The remaining circles and crosses represents the other sample instances, and their size is representative of the proximity to the instance of interest. Finally, the dashed line is the learned explanation which is locally (but not globally) faithful.

To describe how *LIME* works on a lower level, Ribeiro et al. specify the following definitions:

- $g \in G$ is used to represent the explanation of a black box model, where G is a

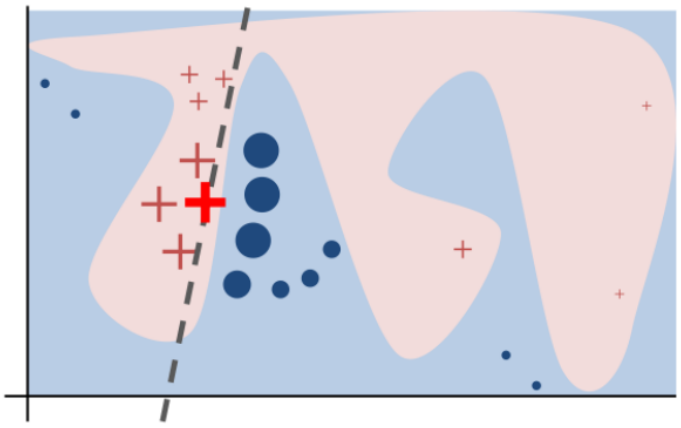


Figure 2.7: An illustration of the intuition behind LIME [36]

class of potentially interpretable model, such as linear models, decision trees, or falling rule lists. This means that the explanation can be displayed to the user by means of visual or textual artefacts. These models are referred to as *potentially interpretable* because, even though they can be easily understood by humans, this largely depends on the complexity of the explanation.

- $\Omega(g)$ is defined as a measure of complexity of the explanation $g \in G$. The higher complexity is, the harder it is for a human to interpret the model. This measure largely depends on the model being used as explanation by the algorithm. For instance, as already discussed in the section about interpretable models, for decision trees $\Omega(g)$ may be the depth of the tree.

- The model being explained is denoted as $f : \mathbb{R}^d \to \mathbb{R}$, where $f(x)$ represents the probability that x belongs to a specific class (if we consider classification algorithms).

- $\pi_x(z)$ denotes the proximity measure between an instance z to x. This is the measure used to weight the instances differently based on the vicinity to the instance being predicted, thus ensuring that the explanation produced by the algorithm is locally faithful.

- Finally, $\mathfrak{L}(f, g, \pi_x)$ is defined as the locality-aware loss and it is a measure of how unfaithful g is in approximating f. This measure is then weighted by the proximity measure $\pi_x(z)$ to define the locality of x. In the article [36], Ribeiro et al. suggest the usage of a locally weighted square loss function, as defined in Equation 2.8, where $\pi_x(z) = \exp(-D(x,z)^2/\sigma^2)$ is an exponential kernel defined using some distance function D with width σ. For instance, cosine distance can be used for text and $L2$ distance for images. Nonetheless, the explanation can be obtained by using any fidelity function $\mathfrak{L}$.

$$\mathfrak{L}(f, g, \pi_x) = \sum_{z,z' \in \mathcal{Z}} \pi_x(z)(f(z) - g(z'))^2 \tag{2.8}$$

By using the previously specified definition, it is now possible to present the formal definition of *LIME*. As already stated, the goal of *LIME* is to provide a model that is both interpretable and locally faithful to the instance being explained. To enforce these properties, the loss function $\mathfrak{L}(f, g, \pi_x)$ must be minimized while keeping $\Omega(g)$ as low as possible in order for the explanation to be easily interpreted by human. The formulation is presented in Equation 2.9.

$$\xi(x) = \operatorname*{argmin}_{g \in G} \mathfrak{L}(f, g, \pi_x) + \Omega(g) \tag{2.9}$$

2.3.2 Shapley values

The Shapley value, introduced by L.S. Shapley in 1953 [37], is a concept taken from game theory. Given a game which requires several players to cooperate, the goal of this technique is to understand how much each player contributed to the final outcome, or payout. To put it differently, it's a method for assigning payouts to players based on their contribution to the total payout. Players cooperate in a coalition and receive a certain profit from this cooperation. This means that the distribution of the total payout is not only based on how well each player performed individually, but also on the marginal contribution provided by this player when it cooperates with other players.

Let N be the set of all n players that participate in the game. $v : 2^N \to \mathbb{R}$ is a function that assign to each subset of players their total contribution to the game. The contribution is expressed as a real number. In particular, $v(\emptyset) = 0$ states that the contribution brought to the game by an empty subset of players is equal to 0. Then, the Shapley value for player i can be expressed as follows:

$$\varphi_i(v) = \sum_{S \subseteq N \setminus \{i\}} \frac{|S|!(N - |S| - 1)!}{N!} (v(S \cup \{i\}) - v(S)) \tag{2.10}$$

where the sum extends over all the possible permutations of the subset of players in which player i is not included and $v(S \cup \{i\}) - v(S)$ computes the marginal contribution that is brought by player i to subset S. The result is given by the average marginal contribution of player i to all possible permutations of subset $N \setminus \{i\}$.

The concept of Shapley values can be applied to machine learning to provide an alternative way of interpreting a prediction performed by a black-box machine learning model. This is achieved by assuming that each feature value of the instance is a player and the prediction outcome minus the average prediction for all instances is the total payout. Then we can use the Shapley values to understand how the payout (i.e. the difference between the actual prediction and the average prediction) can be equally distributed among the different players (feature values of the instance). In this situation, the Shapley value is the average marginal contribution of a feature value across all possible coalitions.

To compute the marginal contribution brought by a feature value to a subset of other feature values the most straightforward way is the following one: let S be a subset of feature values of the instance being predicted and i the feature value for which the Shapley value must be computed. The first step is to draw from the data set a new instance of the data and to substitute its feature values to the ones of the instance being predicted, with the exception of the subset of features $S \cup \{i\}$. Then, a new prediction is performed on the generated instance by using the black box model. The next step is to generate another instance, this time by keeping only the subset S and by replacing all other feature values, and to perform a second prediction on this new instance. The difference between the two predictions can be assumed to be an estimate of the marginal contribution $v(S \cup \{i\}) - v(S)$. Since this estimate depends on the values of the instance that was randomly drawn from the data set, the process is repeated and the computed contributions are averaged to better estimate the marginal contribution of i to S. The process is then repeated for all possible coalitions.

The Shapley value is the only attribution method that satisfies the properties *efficiency*, *symmetry*, *dummy* and *additivity*, which together can be considered a definition of a fair payout [32].

Efficiency The sum of all feature contributions is equal to the difference between the prediction for the instance to be explained and the average prediction.

$$\sum_{i=1}^{n} \varphi_i = \hat{f}(x) - E_X(\hat{f}(X)) \tag{2.11}$$

where $\sum_{i=1}^{n} \varphi_i$ is the sum of all the computed Shapley values, $\hat{f}(x)$ is the actual prediction, and $E_X(\hat{f}(x)$ is the average predicted value.

Simmetry If two feature values i and j equally contribute to all possible coalitions, their contribution should be the same.

Let $val(X)$ be the contribution of subset X to the total payout, if

$$val(S \cup x_i) = val(S \cup x_j), \forall S \subseteq \{x_1, ..., x_n\} \backslash \{x_i, x_j\}$$

then $\varphi_i = \varphi_j$

Dummy If a feature i does not change the predicted value regardless of which subset of features it is added to, then its Shapley value should be 0. If

$$val(S \cup x_i) = val(S), \forall S \subseteq \{x_1, ..., x_n\} \backslash \{x_i\}$$

then $\varphi_i = 0$

Additivity For a game with combined payouts $val + val^+$ the respective Shapley values are as follows: $\varphi_i + \varphi_i^+$

Let's suppose a game is made of several payouts that must be combined together. To do so, it is sufficient to average the payouts - and the associated Shapley values - to get the payout and Shapley values for the whole game.

The main difference between the Shapley values method and LIME is given by the efficiency property, which states that by using Shapley values the difference between the actual prediction and the average prediction is fairly distributed among the feature values of the instance. This is especially useful because the weights produced by LIME can be misinterpreted by humans, and the lack of a solid theory might be an obstacle to the process of building trust on the model predictions. Another advantage of Shapley values is that, by comparing a prediction to the average prediction of a subset of the available data, it is possible to provide *contrastive explanations*, that is, an explanation of why a certain prediction was preferred over another one.

On the opposite, the exact computation of the Shapley values might take a lot of computing time, which makes it unfeasible in most real applications. To solve this issue, an approximated method, such as the one proposed by Štrumbelj and Kononenko [40], might be adopted. Moreover, Shapley values are not suitable when producing sparse representations, since the algorithm requires for all features to be used. Finally, since Shapley values are estimated by replacing the feature values of the predicted instance with the values of a randomly drawn instance, it requires access to the original data.

2.3.3 SHAP

SHapley Additive exPlanation is a model-agnostic explanation framework proposed by Lundberg and Lee [30]. At its core, it works in the same way as the Shapley values method does. In addition, it extends the original idea by incorporating other approaches that allow *SHAP* to overcome some of the disadvantages that comes with the SHapley values method. The framework also introduces a new class of *additive feature importance* measures, along with a theoretical demonstration which shows that in the identified class there is a unique solution with a set of desirable properties. Through this new class, the authors seek to unify six different methods available in literature in order to extend them to provide improved computational performance and/or better consistency with human intuition.

In their work, Lundberg and Lee propose a unified view of several explanation methods, included LIME and the approximated Shapley value method proposed by Štrumbelj and Kononenko known as *Shapley sampling value* method [40]. The authors argue that each of the six identified explanation methods share the same explanation model as they all fall in the category of the *additive feature attribution* methods. To formally define this class, the following definitions must be given:

- Let f be the black-box model to be explained and g the explanation model, that is, the simplified model which is obtained as an output from explanation methods such as LIME. The interest here is not in explaining the whole model, but to provide a local explanation $f(x)$ for a given individual prediction x in the same way as LIME does.

- x' represents the simplified inputs that map to the original instance being predicted through a mapping function $x = h_x(x')$. Since h_x is specific to the instance x, the function is able to correctly map x' even though the simplified inputs contain less information than x.

The goal of local methods is to ensure that, whenever $z' \approx x'$, the explanation model is such that $g(z') \approx f(h_x(z'))$. We can now provide the definition for the identified class.

Definition 2.1. Additive feature attribution methods have an explanation model that is a linear function of binary variables:

$$g(z') = \phi_0 + \sum_{i=1}^{M} \phi_i z'_i \tag{2.12}$$

where $z' \in \{0,1\}^M$, M is the number of simplified input features, and $\phi_i \in \mathbb{R}$.

To put it differently, methods which produce an explanation model that matches Definition 2.1 associate to each simplified feature z'_i an effect ϕ_i. By summing up the effects of all feature attributions it is possible to approximate the output $f(x)$ of the original model.

As Lundberg and Lee state [30], the identified class of additive feature attribution methods has a single unique solution with the three desirable properties described below.

Local accuracy The explanation model g matches the output of the black-box model f for the simplified input x'.

$$f(x) = g(x') = \phi_0 + \sum_{i=1}^{M} \phi_i x'_i \tag{2.13}$$

The explanation model $g(x')$ matches the original model $f(x)$ when $x = h_x(x')$, where $\phi_0 = f(h_x(0))$ represents the model output when all simplified inputs are missing and set to 0. By defining $\phi_0 = E_X(\hat{f}(X))$ and by setting all x'_j to 1, Equation 2.13 can be converted to the *efficiency* property associated with Shapley values (Equation 2.11).

Missingness If a feature is missing in the original input x, then its impact on the explanation model must be null.

$$x'_i = 0 \implies \phi_i = 0 \tag{2.14}$$

Consistency If a model changes in a way that some simplified input's contribution increases or stays the same regardless of the other feature values, then that input's attribution should also increase or stay the same. To put it in a more formal way, let $f_x(z') = f(h_x(z'))$ and $z' \backslash i$ denote setting $z'_i = 0$. For any two models f and f', if

$$f'_x(z') - f'_x(z' \backslash i) \geq f_x(z') - f_x(z' \backslash i) \tag{2.15}$$

for all inputs $z' \in \{0{,}1\}^M$, then $\phi_i(f', x) \geq \phi_i(f, x)$.

Given the previously described properties, there is a single solution in the class of additive feature attribution methods which satisfies all three properties.

Theorem 2.1. *Only one possible explanation model g follows Definition 2.1 and satisfies the properties of local accuracy, missingness, and consistency:*

$$\phi_i(f, x) = \sum_{z' \subseteq x'} \frac{|z'|!(M - |z'| - 1)!}{M!} [f_x(z') - f_x(z' \backslash i)] \tag{2.16}$$

where $|z'|$ is the number of non-zero entries in z', and $z' \subseteq x'$ represents all z' vectors where the non-zero entries are a subset of the non-zero entries in x'.

Lundberg and Lee describes Shapley values as the only method which satisfies all properties of local accuracy, missingness and consistency. In this setting, the values ϕ_i are represented as Shapley values. By proposing a unified approach, the authors seek to improve the other methods (e.g. LIME and DeepLIFT), preventing them from violating the properties of local accuracy and missingness.

In their article, Lundberg and Lee introduce the concept of SHAP values, which are defined as the Shapley values of a conditional expectation function of the original model. These values represent the solution to Equation 2.16, where $f_x(z') = f(h_x(z')) = E[f(z)|z_S]$, and S is the set of non-zero indexes in z'. Through this definition, SHAP values are meant to align with the common methods for estimating Shapley values, as well as allowing for connections with other methods such as LIME and DeepLIFT.

A graphical interpretation of SHAP values is presented in Figure 2.8. Starting from a base value $E[f(z)]$, which corresponds to the value ϕ_0 in Equation 2.12 (that is, the value that would be predicted if no features of the current input were known), the diagram shows how the predicted output $f(x)$ is obtained by sequentially considering the marginal contribution provided by each feature. While the image presents a single ordering, when the model is non-linear or the input features are not independent the order in which these features are added to the expectation may impact on the effects associated with the introduction of the remaining features. Thus, SHAP values arise from averaging the ϕ_i values across all possible orderings.

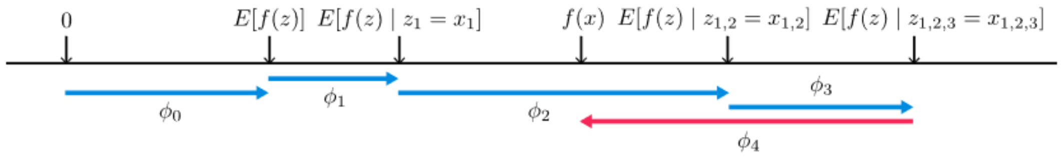


Figure 2.8: A graphical representation of the way SHAP values should be interpreted [30]

SHAP values can be computed in several ways. While the exact computation may be unfeasible for most situations (see the section dedicated to Shapley values), by combining the insights derived from the additive feature attribution methods, it is possible to compute an approximation of SHAP values. Lundberg and Lee present two model-agnostic approximation methods for computing these values:

Shapley sampling values The first algorithm is the one originally proposed by Štrumbelj and Kononenko [40]. While this approach can be successfully used when dealing with a small number of inputs, when the size of the inputs increases the Kernel SHAP method should be preferred, since it requires fewer evaluations of the original model to obtain a similar approximation accuracy.

Kernel SHAP While the original linear LIME algorithm violates the properties of local accuracy and/or consistency, the authors prove that, by properly choosing the loss function $\mathfrak{L}$, the weighting kernel π_x, and the regularization term Ω in the LIME equation (Equation 2.9) , it is possible to use that Equation 2.9 to recover the Shapley values.

Theorem 2.2. *Under Definition 2.1, the specific forms of π_x, $\mathfrak{L}$, and Ω that make solutions of Equation 2.9 consistent with the properties of local accuracy, missingness, and consistency are:*

$$\Omega(g) = 0,$$

$$\pi_{x'}(z') = \frac{(M-1)}{(M choose |z'|)|z'|(M-|z'|)},$$
$$\mathfrak{L}(f, g, \pi_{x'}) = \sum_{z' \in Z} [f(h_x(z')) - g(z')]^2 \pi_{x'}(z'),$$

where $|z'|$ is the number of non-zero entries in z'.

Using Theorem 2.2, Equation 2.9 can still be solved by using linear regression and the Shapley values can be computed using weighted linear regression. Since LIME's simplified input mapping and SHAP approximated mapping are equivalent, it is possible to perform a regression-based, model-agnostic estimation of SHAP values. The authors demonstrate that this approach provides better sample efficiency with respect to the classical Shapley equations.

Along with the introduction of Kernel SHAP, Lundberg and Lee also propose other model-specific approximation methods, which provide faster performances but are restricted to specific model types, such as linear models, deep neural networks, and tree-based algorithms. The authors also provide a library for generating global explanations about the model. This library will be further investigated in Section 2.4, which covers the available tools and frameworks for model interpretability.

Being deeply rooted in the Shapley values method, SHAP shares with the former all its advantages, namely, its theoretical foundation, the fair distribution of the outcome among the feature values, and the possibility to obtain contrastive explanations to compare the prediction being interpreted with the average prediction. In addition, SHAP tries to unify different and otherwise isolated explanation methods, most notably LIME and Shapley values. Finally, SHAP overcomes the main limitation of Shapley values, that is, the slow computation time. This allows to use SHAP values also to provide global model interpretation methods, such as feature importance, feature dependence, and summary plots.

The model-agnostic approximation method proposed by Lundberg and Lee (KernelSHAP), while more efficient than the exact computation of the Shapley values, is still quite slow, which makes its application unfeasible when the size of the data set and the number of instances for which to compute the SHAP values increase. Moreover, many of the issues of Shapley values are shared with KernelSHAP. When generating new instances to compute the marginal contribution of a feature value, the presence of correlated features increase the risk for the explanation to be based on unlikely data points. Finally, to compute SHAP values it is still required to have access to the original data. Some of these limitations can be overcome by adopting model-dependent solutions, such as TreeSHAP.

2.3.4 Anchor

Anchor is a model-agnostic interpretation system proposed by Ribeiro, Singh, and Guestrin [35], the same authors of LIME. Whereas LIME creates a local surrogate model based on the instance being explained, the goal of Anchor is to identify a set of rules that are sufficiently able to "anchor" the prediction locally. A rule anchors a prediction if, by changing the other feature values, the prediction outcome does not change. By using this approach, the resulting explanation can be expressed in the form of IF-THEN rules called *anchors*. In other words, provided that the anchor holds, the prediction for an instance is (almost) always the same. As with LIME, explanations generated by Anchor are obtained through a perturbation-based strategy.

To formally define an anchor, the following definitions must be given:

- x is the instance being explained,
- f denotes the classification model to be explained. As with LIME, the model can be used as a black box by Anchor to produce individual predictions $f(x)$,
- A represents a rule, that is, a set of predicates, such that $A(x) = 1$ when all feature predicates are true for instance x. To put it differently, if all the predicates that compose A correspond to x's feature values, then $A(x) = 1$,
- $\mathfrak{D}$ is used to indicate the perturbation distribution which is generated by many model-agnostic methods, most notably LIME and Anchor. $\mathfrak{D}(\cdot|A)$ denotes the conditional distribution when the rule A applies. It indicates the distribution of neighbours of x that match A,
- $0 \leq \tau \leq 1$ specifies the precision threshold above which a rule A is considered valid.

A is an *anchor* if the following conditions are met:

- $A(x) = 1$, and
- A is a sufficient condition for $f(x)$ with high probability. This means that the precision of anchor A must be greater or equal to τ.

Formally, we can define a rule A as an anchor if

$$\mathbb{E}_{\mathfrak{D}(z|A)}[1_{f(x)=f(z)}] \geq \tau, A(x) = 1. \tag{2.17}$$

which can be interpreted as follows: considering all the instances z in the neighbourhood of x such that z satisfies rule A (i.e. the perturbation space $\mathfrak{D}(z|A)$), if at least τ instances lead to the same result as x ($f(x) = f(z)$), then rule A is a valid anchor for x.

In the same way they did with LIME, the authors provide a visual representation of the intuition behind Anchor. Figure 2.9 compares how the Anchor and LIME algorithms locally explain a complex binary classifier using two exemplary instances, which are represented using the "+" and "-" symbols. The straight line displays the model learned by LIME, while the dashed rectangle illustrates the anchor. The circumference labelled as D delimits the perturbation distribution $\mathfrak{D}$. As the illustration shows, all LIME does is learn the linear decision boundary that best approximates the model's behaviour given a perturbation distribution, without giving further details on how faithful such approximation is (the explanation on the right is a much better local explanation of the black-box model than the approximation on the left). Instead, the explanations built by Anchor are faithful by construction, adapt their coverage to the model behaviour (the anchor on the right of Figure 2.9 is broader), and clearly express the anchors' boundaries.

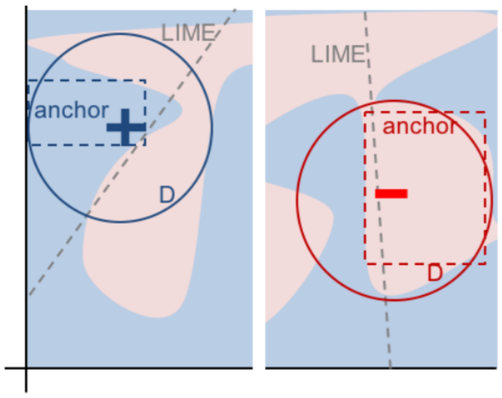


Figure 2.9: A graphical comparison between the intuitions behind Anchor and LIME. [35]

While computing an anchor A for an instance x based on a black-box model f is *theoretically* possible, in practice, given an arbitrary distribution $\mathfrak{D}$, the exact computation of the precision is unfeasible. To overcome this limitation, Ribeiro et al. introduce a new probabilistic definition to select those anchors which satisfy the precision constraint with a high probability. The probability threshold is given by the parameter $0 \leq \delta \leq 1$.

$$\mathbb{P}\{prec(A) \geq \tau\} \geq 1 - \delta \tag{2.18}$$

where $prec(A) = \mathbb{E}_{\mathfrak{D}(z|A)}[1_{f(x)=f(z)}]$. If multiple anchors satisfy this criterion, the ones that describe the behaviour of a larger part of the input space are to be preferred. This attribute of an anchor is called *coverage*, and it can be formally defined as the probability that such an anchor applies to samples from $\mathfrak{D}$.

$$cov(A) = \mathbb{E}_{\mathfrak{D}(z)}[A(z)]$$

With this definition, the search for an anchor can be defined as a combinatorial optimization problem:

$$\max_{A\, s.t.\, \mathbb{P}\{prec(A) \geq \tau\} \geq 1-\delta} cov(A) \tag{2.19}$$

To put it differently, Anchor tries to find those rules that are most likely to have a precision above threshold τ. Among these, it selects the one that describes the largest part of the model. It must be noted that, as the number of predicates in a rule grows, the precision of a rule increase at the cost of the coverage and vice versa. Hence, there is a trade-off between precision and coverage.

Since the number of all possible anchors grows exponentially, the exact solution of the problem presented in Equation 2.19 is intractable. Thus, Ribeiro et al. propose a multi-armed bandit formulation of the problem to efficiently explore the model's behaviour in the perturbation space. The approach adopted by Anchor to build an explanation for prediction $f(x)$ can be divided into multiple parts:

Candidate Generation Anchor A is constructed incrementally starting from an empty rule, which applies to every instance and thus has the highest coverage. Then, a number of candidate rules that extends A by one additional feature predicate, $\{a_i\}$, is generated. From this set, the candidate rule with the highest estimated precision is selected and used to replace A. The process is then repeated. At the end of every iteration, the algorithm checks if the current candidate rule satisfies Equation 2.18. If so, the algorithm terminates and returns the selected candidate rule. This approach leads to the fact that the algorithm favours anchors with the lowest number of feature predicates, which are usually the ones with the highest coverage.

Best Candidate Identification At the end of each candidate generation iteration, the best candidate must be selected. Since computing the exact probability as described in Equation 2.18 is not feasible, the authors propose to formulate the problem as an instance of a pure-exploration multi-armed bandit problem [29]. Using an analogy to slot machines, each candidate rule A can be seen as an arm that can be pulled, and each pull of the arm A is an evaluation of $1_{f(x)=f(z)}$ on a sample taken from the perturbation distribution $\mathfrak{D}(z|A)$. In this setting, the selection of the rule with the highest precision is performed by using the KL-LUCB algorithm [29].

Candidate Precision Validation Once the rule with the highest precision is selected by KL-LUCB, it must be evaluated if the rule can be chosen as an anchor. If there is no statistical confidence that such a rule satisfies the precision criteria (the candidate exceeds the τ threshold), then the perturbation space $\mathfrak{D}(\cdot|A)$ must be sampled once again. The process is repeated until it can be stated with confidence that candidate A is an anchor.

The previously described algorithm follows a bottom-up, greedy approach to find the anchor for a given prediction. This strategy, as noted by the authors, has two major limitations. The first issue is that, due to the greedy nature of the approach, only one rule is selected at each step of the anchor construction. Hence, any suboptimal choice performed by the algorithm is irreversible. Moreover, the greedy algorithm does not directly address the computation of the coverage, though a high coverage is to be expected since the bottom-up approach favours anchors with a lower number of feature predicates.

To address these issues, the authors propose a new approach based on a beam search algorithm which extends the greedy solution in two different ways. First, after generating all possible candidate rules, the KL-LUCB algorithm is used to select the B-best candidates, instead of a single one. Second, if the precision of the given rule is above the τ threshold and the candidate's coverage is the highest one found so far, then the candidate is saved, and its coverage is used to prune the search space in the next iterations. This optimization is based on the fact that, by adding a new predicate to an existing rule, the coverage of a rule can't increase. This way, it is more likely that the algorithm returns an anchor with a higher coverage than the one found using the greedy approach.

There are three main advantages associated with anchors. First, the fact that the algorithm's output is based on if-then rules makes the explanations easier to interpret with respect to the results obtained from the explanation methods described in the previous pages. Second, by providing information about the coverage of the anchor and its precision, the algorithm is able to return a more faithful explanation. Finally, the anchors approach is better suited to situations in which model predictions are non-linear or complex in the perturbation space of the instance to be explained.

The algorithm also presents several shortcomings. When the prediction to be explained is close to a boundary of the black-box model's decision function or belongs to a rare class, the rule computed by Anchor may be overly specific, thus too much complex and with a low coverage. In these situations, Ribeiro et al. suggest that LIME's explanation should be preferred, even though they do not provide any information about the coverage of the output. Moreover, the setup of the algorithm can become complicated, due to the huge number of hyperparameters that must be configured. For instance, the task of identifying a realistic perturbation distribution is particularly challenging, especially for some domains such as images. Finally, there may be situations in which different anchors could be applied to the same instance. While unlikely, this problem can be solved by alerting the user and increasing the precision threshold.

2.3.5 Other explanation algorithms

In the previous pages, a number of model-agnostic explanation algorithms have been introduced. Several other methods have been proposed to provide insights about why a prediction was made. These strategies were not included in the previous analysis due to a lack of documentation on the subject, or because the proposed solution does not align with the definition of model-agnostic explanation method. This section briefly lists some of these other algorithms.

DeepLIFT

Deep Learning Important FeaTures, proposed by Shrikumar et al. [38], is a model-dependent explanation method specifically designed to provide insights on the predictions computed by neural network models. It decomposes a prediction computed by a neural network by back-propagating the contributions of all neurons in the network to every feature of the input.

This approach is fundamentally different from other algorithms such as LIME or Anchor, which can be referred to as perturbation-based approaches. In the latter, an explanation is produced by observing the impact that perturbations of the original instance have on the model. This approach, the authors argue, is computationally inefficient, since each perturbation requires a separate forward propagation through the network. Moreover, perturbation-based approaches may underestimate the importance of features that have saturated their contribution to the output.

Another class of algorithms, to which DeepLIFT belongs, is referred to as back-propagation-based approaches. The algorithms that follow this approach start from the model's output and assign importance scores to the neurons in the layer immediately below. The process is repeated through the lower layers until the input is reached. This strategy allows this type of algorithms to be computationally efficient, since they only need a single backward pass to assign importance scores to every input feature.

Most back-propagation-based algorithms use the gradient of the output to generate an explanation. Instead, DeepLIFT frames the problem of computing the importance scores in terms of differences from a "reference activation", where reference activation is the activation a neuron has when presented with the reference, or neutral, input. This reference input is chosen by the user according to his domain-specific knowledge.

TREPAN

TREPAN is an explanation algorithm which falls into the category of global surrogate models. Introduced by Craven and Shavlik, its goal is "to extract a comprehensible, symbolic representations from trained neural networks" [18]. This is achieved by querying the neural network as a black-box model (in the same way LIME or

SHAP does) in order to build a decision tree which approximates the behaviour of the original model. The decision trees that are generated using the TREPAN algorithm preserve an accuracy comparable to the one obtained by the original neural network, while providing a more comprehensible representation which can be interpreted by humans.

The algorithm works in a similar way to other decision tree algorithms such as CART [15] and C4.5 [33], with the target function being the concept represented by the network. However, it exploits an *oracle* which is queried during the learning process. The role of the oracle is to classify the instances that are presented as queries. There are three main purposes why the oracle is used: first, to predict the labels for the network's training examples; second, to determine the splits for each internal node of the tree; third, to determine the class distribution within a node of the tree.

TREPAN can be considered as a model-agnostic algorithm, since it does not place any requirement on the underlying model. The oracle acts like a black-box model, and the decision tree is built based only on the inputs and outputs of the oracle. This means that, though the algorithm was introduced to approximate the behaviour of neural networks, it can also be applied to models trained using any other machine learning algorithm.

Contrastive explanation method

The contrastive explanation method (CEM) by IBM's researchers Dhurandhar et al. [19] is yet another model-agnostic tool for generating model explanations. It stands out from other methods such as LIME or SHAP because its focus is on justifying the classification of an input by a black-box model based on which parts of the input should be minimally and sufficiently present and, analogously, which parts should be necessarily *absent*. Hence, its goal is to provide contrastive explanations that align more closely to the way humans think.

A contrastive explanation tries to justify a prediction based on what is *absent*, rather than what is *present*. As the authors of the algorithm state, this type of explanation is more natural for humans and is commonly used in domains such as health care and criminology. In these domains, a complete explanation is made of two key notions:

Persistent positives The factors whose presence is minimally sufficient in justifying the final classification;

Persistent negatives The factors whose absence is necessary in asserting the final classification.

To generate this kind of explanation, CEM follows a three-steps approach: first, it finds those features in the input that are minimally sufficient to yield the same classification (i.e. the *pertinent positives*); second, it search for the minimal amount of features that should be *absent* from the input to prevent the classification result from changing (i.e. the *pertinent negatives*); third, it exploits a state-of-the-art convolutional autoencoder in order to obtain explanations that more closely align to the reality.

2.4 Tools and frameworks for model interpretability

In this chapter, a number of techniques to evaluate machine learning models' performances, along with various algorithms designed to provide human-friendly explanations of the predictions performed by these models, have been introduced. This last section provides an overview on some frameworks, libraries and tools that have been proposed to further inspect the machine learning models. Most of these frameworks exploit at least one of the techniques presented in this chapter, sometimes improving them to achieve better results. Some also provide additional tools that can be useful to both data scientists, researchers, and non-experts.

2.4.1 IBM Watson OpenScale

IBM OpenScale is a commercial solution belonging to IBM's Watson suite, introduced with the goal of providing a platform that could be used by businesses to operationalize their artificial intelligence systems and to extend their deployments to the whole enterprise. It offers several tools that helps both data scientists and managers to monitor and understand their model's outcomes. OpenScale not only provides an online application to navigate through the results by means of a graphical user interface, but it also offers an API which allows to programmatically access the platform's services. Among its features, the most relevant are:

- Tracking of several metrics, such as accuracy, recall, and precision, over time;
- Graphical representation of the tracked metrics;
- Storage of the predictions computed by a model, with the possibility to require explanations for the single prediction;
- Based on the predictions performed by the model, detection of biases embedded in the model;
- Mitigation of the previously identified biases.

In this section, the focus is solely on the first three features. The remaining two will be further investigated in the chapter related to machine learning fairness.

Metrics tracking

The platform allows the setup of a monitor that can be used to collect feedback data from the user and to process those data in order to compute several metrics, such as accuracy, AUC-ROC, and F-Score. The feedback data can be submitted to OpenScale from an external application through a dedicated RESTful API. Once the number of feedbacks collected is sufficient to make a statistically sound estimation of the model performance, the metrics are made available through a graphical user interface. The provided interface shows how the model performs under the selected metric in a given time frame. It is also possible to select a minimum threshold: if the model performance under the selected metric is below the user defined threshold, then an alert is thrown by the application. Finally, the application automatically checks for new feedback data once every hour, and recomputes both the metrics and the graphs.

Figure 2.10 presents an example of the previously described interface, starting from a default demo data set provided by IBM. The image depicts a graph of the AUC-ROC metric's trend over the previous week. The teal line is used to represent the tracked metric, while the red line depicts the user-defined threshold. On the left, there are some additional suggestions to help interpreting the graph, as well as further details related to the next planned evaluation. By clicking on a data point of the graph, another view, which presents a more detailed view of the selected point, is provided. A screenshot of this last view is presented in Figure 2.11.

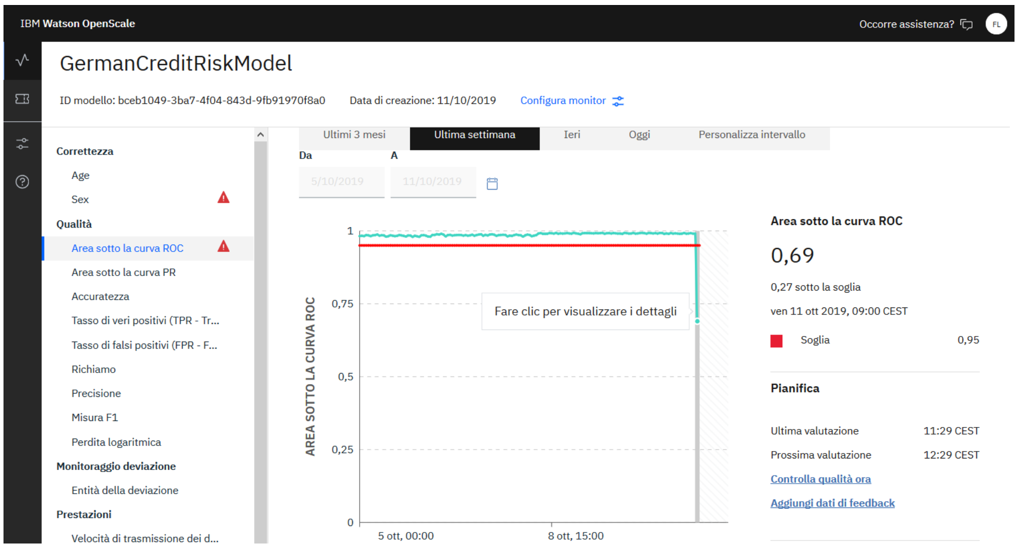
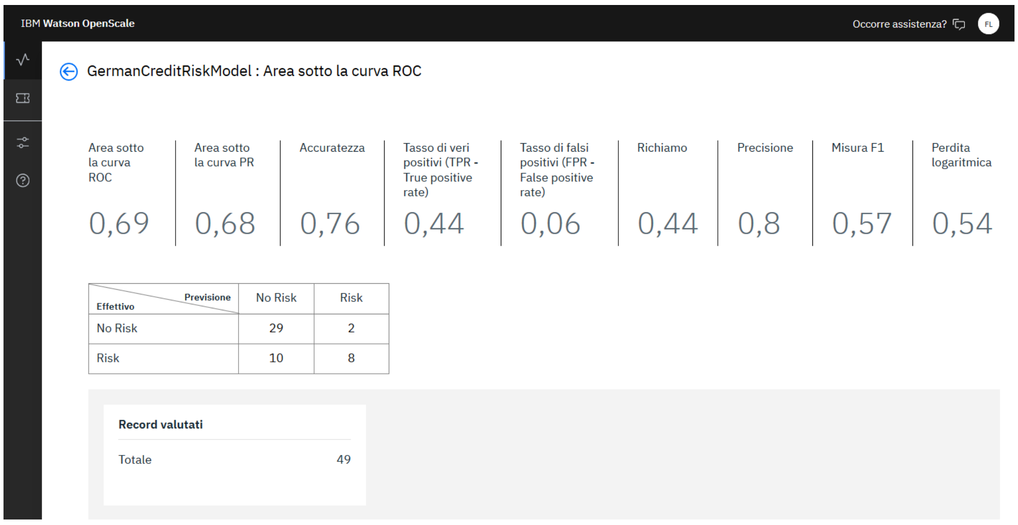


Figure 2.10: Demo view of one of the graphs generated by OpenScale, which depicts the auc-roc metric's trend over the previous week



GermanCreditRiskModel : Area sotto la curva ROC

Area sotto la curva ROC	Area sotto la curva PR	Accuratezza	Tasso di veri positivi (TPR - True positive rate)	Tasso di falsi positivi (FPR - False positive rate)	Richiamo	Precisione	Misura F1	Perdita logaritmica
0,69	0,68	0,76	0,44	0,06	0,44	0,8	0,57	0,54

Effettivo \ Previsione	No Risk	Risk
No Risk	29	2
Risk	10	8

Record valutati

Totale	49

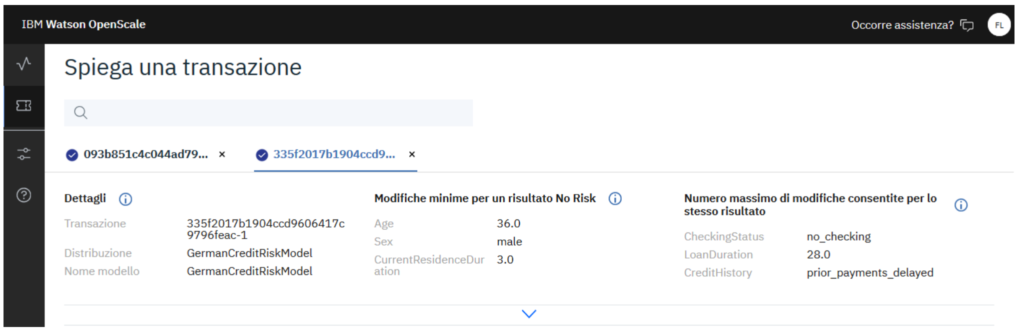
Figure 2.11: A detailed view of the information provided by OpenScale for a single data point

Prediction storage and explanation

In a similar way, OpenScale allows to monitor the predictions made by the machine learning model (in Watson's dialect, a prediction is referred to as a transaction). To correctly query the model, it must be deployed on the IBM Cloud, using another IBM's service called Watson Machine Learning. Through the graphical interface provided by this tool, it is possible to fill a form which allows to specify an instance for which to obtain a prediction. Alternatively, Watson Machine Learning provides a RESTful API that can be queried by an external program. The prediction is then stored within the OpenScale platform, and it can be accessed through a unique identifier to obtain an explanation for the given prediction.

Under the hood, OpenScale leverages LIME to provide an explanation for the specified prediction. LIME's results are then normalized in some way so that the scores associated with each feature sum up to 100. The results are further enriched by applying the Contrastive Explanation Method that was briefly presented in the previous section. It is possible to access these explanation either through the provided GUI, or through a RESTful API that returns the scores associated with each input feature. If the explanation is accessed through the RESTful API, it is also possible to disable the use of the CEM algorithm. In this situation, the results will be computed using plain LIME.

Figure 2.12 and Figure 2.13 show an example of an explanation provided by OpenScale, divided into two sections. Figure 2.12 shows the explanation generated using the CEM algorithm, that is, the information about the identified pertinent positives and pertinent negatives. The second figure (Figure 2.13) depicts the explanation computed using the LIME algorithm, along with the confidence of the prediction. While most of the times OpenScale tries to redistribute the scores associated with each item to match the percentage expressed by the confidence score, this is not always possible (as in the situation presented as an example).



Spiega una transazione

093b851c4c044ad79... ×
335f2017b1904ccd9... ×

Dettagli

Transazione	335f2017b1904ccd9606417c 9796feac-1
Distribuzione	GermanCreditRiskModel
Nome modello	GermanCreditRiskModel

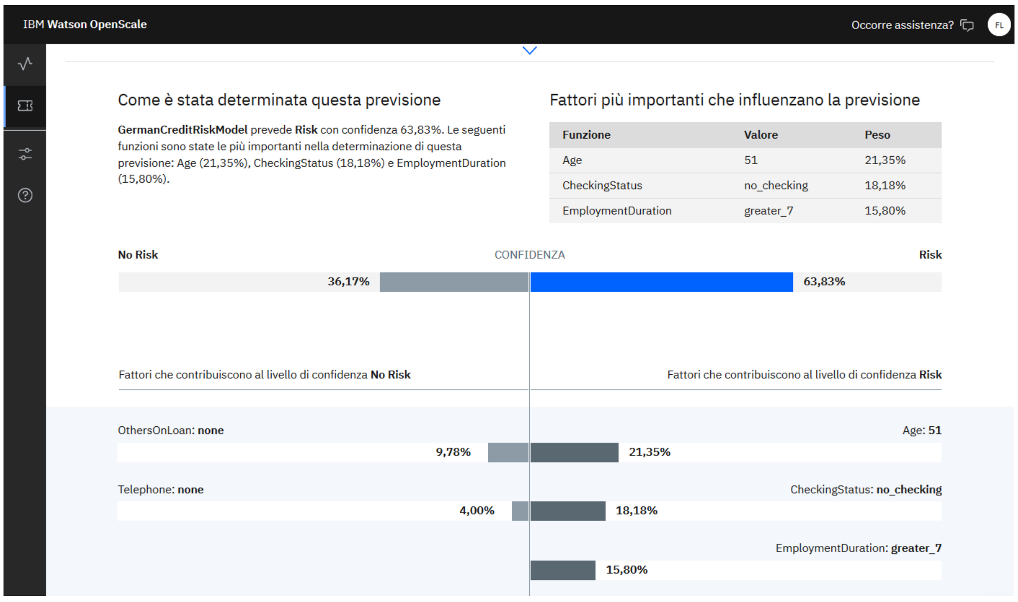
Modifiche minime per un risultato No Risk

Age	36.0
Sex	male
CurrentResidenceDuration	3.0

Numero massimo di modifiche consentite per lo stesso risultato

CheckingStatus	no_checking
LoanDuration	28.0
CreditHistory	prior_payments_delayed

Figure 2.12: OpenScale view of the scores computed by CEM



Come è stata determinata questa previsione

GermanCreditRiskModel prevede **Risk** con confidenza 63,83%. Le seguenti funzioni sono state le più importanti nella determinazione di questa previsione: Age (21,35%), CheckingStatus (18,18%) e EmploymentDuration (15,80%).

Fattori più importanti che influenzano la previsione

Funzione	Valore	Peso
Age	51	21,35%
CheckingStatus	no_checking	18,18%
EmploymentDuration	greater_7	15,80%

Fattori che contribuiscono al livello di confidenza **No Risk**

Fattori che contribuiscono al livello di confidenza **Risk**

Figure 2.13: OpenScale view of the scores computed by LIME

2.4.2 Google What-If tool

The *Google What-If Tool* (WIT) is an interactive tool which allows the user to investigate the model's behaviour and performance through a visual interface [2]. WIT is an initiative of Google's *PAIR* (People + IA Research) team. It has been proposed as a tool to enable people to evaluate machine learning models without the need to write complex code. Through WIT, a user can investigate the behaviour of multiple models, compare them, and extract insights from them. The tool offers a wide array of features:

Model comparison WIT allows to compare the performance metrics of different models within the same workflow. To accomplish this task, WIT exploits the ROC curve and the PR curve (where PR stands for *Precision* and *Recall*) to compare the different models. Additionally, WIT allows to compare the confusion matrices of the evaluated models. It also provides possibility to modify the threshold used by the models to select a label based on the prediction score and to see the difference in the results in real-time.

Inference results visualisation WIT leverages another component created by PAIR called *Facets Dive* [3] to provide a custom visualisation tool which offers the user the possibility to create plots where the instances of the dataset are arranged based on the selected features. After the graph is created, the user can select each instance to visualise all its attribute values. Another functionality allows the user to group the instances in different bins. Through this feature the user is able to analyse the model's behaviour separately for each of the generated groups (Figure 2.14).

Similar data points comparison By selecting a data point from the Facets plot, the user can arrange all other points based on different similarity measures.

Data point variation Once an instance is selected, the user can modify the values of each feature to see how the prediction changes. This allows the user to intuitively experiment with different values and better understand how the model behaves.

Nearest counterfactual comparison After the user has selected a specific data point, WIT allows him to find the most similar instance across the dataset for which a different label was predicted. By comparing similar instances that were classified in opposite ways, the user is able to extract those features that are crucial in predicting the class to which an instance belongs. Figure 2.15 shows an example of such comparison. On the bottom left of the screen, the selected instances' feature values are compared. On the right, the two data points are visually represented through a scatter plot.

Algorithmic fairness Finally, WIT is able to check if the model satisfies a number of algorithmic fairness constraints such as *statistical parity* and *equality of opportunity* (these metrics are covered in more detail in Chapter 3).

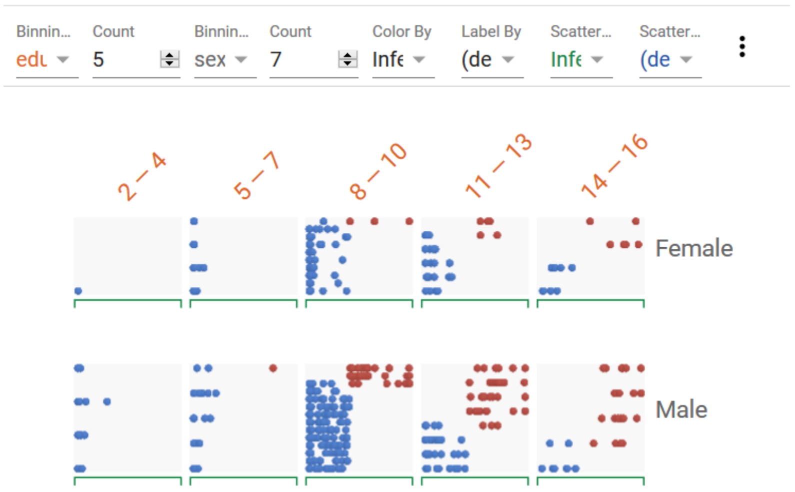
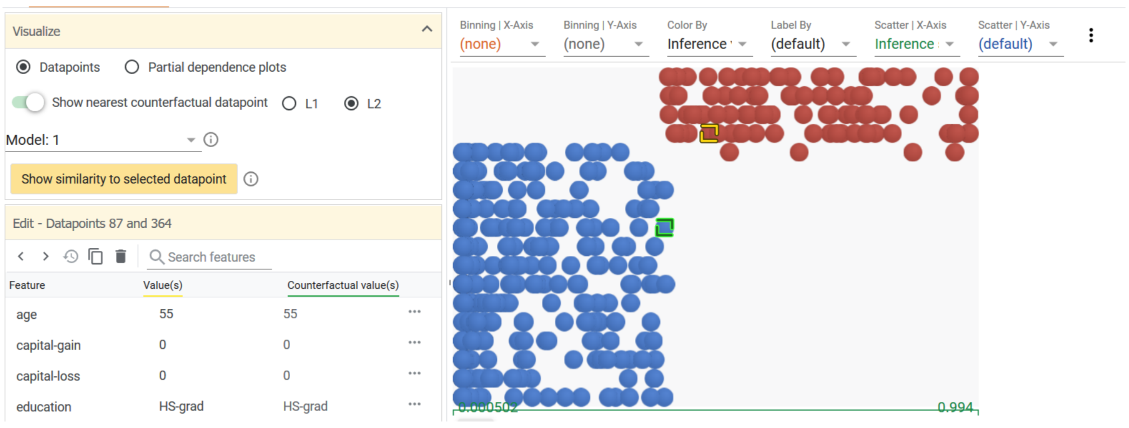


Figure 2.14: Google WIT's scatter plots for different bins



Model: 1

Edit - Datapoints 87 and 364

Feature	Value(s)	Counterfactual value(s)
age	55	55
capital-gain	0	0
capital-loss	0	0
education	HS-grad	HS-grad

Binning | X-Axis (none)
Binning | Y-Axis (none)
Color By Inference
Label By (default)
Scatter | X-Axis Inference
Scatter | Y-Axis (default)

Figure 2.15: Google WIT's example of the selection of the nearest counterfactual data point

2.4.3 SHAP Framework

Authors Lundberg and Lee not only proposed a new theoretical foundation for model explanation methods and a novel model-agnostic algorithm to explain black-box models' predictions [30], but they also provided a Python library which implements the SHAP algorithm and includes additional tools to gather insights from a black-box model [4]. The library exploits SHAP to generate global explanations by computing the SHAP values for every instance of the background data and by combining the obtained results into a matrix of SHAP values. This matrix is then used to interpret how the whole model works.

Although SHAP values describe what is the marginal contribution of each feature to the final output, the plot chosen by the authors to visualise each feature's contribution interprets SHAP's results in a different manner. Each SHAP value is represented as a "force" that pushes the prediction from the base value to the actual model's output. As Figure 2.16 shows, each feature either pushes the prediction to a higher value (in which case the contribution is shown in red), or it pushes it to a lower result (shown in blue). By summing up all the contributions, the difference between the base and final output is obtained.

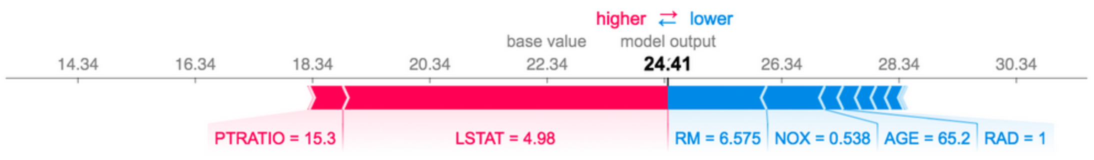


Figure 2.16: A force plot used to graphically represent an explanation based on SHAP. [4]

Along with this new representation of the SHAP values, the authors exploit SHAP to provide other plots that help to explain and evaluate the model. These graphs are presented below.

Multiple explanations

Apart from explaining single predictions, SHAP provides a way to graphically visualise an explanation for multiple predictions at once. To do so, SHAP simply takes the force plot previously presented, rotates it by 90 degrees, and stacks each explanation horizontally. Figure 2.17a shows an example of this plot. Each position on the x-axis represents the force plot for a single prediction, and the samples are sorted by similarity. By using a notebook, it is possible to interactively navigate this plot by selecting different criteria on which to base the clustering of the samples.

Feature importance

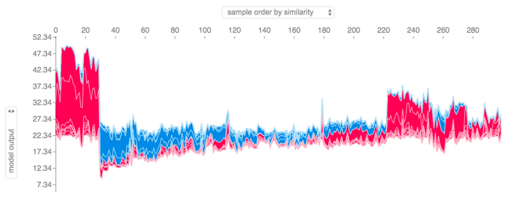
Starting from a matrix of SHAP values, the library computes the average of the SHAP values for each feature in order to estimate which are the most important features. The higher the average SHAP value, the more important the associated feature is. Features are then sorted to provide the user an indication of which features influence the model the most. Though the computation of the importance score is different, the result is similar to the one that would be obtained through the *permutation feature importance* measurement [16] [22], which is not covered in this work. An example of this plot, with the features arranged in descending order of importance, is shown in Figure 2.17b.

Summary plot

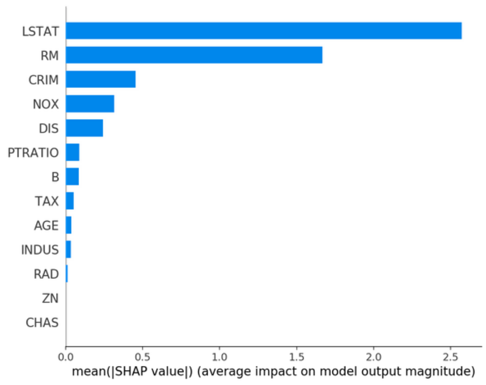
Similarly to feature importance, the summary plot provided by SHAP is used to show which are the most relevant features in our data set. In addition, it offers insights on the impact that each feature has on the instances of the input set. Each point on the plot describes the SHAP value for a single instance and a single feature. As Figure 2.17c shows, features are distinguished by plotting each point on a different line based on the feature it refers to; the horizontal position of the points is used to represent the SHAP value, and the colour is used for quantifying the value associated with each point. When several points have the same SHAP value, instead of overlapping, they pile up in order to better provide a sense of density.

Dependence plot

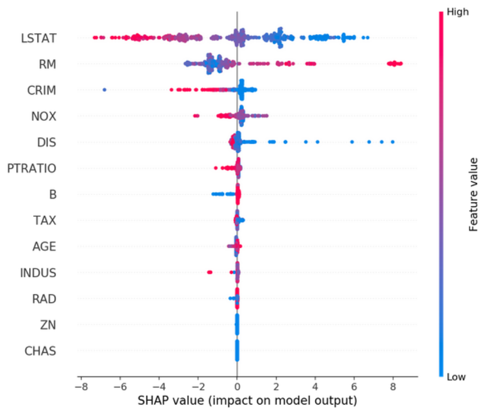
SHAP dependence plots are used to show the impact that one feature has across the whole data set. This is achieved by plotting, for each instance, its feature value (on the x-axis) against its SHAP value (on the y-axis). The plot can be enriched by using different colours to describe the value related to a second feature, thus allowing to simultaneously analyse the relationship between two feature values and the SHAP value for each instance. Figure 2.17d depicts an example of a dependence plot. This results are similar to the ones that would be obtained by using partial dependence plots.



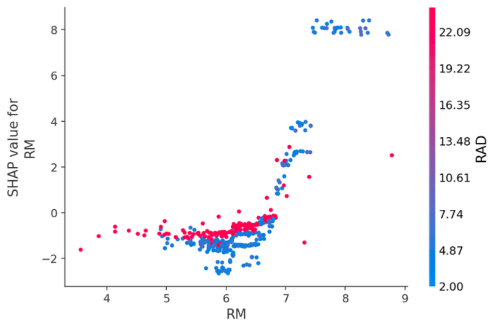
(a) SHAP multiple explanation plot



(b) SHAP feature importance



(c) SHAP summary plot



(d) SHAP dependence plot

Figure 2.17: Examples of different plots provided by SHAP library. [4]

2.4.4 Skater

Skater is an open-source Python library which enables model interpretation regardless of the specific model being analysed [5]. It offers a unified access to implementations of various algorithms for black-box model interpretability (in the terminology used by Skater, these are referred to as post-hoc interpretations). The algorithms contained in Skater are able to provide both local and global interpretations of the model at hand. The main algorithms included in Skater are:

Feature importance It's used to estimate the degree to which a prediction model relies on a particular feature. To compute the importance score for a single feature, Skater first perturbs the feature being evaluated, then it measures the entropy in the change of the predictions. By measuring how the prediction changes in function of the perturbation of the single feature, Skater is able to give an importance score to the evaluated feature.

Partial dependence plots Partial Dependence describes the marginal impact a feature has on model's predictions, while holding other features in the model constant. It is used to highlight the effect of a feature on the predicted outcome of a previously trained model. The algorithm is an adaptation of the algorithm proposed by Hastie et al. [25].

LIME One of the ways Skater uses to provide local interpretations of a black-box model is through the LIME algorithm. Under the hood, what Skater does is to simply provide a wrapper for the original implementation of LIME as proposed by authors Ribeiro et al. [6].

Tree surrogates This algorithm is inspired by the TREPAN algorithm proposed by Craven [18]. Its goal is to generate a global surrogate model (in the form of a decision tree) to provide a more interpretable approximation of a black-box model. See the section on model-agnostic algorithms for more information.

DNN specific algorithms Skater allows access to two algorithms specific to Deep Neural Networks which provide explanation capabilities for single predictions, namely Layerwise Relevance Propagation and Integrated Gradient. These algorithms both exploit a back-propagation-based approach that relies on the gradient of the output to infer the weights to give to different features.

2.4.5 ELI5

ELI5 is a Python package which helps to debug machine learning classifiers and explain their predictions. While the other packages presented in this section tries to provide access to model-agnostic strategies to interpret black-box models, ELI5 is a framework which allows a uniform access to a variety of model-dependent algorithms to compute model's explanations. The package mostly focuses on producing explanations for linear and tree-based models, but it also provides implementations for some black-box algorithm.

There are two major high-level functions that comes with ELI5: one that is used to compute the feature importance scores for the whole model, and a second function used for estimating the weight that each feature value had on a given prediction. These functions then dispatches the work to a concrete implementation based on the type of model being evaluated. There is also a model-agnostic implementation for the computation of feature importances. The provided algorithm follows a permutation based approach, that is, it measures how predictions change when a feature is not available. To do so, it replaces the target feature with random noise which is drawn from the same distribution as the original feature values.

ELI5 package also offers access to locally faithful explanations by exploiting the LIME algorithm. As opposed to other tools such as OpenScale and Skater, ELI5 does not leverage the original implementation by Ribeiro et al. Instead, it provides its own custom implementation of the authors' algorithm. The major differences between the two implementations are related to the different UI, the classifiers supported by the two libraries, and the local surrogate model used to locally approximate the black-box model.

Chapter 3

Machine learning fairness

The previous chapter began by describing how the adoption of machine learning techniques can improve existing businesses. It then focused on one of the main issues that arises when trying to leverage machine learning to provide insights and improve the efficiency of a given task: the requirement that individuals must be able to understand and correctly interpret the output of a previously trained model. While being able to evaluate the performance of a model and understand how it works is crucial in the process of building a trustworthy artificial intelligent system, there is another, more subtle problem, that in recent years has been addressed by several researchers and that is strictly related to the ability to interpret a model's output: the need to make machine learning models accountable for the fairness of their decisions.

The main reason why machine learning is becoming increasingly popular is due to its high efficiency and effectiveness as a decision support tool. Machine learning models can frequently outperform even experts' judgement, leading to more accurate and objective decisions. The problem is that, while the decisions of an artificial intelligence system might be beneficial for the company, they might also impact other people's lives as a consequence. Machine learning is increasingly being used to support processes such as school admission, loan approval, and job offering. Hence, these algorithms are starting to affect every stage of the life of an individual. In these use cases, a bias in the machine learning prediction can have a huge impact on the groups of people to which this decision refers.

The reason why a machine learning model can be biased is grounded in the fact that these systems are trained by examples. The whole process behind any machine learning algorithm is to extract some general rule from a set of real world examples by uncovering hidden patterns that are invisible even to an expert's eye. The extracted rules can then be used to make reasonable assumptions on previously unseen cases. Thus, to successfully train a good machine learning model requires high-quality evidences. However, when using historical data to model human behaviour, the provided examples often reflect the prejudices of the people that made these decisions in the first place. These biases then become the foundation from where machine learning systems are trained. As a result, the predictions made by these systems may favour a majority group over some minority. In other cases, there may be some minority group for which there are not as many training data as for the majority group. Moreover, the features that characterise these groups may be less informative or less reliable. As a consequence, the trained model will be far less accurate for the minority groups with respect to the accuracy measured for the majority groups, which might result in unfair decisions from the model. In this regard, a canonical example comes from the COMPAS algorithm, which is used by several courts in the U.S. as a risk assessment tool to estimate the probability of a person to commit another crime. Based on the algorithm prediction, judges uses COMPAS to decide whether to release an offender. An analysis published by ProPublica in 2016 demonstrated how the algorithm was unfairly judging black offenders, which were wrongly labelled as high risk individuals at almost twice the rate as white defendants [7].

Another problem related to fairness is that even a small bias, compounded over time, might result in harmful decisions for a group of individuals. This is due to the fact that the current model's decisions influence future observations, which will probably confirm the predictions made by the system and will provide fewer examples that contradicts such predictions. An example that is frequently used to clarify this last point is the following one [11]: suppose that the police department of a city decides to use a machine learning system to predict which areas of the city are at high risk for crime. Records of past crimes are used by the police to train the model and, based on the system's predictions, the department dispatches more officers to the areas where crimes are most likely to occur. Such predictions might also lower the officers' threshold for stopping or arresting people. As a consequence, a higher number of crimes will be prevented in those areas, which will provide a validation for the model's behaviour and might further intensify the patrols in those parts of the city. In the meanwhile, the police might neglect other areas in which crime rate is growing, which prevents new crimes from being unveiled and new contradicting observations from being collected.

In the rest of this chapter, the first section reviews some of the most popular criteria proposed in the literature for measuring fairness. The second section presents some algorithms that can be leveraged to mitigate the biases embedded in a machine learning model. Finally, the third section reviews some of the tools available to identify the presence of prejudice within a previously trained model, as well as the frameworks that provide an implementation of the bias mitigation algorithms presented in Section 3.2.

3.1 Fairness definitions and relevant metrics

Algorithmic fairness refers to the capability of an artificial intelligent system to guarantee the fairness of its decisions. For a system to be considered fair, its output must be evaluated based on some fairness criteria. Hence, the first step to identify the presence of prejudice within a machine learning model is to provide a definition of what can be considered a fair outcome. By choosing how fairness is defined, different metrics can then be used to objectively assess if the model behaves in an unfair way. Unfortunately, due to the fact that fairness in machine learning is a relatively new subject, there is no consensus in the literature over what fairness exactly is and which criteria should be followed. Moreover, different terms have been used in the literature to provide equivalent definitions of fairness. This section outlines the main definitions that have been proposed, the relationships between these criteria, and which are the strengths and weaknesses of each approach.

3.1.1 Fairness criteria

Defining a fairness criterion is a complex task due to the different ways biases can arise. What can be considered fair in a specific context might be unfair in another one. Furthermore, different people have different sensibilities about what is fair and what is not. Generally speaking, fairness can be defined in two ways:

Individual fairness According to this definition, similar individuals should be treated similarly.

Group fairness The population should be split into several groups based on some sensitive attribute. After the split, a statistical measure has to be computed for each group. For group fairness to be satisfied, the selected measure should be equal across all groups.

Furthermore, group fairness can be interpreted in opposite ways, based of two different worldviews [41] [23]:

What You See Is What You Get According to this worldview, the predicted result reflects the ability of each individual to perform a task.

We're All Equal This worldview states that every group is identical with respect to the task at hand. Hence, if there is a difference in the predicted outcomes of two groups, this is due to a bias that influences the way the outcome is computed.

Based on the chosen interpretation, different definitions can be proposed. In this section, three different criteria for evaluating fairness are presented, along with some variants obtained by relaxing the original definitions. While giving the most general formulation, the main discussion revolves around the special case of a binary classifier with a single sensitive attribute. Furthermore, a number of variables are used throughout the section:

- B denotes the sensitive feature. The values that can be assigned to this feature are split into two group a and b. a refers to the privileged group, that is, the group for which the classifier predicts a higher rate of favourable outcomes. Instead, b is used to describe the unprivileged group that might be harmed by an unfair classifier.
- R is the score based on which the classifier makes a prediction C.
- Y describes the target variable that the model tries to predict.

To better understand how some of these criteria works, an example is used to further explain some of the concepts presented. The context is the one of a company that wants to use a binary classifier as a support tool to predict whether an applicant should be hired for a job position. In this context, B refers to some sensitive attribute (e.g. gender or race), C is the prediction made by the classifier (hired / not hired), and Y is used to denote if the applicant is truly capable of the position.

Fairness through unawareness

A naive approach to solving the fairness issue is to simply ignore the sensitive attributes. The idea is based on the fact that, by removing the attribute that carries the prejudice, the resulting model would be devoid of biases. This criteria aligns with the notion of *disparate treatment* available in legal literature, which states that sensitive attributes should not be used.

While this approach might seem reasonable, most of the times its effects on the biased classifier are null, if not harmful. Indeed, this strategy ignores the presence of small statistical correlations between the removed feature and other available attributes. When many of such features are available in the data set, these attributes can act as a proxy for the ignored feature, which makes the removal of the sensitive attribute useless.

Independence

Independence is one of the most widely used criterion for assessing fairness. Variants of this concept include *statistical parity*, *group fairness*, *demographic parity*, and *disparate impact*. Given two variables B and C, this criterion is satisfied if B is independent of C ($B \perp C$). In the case of binary classification, this formulation can be expressed as:

$$\mathbb{P}\{C = 1 | B = a\} = \mathbb{P}\{C = 1 | B = b\} \tag{3.1}$$

for each group a and b. If we consider 1 as the positive outcome, this condition means that the percentage of positive outcomes for both groups is the same, hence there is no privileged group.

This notion can be relaxed by accepting the presence of a disparity between the groups, as long as this difference is relatively small. Hence, *independence* can be expressed in two different ways, which are more or less equivalent:

Disparate impact The disparity can be expressed as the ratio of rate of favourable outcome for the unprivileged group to that of the privileged group. By considering a as the privileged group and b as the unprivileged one:

$$\frac{\mathbb{P}\{C = 1|B = b\}}{\mathbb{P}\{C = 1|B = a\}} \geq 1 - \epsilon \tag{3.2}$$

where an acceptable value for ϵ, some authors argue [21], should equal to 0.2. With this setting, the ratio is compliant with the *80%-rule* supported by the U.S. Equal Employment Opportunity Commission [13].

Statistical parity difference Otherwise, independence can be measured in terms of the difference of the rate of favourable outcomes received by the unprivileged group to the privileged group:

$$|\mathbb{P}\{C = 1|B = b\} - \mathbb{P}\{C = 1|B = a\}| \leq \epsilon \tag{3.3}$$

The reason why independence is frequently used in the literature is because it is one of the few definitions that has legal support. The already cited *80%-rule*, or *four-fifth rule*, specified in the U.S. Equal Employment Opportunity Commission guidelines, prescribes that a selection rate for any group (classified by race, orientation or ethnicity) that is less than four-fifths of that for the group with the highest rate constitutes evidence of *disparate impact*, that is, discriminatory effects on a protected group.

One of the main drawbacks is that independence doesn't take into account any possible correlation between the sensitive feature B and the actual outcome Y. Furthermore, by applying independence alone there is a risk of the company being guilty of *laziness*: if a company diligently hires applicants belonging to group a, then randomly selects other applicants from group b at the same rate as a, independence would still be satisfied, but there would be a disproportion in the number of qualified applicants between the two groups. As a consequence, the feedback data that will be collected in the future will only serve as a validation for the prejudice rightfulness. As Barocas et al. note [11], this might happen even without the company being consciously negligent. Indeed, this behaviour might arise from the fact that the company has not enough training data related to group b, which causes the machine learning model to be less accurate, hence, more prone to errors in selecting the candidates.

Separation

In many occasions, a difference in the way two groups are treated might be justified by a business necessity. This happens when there is a correlation between the sensitive feature and the target variable. The separation criterion, also known as *equalized odds*, was first proposed in H, Price, Srebro (2016) [24] and Zafar, Valera, Rodriguez, Gummadi (2016) [14]. This approach seeks to acknowledge the existence and rightfulness of this correlation to the extent that is justified by the target variable. Given the three variables (B, C, Y), the separation criterion is satisfied if B is independent of C conditional on Y $(B \perp C \mid Y)$. If C is a binary classifier, then the criterion can be formulated as:

$$\begin{gathered}\mathbb{P}\{C = 1 \mid Y = 1,\ B = a\} = \mathbb{P}\{C = 1 \mid Y = 1,\ B = b\} \wedge \\ \mathbb{P}\{C = 1 \mid Y = 0,\ B = a\} = \mathbb{P}\{C = 1 \mid Y = 0,\ B = b\}\end{gathered} \tag{3.4}$$

where $\mathbb{P}\{C = 1 \mid Y = 1\}$ represents the true positive rate of the classifier, and $\mathbb{P}\{C = 1 \mid Y = 0\}$ represents its false positive rate. Thus, the goal of the separation criterion is for both groups to obtain the same false positive rate and false negative rate.

In many situations, the two rates might have different weights. Hence, it might be more interesting to relax the condition expressed in Equation 3.4 to account only for the true positive rate or vice versa. *Equality of opportunity* is the name that is used when referring to the variant of the separation criterion that considers only the true positive rate:

$$\mathbb{P}\{C = 1 \mid Y = 1,\ B = a\} = \mathbb{P}\{C = 1 \mid Y = 1,\ B = b\} \tag{3.5}$$

This approach overcomes the main limitation of the *independence* criterion, namely, the issue that arises when only one of the considered groups is evaluated properly. On the other hand, *separation* might not help closing the gap between the two groups in the long run. Using the same example as before, there might be a disproportionate amount of qualified individuals belonging to group a with respect to the applicants in group b. In this situation, the *separation* criterion might justify the higher acceptance rate experienced by group a. If the offer refers to a well-paid position, the disproportionate choice will impact the future perspectives of the two groups, which will increase the pre-existing gap in the long term.

Sufficiency

The *sufficiency* criterion, also referred to as *predictive rate parity*, is based on the idea that the sensitive feature might be already subsumed in the score used for predicting the target label. To put it differently, given three random variables (B, C, Y), the sufficiency criterion states that, if C is available, then B is not needed in order to compute the score of Y. More formally, variable C is sufficient for B if $Y \perp B \mid C$.

In a binary setting where $Y \in \{0,1\}$, a variable C is deemed as sufficient for B if, for all groups $a, b \in B$, the following condition is satisfied:

$$\mathbb{P}\{Y = 1 \mid C = y,\ B = a\} = \mathbb{P}\{Y = 1 \mid C = y,\ B = b\},\ \forall y \in \{0,1\} \tag{3.6}$$

In other words, the sufficiency criterion requires that the positive and negative predictive values should be equal across all groups.

Using *sufficiency* as a fairness definition guarantees that, if the applicant is hired, than his success probability will be the same independently of the group it belongs to. Thus, it is able to overcome the laziness issue. However, *sufficiency* shares with the *separation* criterion the limitation that it does not help to close the gaps between the privileged and unprivileged groups.

3.1.2 Relationships between different criteria

While it can be useful to consider different criteria when evaluating the presence of bias in an artificial intelligence system, these cannot be used simultaneously as hard constraint. This limitation arise from the fact that, with the exception of degenerate cases, the three criteria of *independence*, *separation*, and *sufficiency* are mutually exclusive. As a consequence, fairness can be achieved by finding an optimal trade-off between these three criteria.

Independence vs Sufficiency

Let C be the sensitive attribute and Y the target variable, where C and Y are not independent ($C \not\perp Y$). This assumption means that, given two groups a and b, one of the two groups has a positive outcome rate higher than the other one. In this situation, either *separation* holds or *sufficiency* does, but not both. As a proof:

$$\text{if } C \not\perp Y \text{ and } C \perp Y \mid B \implies C \not\perp B$$

Independence vs Separation

Let C be the sensitive attribute and Y the target variable, where C and Y are not independent ($C \not\perp Y$) and Y is a binary variable. If $B \not\perp Y$ is also satisfied, then independence and separation cannot both hold. As a proof:

$$\text{if } B \perp C \text{ and } B \perp C \mid Y \implies C \perp Y \text{ or } B \perp Y$$

Separation vs Sufficiency

Assume all events in the joint distribution of (B, C, Y) have positive probability. If C is dependent of Y, either separation holds or sufficiency does but not both. As a proof:

$$\text{if } C \perp B \mid Y \text{ and } C \perp Y \mid B \implies C \perp (Y, B) \implies C \perp Y$$

3.2 Bias mitigation algorithms

The need to achieve algorithmic fairness led many researchers to propose several strategies to cope with biases embedded in a machine learning model. These algorithms try to strike a balance between preserving the accuracy of the original model and enforcing the fairness in the model's predictions. The way they work mainly depends on two factors: the criterion selected for measuring the fairness, and the step in the workflow of a machine learning process in which they are applied. This section's structure mirrors this latter aspect and, for each step in a machine learning's workflow, presents a selection of the algorithms that has been proposed to pursue algorithmic fairness.

3.2.1 Pre-processing

Let X be a record in a training set D, A the set of non-sensitive features and B the sensitive feature in a training data set. The goal pre-processing algorithms is to learn a new representation Z which preserves the information correlated to the A features but ignores the information of B. This new representation will then be used as a training data set based on which a new model will be trained. As a result, the output model will be characterised by an accuracy comparable to the one that would be obtained by a model trained using also feature B, but its predictions will preserve the fairness criteria on which the selected algorithm is based.

Reweighing

The reweighing algorithm, first proposed by Kamiran and Calders [27], is a pre-processing algorithm which weights the examples in each (group, label) pair differently to ensure fairness before classification. In this algorithm, the chosen fairness criterion is the *independence* one. For instance, suppose a record X belongs to the unprivileged group b, that is, $X(B) = b$. If the target label is negative ($X(C) = -$), then a lower weight will be given to the sample. If the record's target label is positive ($X(C) = +$), then it will receive a higher weight. Vice versa, a record's belonging to the privileged group a will get lower weights when $X(C) = +$ with respect to the situation when $X(C) = -$.

Let D be the training dataset, $B \in \{a, b\}$ the sensitive attribute, where a is the privileged group and b the unprivileged group, and $C \in \{-, +\}$ is the target label. By using the *independence* criterion, the fact that D is unbiased means that B and C are statistically independent ($B \perp C$). In such situation, the expected probability that a record X belongs to group b and is classified as $X(C) = +$ would be:

$$\mathbb{P}_{exp}\{B = b,\ C = +\} = \frac{|\{X \in D \mid X(B) = b\}|}{|D|} \cdot \frac{|\{X \in D \mid X(C) = +\}|}{|D|} \tag{3.7}$$

If we drop the hypothesis $B \perp C$, which is the case that is observed in the real world, then the observed probability in D is:

$$\mathbb{P}_{obs}\{B = b,\ C = +\} = \frac{|\{X \in D \mid X(B) = b,\ X(C) = +\}|}{|D|} \tag{3.8}$$

If the expected probability is higher than the observed probability value, it shows that there is a bias towards group b. Therefore, a set of weights W, one for each object X, will be computed to compensate group b for the bias. The weights are computed as follows:

$$W(X) = \frac{\mathbb{P}_{exp}\{B = b,\ C = +\}}{\mathbb{P}_{obs}\{B = b,\ C = +\}}. \tag{3.9}$$

Through this process, the representation Z is learned by adding the weights W to set D. This new set is unbiased and can be used to train a fair classifier.

Disparate impact remover

As the name implies, this strategy leverages the disparate impact measure, also known as *independence*, to learn a fair representation Z. More precisely, the goal of the disparate impact remover algorithm is to increase group fairness while preserving rank-ordering within groups by editing feature values. The algorithm was proposed by Feldman et al. in 2014 [21].

To describe the disparate impact remover algorithm, the following definitions must be given:

- D is the training dataset, A denotes a single numerical attribute belonging to the set of non-sensitive features, B is the sensitive attribute, and C represents the predicted label;
- $A_b = \mathbb{P}\{A \mid B = b\}$ describes the marginal distribution on A conditioned on $B = b$;
- $F_b : A_b \rightarrow$ [0,1] is the cumulative distribution function for $a \in A_x$. F_x is the function used to rank the values of A;
- F_b^{-1} : [0,1] $\rightarrow A_b$ is the quantile function associated to F_x. For instance, $F_b^{-1}(1/2)$ is used to denote the values $a \in A_b$ such that $\mathbb{P}\{A \geq a \mid B = b\}$;
- $\hat{A}$ is called the "repaired" version of A, and Z is therefore the "repaired" version of D. Z is said to *strongly preserve rank* if, for any $a \in A_b$ and $b \in B$, its "repaired" counterpart $\hat{a} \in \hat{A}_b$ has $F_b(a) = F_b(\hat{a})$.
- $F_M^{-1}(u) = \text{median}_{b \in B} F_b^{-1}(u)$ describes the median distribution M in terms of the quantile function F_b^{-1}

The algorithm works by replacing A with $\hat{A}$, where $\hat{a} \in \hat{A}_b$ are computed as follows:

$$\hat{a} = F_M^{-1}(F_b(a)), \ \ \forall a \in A_b \tag{3.10}$$

As a result, the algorithm works by replacing the non-sensitive attribute(s) A with its "repaired" version $\hat{A}$ while maintaining both the sensitive feature B and the class C. The replacement is structured in such a way that, for any group $b \in B$, the ranking in $\hat{A}_b$ is the same as in A_b, but the "repaired" data set Z is fair. Moreover, by preserving the ranking, the resulting model will still be able to positively label the most promising records.

Optimized pre-processing

One last pre-processing strategy that is presented is the *optimized pre-processing* algorithm proposed by Calmon et al. [17]. This algorithm works by learning a probabilistic transformation which alters both the non-sensitive features and the label based on three different constraints. As with the *disparate impact remover* algorithm, *optimized pre-processing* retain the sensitive feature as-is.

Let A represent one or more non-sensitive attributes. B represents the sensitive features, and $C \in \{0,1\}$ is a binary target label. The goal of this algorithm is to find a mapping $p_{\hat{A},\hat{C} \mid A,B,C}$ that satisfies two properties:

- Starting from data set D, the mapping must able to obtain a new data set Z which can be used to train a new model. The new data set is created by replacing the pair $\{A, C\}$ with a new pair $\{\hat{A}, \hat{C}\}$ for each record. The new pair is computed by applying $p_{\hat{A},\hat{C} \mid A,B,C}$ to the triplet (A, B, C).
- The mapping should be applicable independently of the data, as soon as this data can be fed to the machine learning model. This means that $p_{\hat{A},\hat{C} \mid A,B,C}$ can also be applied to the testing set. Since in this setting the label $\{C\}$ is not available and $\{\hat{C}\}$ may not be needed, a reduced mapping function $p_{\hat{A} \mid A,B}$ can be used.

Additionally, the learned transformation must comply with the following constraints:

Discrimination control The mapping should limit the dependence of the outcome C to the sensitive feature B. As in the previous algorithms, the criterion used for estimating the fairness of the classifier is the *independence* one. The authors propose two alternative formulations of this constraint. The first one requires that the rate of positive outcomes for a given group should be similar to the rate of positive outcomes for the whole data set D:

$$\begin{aligned} J(\mathbb{P}\{\hat{C} = c \mid B = b\}, \mathbb{P}\{C = c\}) \leq \epsilon_{c,b} \\ \forall b \in B, c \in \{0,1\} \end{aligned} \tag{3.11}$$

where J is used to denote a distance function. A second formulation constrains the probability of $\hat{C}$ conditional on B to be similar for any pair of values of D:

$$\begin{aligned} J(\mathbb{P}\{\hat{C} = c \mid B = b_1\}, \mathbb{P}\{\hat{C} = c \mid B = b_2\}) &\leq \epsilon_{c,b_1,b_2} \\ \forall b_1, b_2 \in B, c &\in \{0,1\} \end{aligned} \tag{3.12}$$

To better align with the *80%-rule*, the authors suggest the following function to compute the distance:

$$J(p, q) = |\frac{p}{q} - 1| \tag{3.13}$$

Distortion control The mapping function should satisfy some distortion constraints to avoid large alterations of the original feature values which might distort the meaning of those attributes. An example given by the authors, in the context of a loan approval decision, is the mapping of a low credit score to a high credit score. Given a distortion metric $\delta : (A \times C)^2 \to \mathbb{R}$, where $\delta(a, c, a, c) = 0$ for all $a, c \in A \times C$, the conditional expectation of the distortion can be formulated as:

$$\begin{aligned} \mathbb{E}[\delta((a, c), (\hat{A}, \hat{C})) \mid A = a, B = b, C = c] &\leq l_{a,b,c} \\ \forall a, b, c &\in A \times B \times C \end{aligned} \tag{3.14}$$

where l is the desired level of control.

Utility preservation Finally, the distribution $(\hat{A}, \hat{C})$ should be statistically closed to the distribution of (A, C) to ensure the closeness of the behaviour between the models trained with the original data set D and the fair data set Z. Given a dissimilarity measure Δ, the dissimilarity between two probability distributions $(\Delta(\mathbb{P}(\hat{A}, \hat{C}), \mathbb{P}(A, C)))$ must be as small as possible.

By combining the constraints previously defined, the *optimized pre-processing* algorithm can be formulated as an optimization problem in which the goal is to minimize the dissimilarity measure while also satisfying the other two constraints. The result is the transformation $p_{\hat{A},\hat{C} \mid A,B,C}$.

$$\begin{aligned} \min_{p_{\hat{A},\hat{C} \mid A,B,C}} \quad & \Delta(\mathbb{P}(\hat{A}, \hat{C}), \mathbb{P}(A, C)) \\ \text{subject to} \quad & J(\mathbb{P}\{\hat{C} = c \mid B = b\}, \mathbb{P}\{C = c\}) \leq \epsilon_{c,b} \\ & \mathbb{E}[\delta((a, c), (\hat{A}, \hat{C})) \mid A = a, B = b, C = c] \leq l_{a,b,c} \\ & \forall a, b, c \in A \times B \times C \end{aligned} \tag{3.15}$$

3.2.2 In-processing

The class of the *in-processing* algorithms includes those approaches that try to enforce algorithmic fairness by changing the learning strategy of the model. These algorithms usually work by adding a regularisation term or another constraint to the existing optimisation function, while leaving the training data as-is.

Adversarial debiasing

Adversarial debiasing is an in-processing technique proposed by Zhang, Lemoine, and Mitchell [42]. This algorithm learns a classifier that, while maximising the prediction accuracy, it simultaneously reduces an adversary's ability to determine the sensitive attribute from the predictions of the trained model. It has the additional benefit that it is not restricted to the *independence* criterion and can be adapted to satisfy other fairness definitions.

To describe how the algorithm works, a neural network will be used as an example even though the algorithm can be classified as model-agnostic. Let X be the input based on which the model will be trained, C the output variable, and B the sensitive feature. A classifier P is characterised by a function $\hat{C} = f(X)$ and is usually given access to the sensitive variable B. In the description that follows, $\hat{C}$ is referred to as the output layer of the network. Let's assume that the model is trained by modifying the weight W to minimise some loss function $\mathfrak{L}_P(\hat{c}, c)$, and that a gradient-based method is used to accomplish this task. The output layer $\hat{Y}$ is used as an input to another model A called the *adversary*, which goal is to predict the sensitive feature B based on another loss function $\mathfrak{L}_A(\hat{b}, b)$ and a different set of weights U. Additionally, the adversary may have other inputs, based on the fairness criterion that the algorithm is trying to enforce. Figure 3.1 depicts the context described above. The goal of the *Adversarial debiasing* algorithm is to update the set of weights W in order to protect the sensitive attribute from being discovered by the adversary.

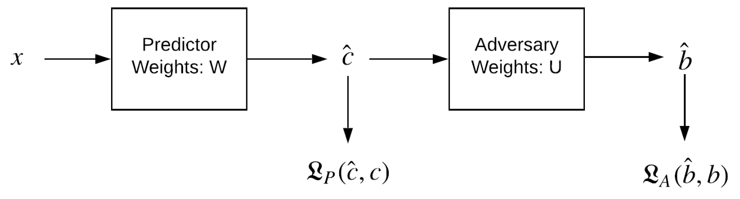


Figure 3.1: The architecture of the adversarial network, as described by the adversarial debiasing algorithm.

Adversarial debiasing works by enriching the machine learning algorithm from which the model is trained with a number of additional steps. At each training time step, the weights U are updated based on the output $\hat{C}$ of the network P and $\mathfrak{L}_A$ is minimised according to the gradient $\nabla_U \mathfrak{L}_A$. Next, W is modified by applying the following expression:

$$W = \nabla_W \mathfrak{L}_P - \mathrm{proj}_{\nabla_W \mathfrak{L}_A} \nabla_W \mathfrak{L}_P - \alpha \nabla_W \mathfrak{L}_A \tag{3.16}$$

where α is an hyperparameter that can be tuned by the user. The first term of the expression $\nabla_W \mathfrak{L}_P$ is the gradient that is usually applied to compute W, the second term $\mathrm{proj}_{\nabla_W \mathfrak{L}_A} \nabla_W \mathfrak{L}_P$ prevents $\hat{C}$ from helping the adversary A to lower its loss function, and the last term $\alpha \nabla_W \mathfrak{L}_A$ introduces a noise that increases the adversary's loss.

Disparate mistreatment remover

Disparate mistreatment remover, proposed by Zafar et al. [14], is an in-processing algorithm which goal is to train a fair model which satisfies the *separation* criterion (Equation 3.4) by adding a number of constraints to the optimisation function used to minimise the loss. In this paper, the authors use the term *disparate mistreatment* when referring to the *separation* fairness definition.

The algorithm proposed by Zafar et al. focuses on decision boundary-based classifiers such as logistic regression and support vector machines. Furthermore, the following definitions must be provided:

- $A \in \mathbb{R}^d$ is the set of non-sensitive attributes
- $Y \in \{-1,1\}$ refers to the binary class label, and $C \in \{-1,1\}$ is the predicted label
- θ describes the parameters to learn
- $\mathfrak{L}(\theta)$ denotes the convex loss function from where classifiers usually learn their decision boundary
- $d_\theta(A)$ refers to the signed distance from B to the decision boundary
- $f_\theta(A)$ is the classifier function. This function is equal to 1 if $d_\theta(A) \geq 0$, otherwise it is -1.
- $B \in \{0,1\}$ describes the sensitive attribute

In the paper, the fairness formulation used to describe the disparate mistreatment is the relaxed condition expressed in Equation 3.5, that is, the *equality of opportunity.* In this setting, such definition can be reformulated as

$$\mathbb{P}\{C \neq Y \mid B = 0,\ Y = -1\} = \mathbb{P}\{C \neq Y \mid B = 1,\ Y = -1\} \tag{3.17}$$

to denote the false positive rate. By incorporating this condition into the classifier formulation, the optimisation function can be written as:

$$\begin{aligned} \min \quad & \mathfrak{L}(\theta) \\ \text{subject to} \quad & \mathbb{P}\{C \neq Y \mid B = 0\} - \mathbb{P}\{C \neq Y \mid B = 1\} \leq \epsilon \\ & \mathbb{P}\{C \neq Y \mid B = 0\} - \mathbb{P}\{C \neq Y \mid B = 1\} \geq -\epsilon \end{aligned} \tag{3.18}$$

where $\epsilon \in \mathbb{R}^+$ is a tunable parameter that defines how fair the final classifier can be. The smaller ϵ is, the more fair the decision boundary will be. The additional constraints define a fair boundary that is used to limit the optimal decision boundary.

As the authors note, the condition expressed in Equation 3.17 is, in general, non convex, which makes the task of solving Equation 3.18 intractable. To overcome this issue, Zafar et al. propose to substitute Equation 3.17 with a tractable proxy which relaxes the constraints. The chosen proxy is the covariance between the sensitive attributes B and the signed distance between the feature vector A of misclassified users and the decision boundary of the classifier.

$$\begin{aligned} \operatorname{Cov}(B, g_\theta(Y, A)) &= \mathbb{E}[(B - \bar{B})(g_\theta(Y, A) - \bar{g}_\theta(Y, A))] \\ &\approx \frac{1}{N} \sum_{A,B,Y \in D} (B - \bar{B}) g_\theta(Y, A) \end{aligned} \tag{3.19}$$

where $g_\theta(Y, A) = \min(0, \frac{1-Y}{2} Y d_\theta(A))$. The term $\mathbb{E}[B - \bar{B}]\ \bar{g}_\theta(Y, A)$ is cancelled out since $\mathbb{E}[B - \bar{B}] = 0$. With this proxy, Equation 3.18 can be rewritten as follows:

$$\begin{aligned} \min \quad & \mathfrak{L}(\theta) \\ \text{subject to} \quad & \frac{1}{N} \sum_{A,B,Y \in D} (B - \bar{B}) g_\theta(Y, A) \leq c \\ & \frac{1}{N} \sum_{A,B,Y \in D} (B - \bar{B}) g_\theta(Y, A) \geq -c \end{aligned} \tag{3.20}$$

where $c \in \mathbb{R}^+$ is a covariance threshold that the user can set to control how adherent to the *separation* criterion the boundary should be.

3.2.3 Post-processing

Post-processing algorithms try to satisfy fairness constraints by slightly modifying the output of a model, without the need to change the training data and/or to retrain the model. These algorithms are usually used when the two previous approaches are not viable because the training data set or the machine learning algorithm are not accessible. Through these strategies, the idea is to correct the threshold used to select the proper output for each group in order to enforce some fairness criterion.

Reject option classification

Reject option classification is a post-processing technique proposed by Kamiran et al. [28] that gives favourable outcomes to unprivileged groups and unfavourable outcomes to privileged groups in a confidence region around the decision boundary with the highest uncertainty. The algorithm determines the fairness of the outcome by applying the *independence* definition (Equation 3.1).

Let A be the set of non-sensitive attributes and B the set of sensitive features. $C \in \{0,1\}$ describes the target label, where 0 is the negative outcome and 1 is the positive one. $D = \{A_i, B_i, C_i\}_{i=1}^{N}$ refers to a data set of N samples, $f : (A, B) \rightarrow \{0,1\}$ describes the classifier which output must be transformed according to the *independence* criterion, and $\mathbb{P}\{1 \mid A, B\}$ is the posterior probability computed by f for a given sample.

When the probability $\mathbb{P}\{1 \mid A, B\}$ is closer to 1 or 0 it means that the classifier has a higher degree of certainty that the record should be classified positively or negatively. As $\mathbb{P}\{1 \mid A, B\}$ gets closer to 0.5, then the classification outcome becomes more uncertain. In a situation like the one previously described, we can define a *critical region* such that $\max[\mathbb{P}\{1 \mid A, B\}, 1 - \mathbb{P}\{1 \mid A, B\}] \leq \theta$, where $0.5 \leq \theta \leq 1$ describes how large the critical region is. The closer θ is to 0.5, the smaller the critical region is. The instances that fall into this region are considered to be ambiguous and subject to biases, which justifies the introduction of what is called a *reject option*. This option is introduced to compensate for the biases of the model and requires that all instances belonging to the unprivileged group must be classified positively, and all instances belonging to the privileged group have to be labelled negatively. On the contrary, the predictions outside of the critical region are considered to be free from prejudice and are classified coherently with the model's output.

Equalized odds post-processing

Equalized odds post-processing is a post-processing technique proposed by Hardt et al. [24] which tries to satisfy the *separation* definition of fairness (in Hardt's work, this criterion is called *equalized odds*). Its task is to solve a linear program to find the probabilities with which to change the label predicted from a classifier's output score in order to optimise the equalized odds.

Before presenting the algorithm, the following definitions must be given:

- $A \in \{0, 1\}$ is a binary sensitive attribute, X the set of non-sensitive features, $Y \in \{0, 1\}$ the true target, and $\hat{Y} \in \{0, 1\}$ the label predicted by the classifier;
- $R(A, X) \in \mathbb{R}$ denotes the score function based on which the label $\hat{Y}$ is selected;
- $\tilde{Y}$ is a label derived from A and R in such a way that $\tilde{Y}$ is independent of X conditional on (R, A), that is, $\tilde{Y} \perp X \mid R, A$;

- $\mathfrak{L}(\hat{y}, y) \in \mathbb{R}$ is the loss function used to estimate the "goodness" of the classifier, and $\mathbb{E}\,\mathfrak{L}(\hat{y}, y)$ is the expected loss to be minimised;
- $\gamma_a(\hat{Y}) = (\mathbb{P}\{\hat{Y} = 1 \mid Y = 0, A = a\}, \mathbb{P}\{\hat{Y} = 1 \mid Y = 1, A = a\})$ is a pair composed by the false positive rate and true positive rate within the demographic satisfying $A = a$. For the *equalized odds* definition to be satisfied, $\gamma_0(\hat{Y}) = \gamma_1(\hat{Y})$. The *equality of opportunity* criterion (Equation 3.5) can also be satisfied. In this case, $\gamma_0(\hat{Y})$ and $\gamma_1(\hat{Y})$ only need to agree in the second component (therefore, the true positive rate is the same for both groups);
- $P_a(\hat{Y}) = \mathit{convhull}\{(0{,}0), \gamma_a(\hat{Y}), \gamma_a(1 - \hat{Y}), (1{,}1)\}$ is a two-dimensional convex polytope defined as a convex hull of four vertices. $\tilde{Y}$ can be derived if and only if $\gamma_a(\tilde{Y}) \in P_a(\hat{Y}), \forall a \in \{0{,}1\}$.

The linear optimisation problem can be formulated as follows:

$$\begin{aligned} \min_{\tilde{Y}} \quad & \mathbb{E}\,\mathfrak{L}(\tilde{Y}, Y) \\ \text{subject to} \quad & \gamma_0(\tilde{Y}) = \gamma_1(\tilde{Y}) \\ & \forall a \in \{0{,}1\} : \gamma_a(\tilde{Y}) \in P_a(\hat{Y}) \end{aligned} \tag{3.21}$$

The task performed by the *equalized odds post-processing* technique can be represented graphically as shown in Figure 3.2. If the ROC curve is printed independently for the instances with $A = 0$ and $A = 1$, the different behaviour of the model is clear. The optimal $\tilde{Y}$ that satisfies the equalized odds criterion (figure on the left) is found at the intersection of the two curves, while to satisfy the equality of opportunity definition (figure on the right) it is enough to constrain the true positive rates for the two classes to be the same. Thus, the selected points simply need to be on the same horizontal line.

3.2.4 How to choose a bias mitigation algorithm

In this section, a number of different algorithms for bias mitigation have been presented. This list is not intended to be exhaustive, but to provide an overview of some of the techniques that have been proposed by researchers in the last years. The number of algorithms that tries to find an optimal solution for the machine learning fairness problem grows every year, and no algorithm is clearly superior to the others. This is due to a multitude of reasons.

As previously described, the choice of the algorithm is constrained by the chosen definition of fairness, which may vary from case to case. Moreover, different criteria can't be pursued simultaneously and each algorithm mainly focuses on a single definition. For instance, the *reweighing* algorithm is based on the *independence* criterion, while the *disparate mistreatment remover* technique uses the *separation* definition.

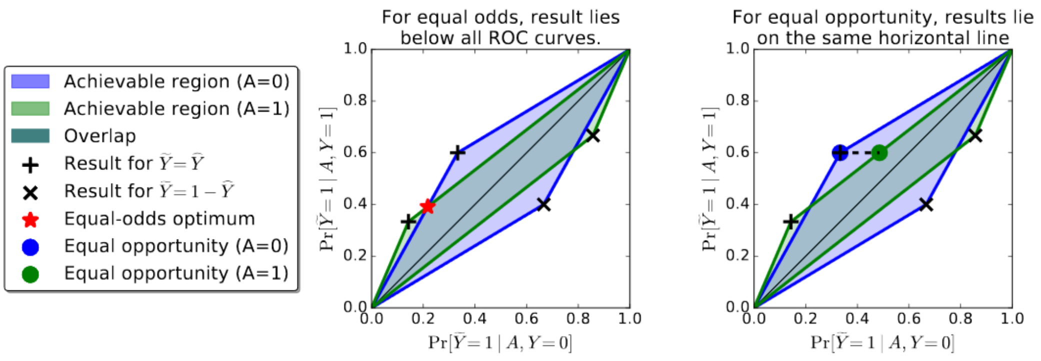


Figure 3.2: A graphical interpretation of the idea behind the equalized odds post-processing technique. On the left, the label that satisfies the equalized odds definition is selected. On the right, the chosen label is the one that satisfies the equality of opportunity criterion [24]

Another factor that influences the decision is the part of the machine learning pipeline in which the user is allowed to intervene. As a rule of thumb, the earlier the algorithms are applied, the most flexible and effective the intervention will be. If it is possible to alter the training data, then pre-processing techniques should be preferred. Otherwise, if the user is allowed to change the learning algorithm, but can't modify the training data, then he might adopt an in-processing technique. If none of these strategies is viable, the only possibility is to work on the posterior probability produced by the model through a post-processing algorithm.

Similarly, the selection of the algorithm depends on the requirements of the algorithm itself. For instance, the equalized odds post-processing technique, despite being a post-processing strategy, requires access to the sensitive feature in order to compute the right label. Other algorithms have some limitations in terms of the types of classifier they can be applied to. Some algorithms, such as the *reject option classification* one, are deterministic, while others have a randomised component (e.g. *disparate mistreatment remover*).

Finally, when multiple algorithms can be applied to the problem at hand, all of them should be tested, to see how they perform on the specific data set available for the task.

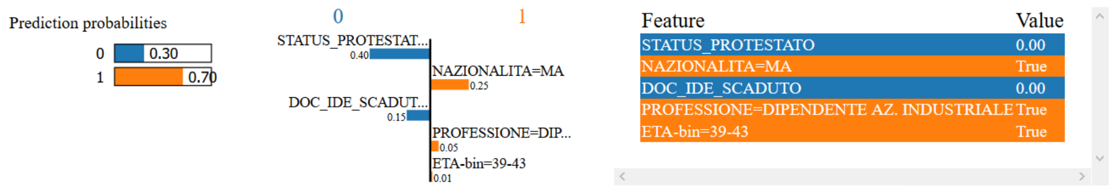
3.3 Tools and frameworks for ML fairness

The application of fairness criteria to machine learning models is a topic that has received a lot of attention in recent years, thanks to a greater awareness about the risks that unfair machine learning models might pose to specific social groups. Not only has this popularity led to a growth in the number of scientific publications related to fairness, but it also encouraged the introduction of a number of tools which goal is to monitor the behaviour of a model to alert the user in case of unfair treatment. Moreover, some of these solutions also provide an implementation for the algorithms previously described. This section provides an overview of the main tools available to audit a model for biases and mitigate the prejudice found in an artificial intelligence system.

3.3.1 Model interpretability as a bias detection tool

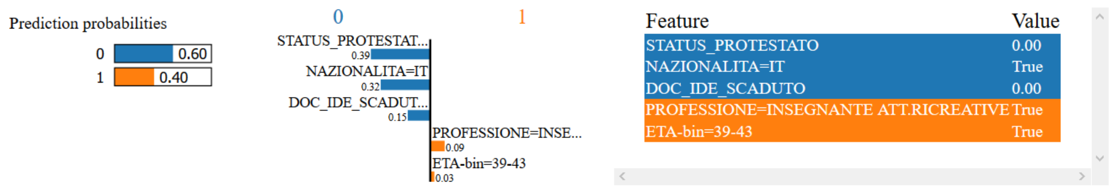
In Chapter 2, various solutions for interpreting a model's prediction and understanding its behaviour have been introduced. These techniques can be leveraged to provide insights on which features contributed the most to a prediction. If a sensitive feature had a huge impact on the output of the model, then interpretability algorithms should be able to expose the prejudice and present it to the user. Additionally, these algorithms can be used to check the absence of prejudice after a bias mitigation algorithm has been applied to the training data or to the learning algorithm. If the bias mitigation technique has been successful at removing the prejudice, then the weight given to the sensitive feature should be close to zero.

An example of the application of an interpretability algorithm as a tool for detecting unfair behaviours is presented in Figure 3.3. The field of application is a loan approval process in which, based on the customer information, a manager must decide if a loan application should be accepted. This problem will be further investigated in the next chapter, but a small description is provided to understand the context of the example. The positive outcome (the application is accepted) is 0, while 1 means that the application should be rejected. On the left side of the image, the probabilities for each class label are shown. In the right box, the details about the customer are listed, and the middle box depicts LIME's score estimation for the given record. The feature of interest is "NAZIONALITA", which refers to the nationality of the customer. Figure 3.3a shows how the nationality of the first customer (MA - Morocco) negatively influenced the model's prediction, ultimately leading to a negative prediction. On the contrary, the classifier assigned a positive label to the second customer even though the two records are very similar (Figure 3.3b). LIME's scores provide an estimate of the influence that each feature had on the final output, and show how the factor that has influenced the change in the prediction the most is the fact that the customer is italian (IT - Italy). Thus, by comparing LIME's output for both customers, it has been possible to expose the bias of the model towards some nationalities.



Feature	Value
STATUS_PROTESTATO	0.00
NAZIONALITA=MA	True
DOC_IDE_SCADUTO	0.00
PROFESSIONE=DIPENDENTE AZ. INDUSTRIALE	True
ETA-bin=39-43	True

(a) LIME's output for a moroccan customer



Feature	Value
STATUS_PROTESTATO	0.00
NAZIONALITA=IT	True
DOC_IDE_SCADUTO	0.00
PROFESSIONE=INSEGNANTE ATT.RICREATIVE	True
ETA-bin=39-43	True

(b) LIME's output for an italian customer

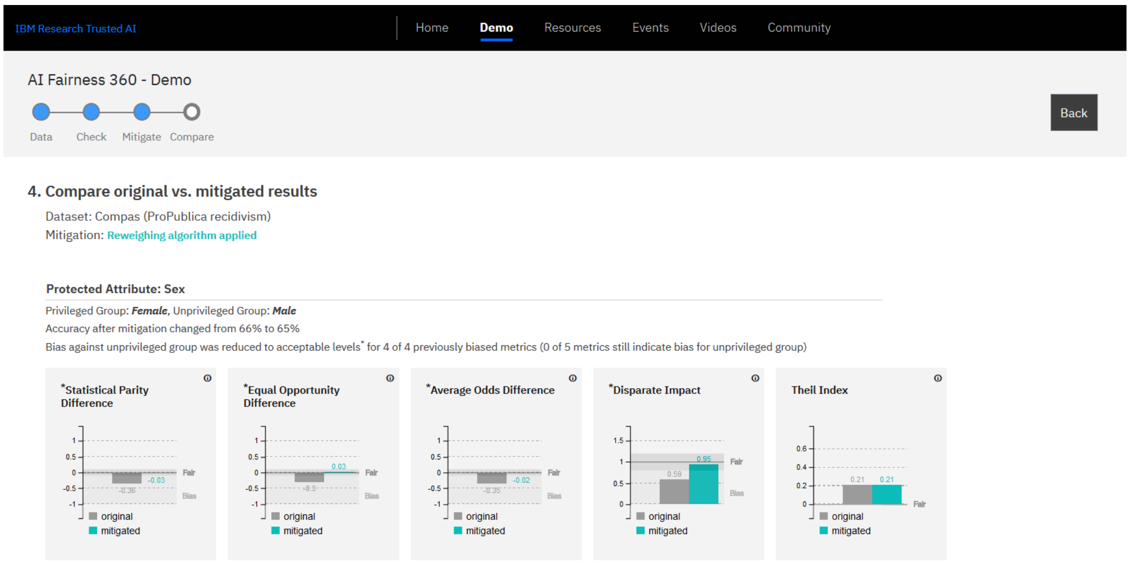
Figure 3.3: A comparison between two LIME's outputs which reveals a bias in the model.

3.3.2 IBM AI Fairness 360

At the time of writing, IBM AI Fairness 360 (AIF360) is perhaps the biggest open source toolkit available for machine learning fairness. Its goal is to "examine, report, and mitigate discrimination and bias in machine learning models throughout the AI application lifecycle" [8]. It is an extensible framework capable of unifying most of the metrics and algorithms presented in this chapter. It also includes a bias explanation feature that gives further insights about the computed metrics. The toolkit is available as a Python library [9], and its architecture is further analysed in the accompanying paper [12].

AIF360 provides a wide variety of metrics to support the process of inspecting the model for unfair behaviour. The featured measurement functions ranges from traditional performance evaluation metrics such as *precision* and *recall*, to bias detection metrics such as *disparate impact* (Equation 3.2) and *equal opportunity difference* (Equation 3.5). It also provides a number of functions for computing the distance between different samples.

The toolkit introduced by IBM provides an implementation of most of the algorithms presented in Section 3.2. The framework created by the authors, based on a custom Dataset class, resembles the *fit-transform-predict* paradigm typical of other popular libraries such as *scikit-learn*, which simplifies its adoption by the data science community. Moreover, the standardised approach to bias mitigation allows the user to test several algorithms in a limited amount of time, which is crucial since the choice of a bias mitigation algorithm is mostly based on a "trial and error" strategy. AIF360 has been designed to be easily extensible by external contributors, which provides a way for researchers to quickly implement their own bias mitigation algorithms by using AIF360 as their base framework.



4. Compare original vs. mitigated results

Dataset: Compas (ProPublica recidivism)
Mitigation: Reweighing algorithm applied

Protected Attribute: Sex

Privileged Group: *Female*, Unprivileged Group: *Male*
Accuracy after mitigation changed from 66% to 65%
Bias against unprivileged group was reduced to acceptable levels* for 4 of 4 previously biased metrics (0 of 5 metrics still indicate bias for unprivileged group)

Figure 3.4: The interactive experience provided by IBM AI Fairness 360. It offers an introduction to the topics of algorithmic fairness and bias mitigation.

Along with the toolkit, the researchers behind AIF360 also provided an interactive web experience which introduces the user to the notions of algorithmic fairness, bias detection and bias mitigation. This interactive experience provides a simple interface through which the user can select between three biased data sets, compare a selection of metrics to evaluate the prejudice of the trained model, and choose a bias mitigation algorithm to obtain a fair classifier. Finally, it allows to compare the performance and bias metrics of the original model with the performance and bias metrics of the model obtained by applying the selected bias mitigation technique. Figure 3.4 shows an example of the described interface. The dataset selected for this test is the one that was originally used by ProPublica during its investigation of the COMPAS algorithm [7], the strategy chosen to reduce the prejudice is pre-processing, and the algorithm used to mitigate the bias is the *reweighing* algorithm [27]. The interface also compares the metrics of the original model with the metrics of the mitigated model (grey and blue, respectively). Among the presented metrics, *disparate impact* (Equation 3.2) and *statistical parity difference* (Equation 3.3) refers to the *independence* criterion, *equal opportunity difference* uses the *equality of opportunity* definition of fairness (Equation 3.5), and *average odds difference* is a way to measure *separation* (Equation 3.4).

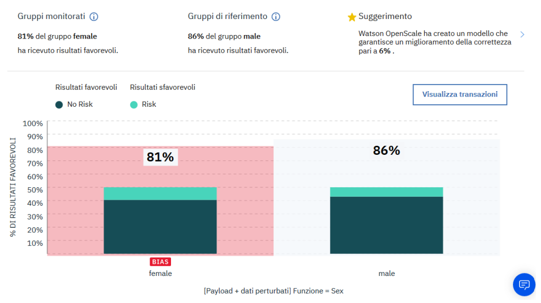
3.3.3 IBM Watson OpenScale

In Section 2.4, the IBM Watson OpenScale is already explained in most of its features. In this section, the overview of the tool is completed by considering its bias detection and bias mitigation capabilities. 5. Application to the loan approval use case

Along with the performance monitoring, OpenScale also gives the possibility to setup a monitor to track the fairness of the model at hand. This setup requires the user to inform OpenScale about which are the sensitive features that must be tracked. In addition, the user is asked to manually provide the distinction between the privileged and unprivileged group. In one of the most recent updates of the tool, the possibility to automatically recognise which features might be subject to biases has been introduced, but this feature is really just a comparison between the actual feature name and a dictionary of potentially biased feature names.

After the monitor has been properly set up, OpenScale uses the outputs from the previously requested predictions to compute the rate of favourable outcomes for the privileged and unprivileged groups. The presence of biases in the model is estimated based on the *disparate impact* metric (Equation 3.2), which is compared to a user defined threshold to decide whether to throw an alert. As for the performance metrics tracking, OpenScale allows the user to group the predictions differently based on the selected time frame, and to check for each time frame if the fairness condition is satisfied (Figure 3.5a).

Finally, the tool allows the user, given a prediction which is suspected to be biased, to generate an alternative prediction in which the distortion produced by the prejudice has been removed. OpenScale also provides an interface that shows the difference in performance between the original model and the model obtained through the bias mitigation procedure (Figure 3.5b). The approach chosen by OpenScale to generate its bias-free predictions is not specified by IBM.


Gruppi monitorati

81% del gruppo **female**
ha ricevuto risultati favorevoli.

Gruppi di riferimento

86% del gruppo **male**
ha ricevuto risultati favorevoli.

Suggerimento

Watson OpenScale ha creato un modello che garantisce un miglioramento della correttezza pari a **6%** .

(a) Openscale's comparison between the privileged and unprivileged groups.

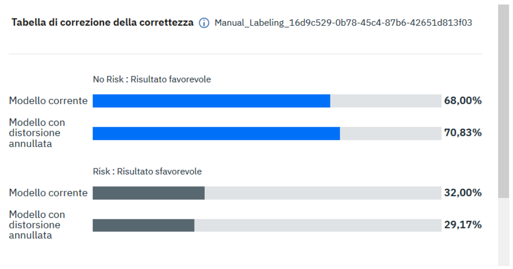


Tabella di correzione della correttezza Manual_Labeling_16d9c529-0b78-45c4-87b6-42651d813f03

	No Risk : Risultato favorevole
Modello corrente	68,00%
Modello con distorsione annullata	70,83%

	Risk : Risultato sfavorevole
Modello corrente	32,00%
Modello con distorsione annullata	29,17%

(b) OpenScale's comparison between the performances of the original model and of the unbiased model

Figure 3.5: IBM OpenScale's fairness monitoring feature.

Chapter 4

Application to the loan approval use case

So far, the focus of this work has been on analysing and presenting the main theories that have been proposed in the field of eXplainable machine learning and machine learning fairness. Along with the theoretical discussion, a selection of libraries and tools that implements the described concepts has been provided.

In this chapter, an application of the theories and frameworks previously presented is discussed. The concepts of interpretability and fairness are applied to the use case of a loan approval process, and their results are made accessible through an internet-based software application, which will be later referred to as the *model management* application. This tool allows the user to manage different models and datasets, monitor each model's performance, and generate predictions alongside an explanation of the factors that mostly influenced the model's decision. Furthermore, the application provides an interface through which the user can analyse the managed datasets and models and recognise whether some of them are affected by prejudice. If a model's behaviour is classified as biased, then the tool also offers the feature to train an unbiased version of the model, which can be used to obtain predictions in which the prejudice's influence has been removed.

The chapter begins by presenting the use case in more depth. In Section 4.2, the main tools and libraries used during the application development are described, while in Section 4.3 the architecture of the software is presented. Section 4.4 provides a description of all the features available in the final application. Finally, in Section 4.5 the model management application is compared to some of the solutions presented in previous chapters.

4.1 Context description

Money lending is one of the core businesses of every bank and it is one of the most profitable activities conducted by these institutions. By granting loans to their customers, banks earn a profit in the form of interest rates that customers accept to pay in exchange for receiving the money. On the other hand, these activities represent one of the greatest risks that a bank has to undertake. In order to asses and limit the risk of lending money to a customer who won't be able to pay, banks must collect and analyse a lot of information about the individual who applied for the loan. This data is needed to estimate the customer's ability to pay his debt. The pieces of information needed include the personal data about the customer who made the application, his financial situation and credit history, as well as the entity of the specific request made by the individual. The amount of data that is required for the analysis, combined with the abundance of past information about granted loans, make this one of the most interesting application fields for machine learning techniques.

Although loan approval might benefit for the introduction of machine learning techniques to support the decision process, there are two factors that limits their application possibilities in this field: first, loan approval processes are high risk activities which requires the manager to understand the motivation behind a machine learning model's decision; second, the decision has a considerable impact on the future of the customer who applied for the loan, and he must be provided with explanations for why his application was rejected. Furthermore, such decision must be devoid of biases to ensure that individuals from different backgrounds are treated fairly.

In the specific use case presented in this work, the original goal was to analyse a dataset of previously granted loans to extract some insights useful to predict which customers were likely to default on their loans. The information produced by this analysis would then be extended to those customer who had applied for a loan, to estimate their future ability to live up to their promise to pay. The analysis' results were made accessible through a internet-based application which provided the skeleton for the machine learning models' management platform. The interface was then extended to leverage some of the model explanation methods presented in Chapter 2 and some of the bias monitoring and mitigation techniques describes in Chapter 3. Finally, a number of features to allow a user to intuitively load new datasets, train new models and monitor the performance of existing models were added to the final tool.

4.2 Programming languages, frameworks, and tools adopted

4.2.1 Backend

The backend of the model management application was built using Python 3.6 as main programming language. To simplify the development phase, the database chosen for data storage is SQLLite, which is accessed through a library called SQLAlchemy. The application also leverage various libraries that provides extended functionalities and data structures. This section lists the most important packages exploited by the tool.

Pandas

Pandas is an open source Python package which provides a number of data structures and data analysis tools to efficiently manipulate the data at hand. The main data structures offered by Pandas are the Series structure, to handle one-dimensional data, and the DataFrame structure, which is used to store and manipulate two-dimensional data. The package also provides various utilities aimed at simplifying the Input/Output operations from/to several formats (CSV, XLSX, etc.).

Numpy

Numpy is a Python package for scientific computing. It provides a container to store and manipulate multi-dimensional generic objects, with the possibility to define arbitrary data-types. Furthermore, it includes a variety of mathematical functions designed to efficiently work with the provided data structures.

Scikit-learn

Scikit-learn is an open source machine learning library for Python, which provides an implementation of several supervised and unsupervised machine learning algorithms such as decision trees, SVMs, and k-Means. Through its interface, data scientists can test different techniques with minimal changes in the code. This library also provides a set of preprocessing features, as well as a number of functions to perform model validation and to compare a model with another.

Flask

Flask is a lightweight web application microframework for Python. The term “micro” refers to the fact that the framework does not include a database abstraction layer, form validation, or upload handling. Instead, Flask provides a minimalistic framework that can be extended through several wrapper packages that leverage existing libraries and work seamlessly with the other extensions supported by Flask. For instance, through its extension Flask-RESTful, the framework can be extended to support the development of REST APIs. At its core, Flask acts as a wrapper for other two packages:

Werkzeug This library includes a number of utilities for WSGI applications. WSGI (Web Server Gateway Interface) is a calling convention for the Python language which specifies how a web server communicates with web applications, and how to chain several web applications together to process one request. Among its features, Werkzeug provides a request and response object to interact with other web applications, as well as a routing system for matching URLs to endpoints. It also includes a WSGI server to develop applications locally.

Jinja Jinja is a templating engine for Python applications. It allows to create templates, such as HTML page templates, which include special placeholders used to write code using a Python-like syntax. These templates are rendered by the engine to dynamically build the final page.

Hence, Flask provides a framework for reacting to external requests coming to a specified URL, map it to a endpoint using Werkzeug utilities, and select and customize an view to return as a response through the Jinja's template engine.

SQLAlchemy

SQLAlchemy is a Python package that includes a suite of enterprise-level persistence patterns used to provide access to a database storage in an efficient and high-performing way. It implements a SQL abstraction toolkit which provides a number of features to abstract from the specific DBMS implementation and to generate custom SQL queries through Python functions. The package also includes an optional Object Relational Mapper (ORM) extension, which extends the core functionalities with features like the Unit Of Work system, used to rearrange SQL operations for optimised performance, and the caching of previous queries.

Model interpretability and machine learning fairness packages

Several libraries have been used to extend the tool with model explanation and bias mitigation capabilities. The application exploits the LIME [6] and SHAP [4] libraries provided by the algorithms' authors to estimate the feature scores or contributions for a given prediction. Furthermore, the Anchor algorithm has been tested by leveraging the original Anchor library provided by Ribeiro et al. Finally, the application uses the bias mitigation algorithms' implementations provided by IBM's open source framework AI Fairness 360 [9] to train fair models from biased ones.

IBM Watson's APIs

Even though the tool was created using exclusively open source technologies, a variant that exploits the IBM Watson suite services for model interpretability has been implemented. This version of the application interacts with the Watson Machine Learning service through its REST API to request the model's predictions. Furthermore, it communicates with the Watson OpenScale service to obtain explanations for the requested predictions. The API provided by OpenScale is also used for sending feedback data to the IBM's tool.

4.2.2 Frontend

The graphical user interface of the application, accessible through a web browser, is based on HTML pages that exploit CSS to enhance the pages' layout. The interface also has a number of interactive features created using JavaScript. As for the backend, the frontend relies on external libraries to implement some functionalities and create a better user experience.

JQuery

JQuery is one of the most popular JavaScript frameworks. It provides an API to simplify a variety of tasks, such as HTML document manipulation and traversal, event handling, and AJAX. Part of its popularity is due to its consistency in the way it behaves across several browser, which simplifies the work of the programmer who wants to implement an interactive functionality that runs independently of the user's browser. Another advantage of JQuery is its syntax, which allows the programmer to write fewer lines of codes with respect to vanilla JavaScript, which improves the readability and maintainability of the software.

BootStrap

Bootstrap is a HTML, CSS and JavaScript framework for creating web sites and web applications. It provides a layout system (called Grid system) which allow to build responsive web pages with minimal efforts. It follows a mobile-first approach, which means that its components are optimised to work with mobile devices and are scaled to fit a different layout when needed. Another feature of Bootstrap is the set of pre-made, easily customisable components that are made available by the frameworks. The components provided by Bootstrap are commonly used web elements such as alert boxes, styled buttons, custom form layouts and navigation bars. By combining Bootstrap with JQuery, other frequently used components such as modal boxes and slideshows become accessible.

Chart.js and ProgressBar.js

Along with Bootstrap, two other libraries are used to provide better looking plots and other graphical elements. *Chart.js* is a JavaScript library for creating interactive and responsive plots. It exploits the HTML5 canvas element, which guarantees optimal rendering performance across all modern browsers, and it offers a variety of charts such as horizontal and vertical bar charts, line charts, and bubble charts. The library also provide a wide set of feature to customise every detail of the plot. *ProgressBar.js*, on the other hand, is a simple, lightweight JavaScript library which offers a set of functions to create responsive progress bars with different shapes. The library offers a number of pre-made shapes such as lines and circles, but also provide the user the ability to create its custom-made progress bar shape.

4.2.3 Support tools

For an application to fit into a company's workflow and coexist with other programs, a number of additional tools are needed. These pieces of software are used to automate tasks such as testing or deployment, track an application's versions, and share someone's work with the rest of the team. In the case of the application described in this chapter, the tools that supported the software development phases that followed coding are described below.

Version control systems and GitLab

Version control systems (VCS) are a type of software that allows the programmer to keep track of the changes that occur to a project's artefacts over time by storing different versions of each file. The assets that can be put under version control include source code, text documents, and images, but it can be any file that makes up the final project. By storing different versions of the same file, the programmer can revert back to a past version when new updates inadvertently introduce errors in the code or in other documents.

A particular type of VCS, called *distributed version control system* (DVCS), provides further advantages by allowing users to share code and documents among team members. Through this approach, the files under version control are stored in one or more centralised servers, and the server's content is mirrored by all clients that are collaborating to the project. This way, every client has its own copy of the repository and its history. DVCSs also provide other features:

Traceability VCSs track the changes made by any individual to a project's files, and enable users to trace every change back to the person that introduced it and the time when the change occurred.

Access control Different users can be granted different visibility and/or editing privileges.

Branches A project can simultaneously evolve in different directions. VCSs track all the evolutions of the project (called branches), and allow to merge different branches when needed (e.g. when a feature is completed and can be safely added to the rest of the application).

Conflict resolution When several people concurrently work on the same file, conflicts between the different versions of the document may arise. Version control provides a set of utilities to manage the conflicts and merge the coexisting versions of the file into a single version.

GitLab is a repository hosting and management platform built on top of Git, one of the most popular DVCSs available today. It offers all the common features of a version control system, while also integrating in the same product several DevOps utilities such as issue trackers, continuous integration, and continuous delivery. Furthermore, it provides greater security and a finer grained access control compared to its competitors. These characteristics make this tool one of the preferred VCSs at enterprise-level.

Jenkins

Continuous integration (CI) is a development practice that requires developers to integrate their work in a shared repository on a daily basis, with the goal of minimising the amount of effort and the complexity required to integrate together pieces of software built by different people. Whenever a programmer shares his code with its peers, the code should be automatically built and tested. This way, issues can be discovered early in the process, which reduces the time spent in debugging the application at integration time.

Jenkins is an open source automation server which can be used to orchestrate a chain of actions that are automatically executed when a developer shares its work on a repository, thus simplifying the adoption of CI. The tasks that can be automated by Jenkins include building, testing, delivering and deployment. The tool offers a graphical user interface through which the developer can monitor the status of each automated task and can be notified whenever a problem occurs. Jenkins can also be connected to repository management platforms like GitLab. This way, whenever a developer pushes a new version of the code to the shared repository, Jenkins' execution is triggered and the new code is automatically integrated with the existing application.

Docker

When working on complex systems in distributed environments, correctly setting up the configuration of each software component and its dependencies is a difficult task. This is especially true when the same configuration has to be run on several machines, each of which might have its own requirements. To overcome these limitations, *Docker* containers can be used.

A container is an encapsulated environment which is used as a wrapper for the application and its dependencies. Based on software-level virtualisation (which guarantees better performances with respect to the traditional virtual machines), a container can be configured to include all the system tools and libraries required by the application. Each container is isolated from the others, which allows for several containers to be running at the same time.

The container configuration is described by what is called a *Docker image*. This image includes all the code, dependencies, and system tools required to run the application. Therefore, Docker makes the task of replicating the setup of an application on different machines as simple as sharing a container image, which will be executed by each machine using Docker.

4.3 Architecture design

4.3.1 High-level overview

Section 4.2 described which libraries and frameworks are exploited by the model management application. Among the frameworks presented in that section, the one that influences the application's structure the most is Flask. While the package does not dictates any particular architectural choice, its developers has provided some design recommendations for a better organisation of the project. The model management application follows the suggested approach, which requires to group all components into a single package. Figure 4.1 shows the project's high level structure, and its main elements are described below:

- `/run.py`: This file is invoked to start up the development server. It retrieves a copy of the application from the package and runs it.
- `/Dockerfile`: A text document used by Docker to assemble an image for the application.
- `/requirements.txt`: This file lists all the dependencies of the application.
- `/application/`: The package that contains the application.
- `/application/__init__.py`: This file initializes the various components of the application (ORM, security configuration, etc.).
- `/application/templates/`: The `templates` directory stores the html templates that are used by Jinja2 to generate the pages to return to the user.

- `/application/static/`: This directory includes the style sheets, JavaScript files, and images used by the html templates

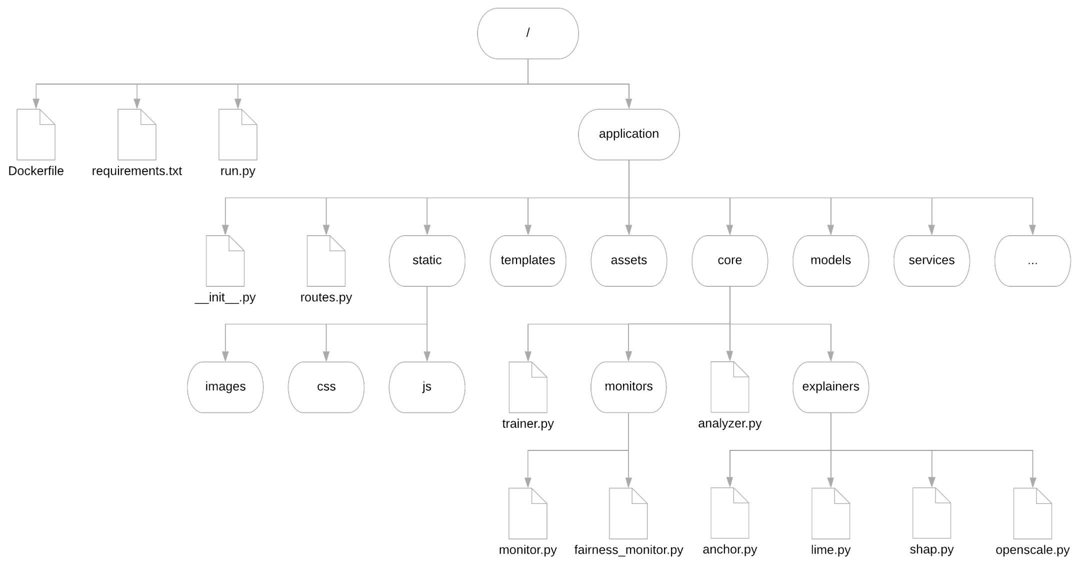


Figure 4.1: Part of the project's directory structure

Another pattern followed during the design of the architecture is the classic web-based application's division into presentation, service, and domain layers. In the case of the described application:

- `/application/routes.py` is the component that deals with the requests that come to a specific URL, forwards the requests to the service layer, and selects a specific view based on the output of the service layer (most of the times, the view is derived from one of the HTML templates available);
- `/application/services/` represents the directory that contains the files which compose the service layer. This layer's job is to describe the business logic of the application. It receives the requests from the `/application/routes.py` and, for the most complex tasks, it delegates the work to the classes provided in `/application/core/`;
- `/application/models/` is the set of classes that describes the data domain layer, that is, the entities that are handled by the application. These objects can be stored in the database and retrieved from it.

One last directory that needs to be discussed is the `/application/core/` folder. This folder is where most of the value adding functionalities of the application are implemented. It contains all the utility methods and classes used to train and monitor the machine learning models. It also includes the classes that generate the model's explanations and the ones that apply the bias mitigation algorithms.

4.3.2 Model's description

The domain layer of the model management application is the foundation of the whole software program. It describes the main entities that are handled by the application and how they interact with each other. In this project, these classes are mainly used for storing and retrieving information from the database, whereas most of the computation is based on Pandas' dataframes or Numpy's arrays.

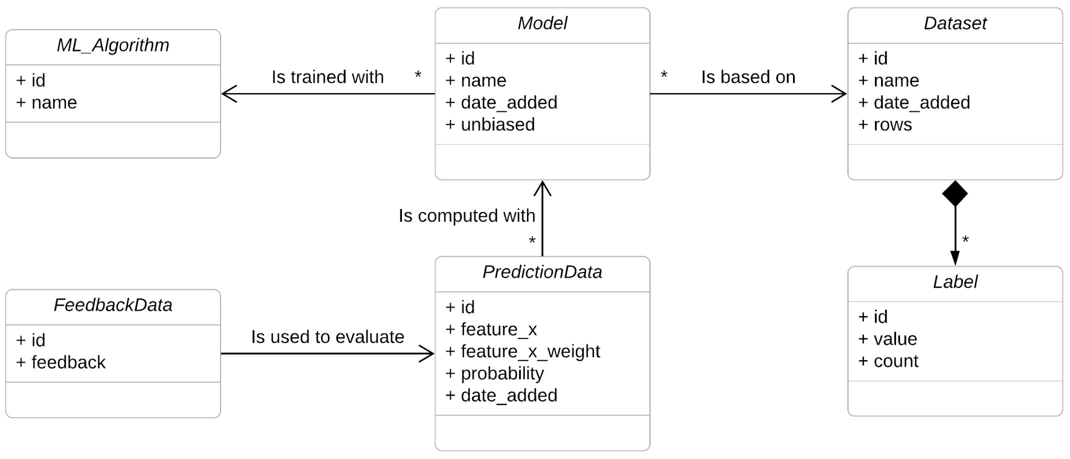


Figure 4.2: Class diagram of the model's structure

Figure 4.2 shows the class diagram for the application's domain. A description of the classes is also provided below:

Dataset It describes the dataset used to train the model. It is characterised by an identifier, the number of rows of the dataset, and a name, which is used as a user friendly identifier of the dataset. The dataset ID is also used to retrieve the dataset's content from the local storage.

Label The Label class represents the values that can be assigned to the labels of a dataset. Each possible combination $\{dataset\_id, label\_value\}$ is described by an instance of this class. To characterise each label, the information about its value and its number of occurrences in the associated dataset is added.

Model It denotes the model trained using a specific dataset. Each model is described by an identifier, a descriptive name, and by the date it was added to the system. The *unbiased* boolean attribute is used to specify if the model was obtained by applying a bias mitigation algorithm to another model.

ML_Algorithm This entity is used to describe the algorithm with which the model's is trained with. It is used to provide some additional information to the user and during the bias mitigation process to train an unbiased version of an unfair model.

PredictionData It is used to represent the prediction computed by a model for a given instance provided by the user. The outcome of the prediction is stored in the form of the probability returned by the model. The entity also includes, for each attribute of the instance being predicted, its feature value and the related score or weight obtained using a model-agnostic interpretability algorithm like LIME or SHAP.

FeedbackData It describes a feedback provided for a given prediction. The feedback is represented as a boolean attribute, where *true* means that the prediction aligns with the expectation of the user or with the real outcome.

4.3.3 Basic application's workflow

To allow a better understanding the most relevant core features of the application, and provide a context for their usage, the basic workflow of the program is presented now. The graphical user interface that supports this process is described in detail in Section 4.4.

Dataset preprocessing and setup

Before loading a dataset into the application, a preliminary data cleaning step is required in order for the program's methods to work properly. This procedure is required to get rid of missing data in the dataset (either by removing the entire row or by replacing it with a reasonable value), standardise equivalent values, and group the values of high cardinality features into discrete bins. After a dataset has been properly prepared, it can be loaded in the application and used to train one or more machine learning's models. In the current iteration of the project, the column's names and types are hard coded, but in the next versions of the software these pieces of information will be provided by the user during the setup phase of the dataset.

Training the model

Once the dataset is available for selection, the training phase can begin. The system trains several models using different machine learning algorithms. The trained models are presented to the user along with some metrics to evaluate the performance of each model. The user can then compare the different models and select the best one, which is permanently stored in the database.

Requesting a prediction and collecting feedback

The user can now select a model to predict the probability that a customer will be able to repay a loan based on his personal data. At the same time, the system will compute the explanation using one of the algorithms presented in Section 2.3 and will store the prediction's details in the database. Once the prediction has been computed, the user can provide a feedback which is also stored in the database.

Performance monitoring

The feedback collected from the user is used to compute some basic statistics about the performances of the managed models, such as accuracy, recall, and precision.

Fairness monitoring and bias mitigation

The information derived from the model's predictions is used to monitor how a sensitive attribute influences the model's behaviour. If a model's decisions are considered to be unfair, then an unbiased version of the model can be trained using one of the bias mitigation algorithms presented in Section 3.2. Additionally, the application checks if a prejudice can be traced back to the original dataset label's distribution.

4.3.4 A standardised explanation framework

The model management application provides several ways through which the user can generate an explanation for a given prediction. The backend approach followed to provide these functionalities is based on a small framework that offers a standardised access to all these methods. The framework's main classes are described below and are represented by the class diagram depicted in Figure 4.3.

Configuration class

The *Configuration* class performs all the preprocessing tasks required to use the interpretability algorithms. It requires a DataFrame object, from which most of the information required to initialise the interpretability algorithms is generated. The provided DataFrame is used to extract the categorical and numerical features to be evaluated, convert the categorical values into numerical data (while storing the mapping between the two types to allow the reverse process), and apply the one hot encoding procedure to the categorical attributes. The *Configuration* class is

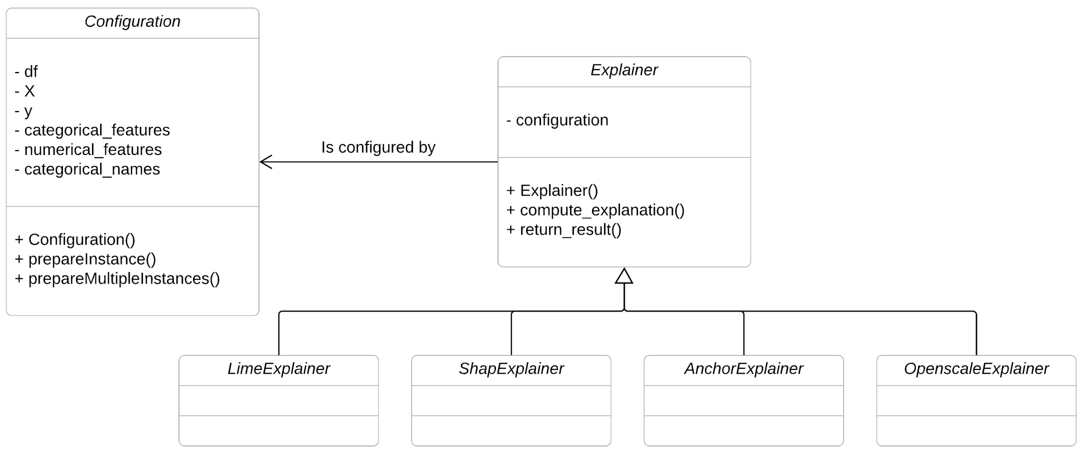


Figure 4.3: Class diagram of the model explanation framework

also used to prepare the instance to be explained for the application of the interpretability algorithms by exploiting the data extracted during the Configuration's initialisation phase.

Explainer interface

All the "explainers" available shares the same structure, which provides a common interface that standardise the access to the different interpretability algorithms and allows to easily switch from one explainer to the other or to use different explainers at the same time. The explainer classes are initialised using the Configuration parameter previously described, and their explanations are generated through the `compute_explanation()` method. The produced explanation can be returned using different formats (list, dictionaries, etc.) depending on the specific implementation.

The Explainer implementations

There are four main implementations of the "interface" presented above. Each of these classes leverage some other library to offer its services, but the work required to initialise and format the requests to these external packages is hidden from the developer and is handled by the Explainer and Configuration objects.

LimeExplainer This class produces a score for each feature based on the LIME algorithm. It exploits the LIME implementation proposed by Ribeiro et al. [6], but normalises the score produced by the algorithm in such a way that, by summing up all the scores, the total value amounts to 100. This operation is motivated by the fact that the average user might misinterpret LIME's results and might find the its scores to be a bit confusing.

ShapExplainer Analogously, the *ShapExplainer* class is a wrapper around the SHAP implementation provided by Lundberg and Lee [4]. SHAP's output explanation is then processed by the *ShapExplainer* class to produce a result similar to the one provided by LIME.

AnchorExplainer Another algorithm for which an explainer is provided is the Anchor algorithm by Ribeiro et al. [35]. As with the previous explainers, the *AnchorExplainer* class leverages the implementation proposed by the authors and provides a standardised access to the external library.

OpenscaleExplainer Finally, an explainer adaptation to connect to IBM's Watson Machine Learning and Openscale APIs is provided. Due to its peculiar nature, this class works a bit differently with respect to the previous explainers and requires some additional setup. Nonetheless, its usage is similar to the one of the other explainers and most of the issues involved with making the API requests are hidden from the developer.

Explanation framework's workflow

The standard process, which is also shown in Figure 4.4, is described as follows: first, a *Configuration* object is created by the service layer using the original training dataset (or an equivalent dataset). Second, the generated instance is used to initialise an *Explainer* object. In the third step, the service layer requires the new *Explainer* object to provide an explanation for a given instance. Since the actual interpretability algorithm requires the instance to be provided in a specific format, the *Explainer* exploits its internal *Configuration* to prepare the given instance accordingly. Once the *Explainer*'s method has finished its computation, the explanation can be retrieved by the service layer.

4.3.5 The bias detection and mitigation processes

One of the most important features of the model management application is the capability to inspect the dataset label distribution and the model's behaviour for biases. In the current version of the software, only a sensitive attribute (i.e. the *nationality* of the customer) is tracked. In future iterations, the goal is to extend this process to all other attributes, in order to directly suggest to the user which features might carry some prejudice without any hard coded limitation.

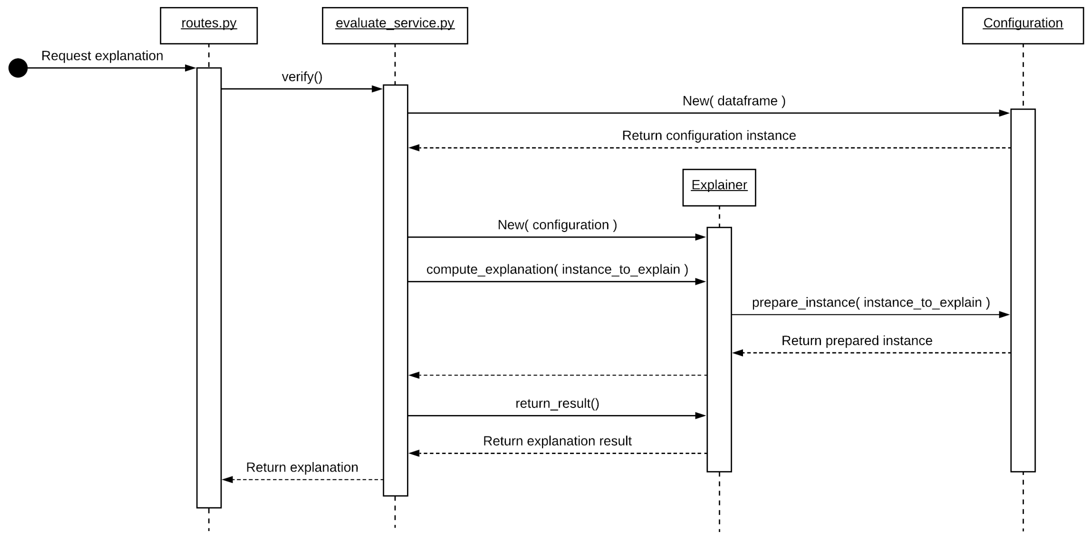


Figure 4.4: Simplified sequence diagram of the model explanation framework

Determining the bias presence

The model management application can inspect both the original dataset and model's behaviour to determine the presence of prejudice. In the case of the dataset, its rows are used to identify if a bias can be traced back to the original data (this feature is meant for diagnostic purposes only); instead, the system uses the predictions made by the model being analysed to inspect the model's behaviour. If a bias is detected from the model's predictions, then an unbiased version of the model can be trained and stored for future predictions. Apart from these differences, the algorithm used to identify the presence of prejudice is the same for both dataset and model.

The *bias detection* base algorithm is described next. The algorithm's input is provided in the form of a pandas' DataFrame object D, which is then inspected for biases using the *disparate impact* metric (Equation 3.2). The choice of using the *independence* criterion is based on the fact that this definition has legal support. Moreover, the selected criterion is supported by a higher number of bias mitigation algorithms. As part of the process, the DataFrame's rows $d \in D$ are grouped based on the sensitive attribute value s (e.g. the nationality). These groups are later referred to as the *sensitive groups* $g \in G$. For each sensitive group g, the ratio of positive outcomes is computed, and G is split into one or more *privilege classes* $C \in \mathfrak{C}$, where $C = \{g_i \mid g_i \in G,\ 0 < i \leq |G|\}$, based on two factors: each privilege class C must represent at least 5% of the entire population of D, and the disparate impact between C and any other sensitive group g (or vice versa) must be lower than

0.8. The first constraint is applied to guarantee that each privilege class contains a statistically relevant number of instances, while the 0.8 threshold has been selected to comply with the *80%-rule* [13]. The output of the algorithm is a set of privilege classes $\mathfrak{C}$, where the class with the highest rate of positive outcomes is considered to be the *privileged* class, and the other classes are referred to as the *unprivileged* classes. Thus, if $\mathfrak{C}$ in composed by at least two privilege classes, then the dataset or model being evaluated is considered to be biased. An outline of the process is presented in Algorithm 1.

Algorithm 1 Outline of the bias detection algorithm

```
function ComputePrivilegeClasses($D$)
    $C = \emptyset$;
    $\mathfrak{C} = \emptyset$;
    $t \leftarrow len(D)$;
    $G \leftarrow$ Select $*$ From $D$ GroupBy $s$;
    for all $g \in G$ do
        if $(len(C) \geq t)$ and $(disparate\_impact(C,\ g) < 0.8)$ then
            $\mathfrak{C} \leftarrow \mathfrak{C} \cup \{C\}$;
            $C \leftarrow \emptyset$;
        end if
        $C \leftarrow C \cup g$;
    end for
    $\mathfrak{C} \leftarrow \mathfrak{C} \cup \{C\}$;
    return $\mathfrak{C}$;
end function
```

Training an unbiased model

If a model's behaviour is considered to be unfair, the model management system provides the user the possibility to train an unbiased version of the same model. For this purpose, Algorithm 1 is used to split the model into two classes: the *privileged class*, which is the set of the sensitive feature values with the highest ratio of positive outcomes, and the *unprivileged class*, which contains the remaining values. It must be noted that there may be values of the sensitive feature for which no prediction is available yet. Based on the criterion described above, these values will be assigned to the unprivileged class. The rationale behind this choice is that, in a situation in which the system has no information about how a model perceives a specific value, by assigning it to the unprivileged class the system is guaranteed to not exacerbate pre-existing unknown prejudices. Once the division of the feature values between privileged and unprivileged classes has been determined, the model management application exploits the *IBM AIF360* framework to apply a bias mitigation algorithm. The specific algorithm selected for this task is the *Reweighing* algorithm presented in Section 3.2. There are several reasons why this algorithm was chosen: first, the system has access to the dataset used for training the inspected model, which means that a pre-processing strategy like the Reweighing algorithm, which is likely to provide better results, can be applied; second, the Reweighing algorithm bases its decision on the *independence* criterion, which is the same fairness definition used to perform the distinction between privileged and unprivileged group; third, the selected algorithm's output is a set of weights, which is easier to interpret with respect to other techniques' outputs. Once the new set of weights has been determined, the unbiased version of the inspected model can be trained using the same machine learning algorithm used for the original model. The new model is then stored in the database and is available for performing new predictions.

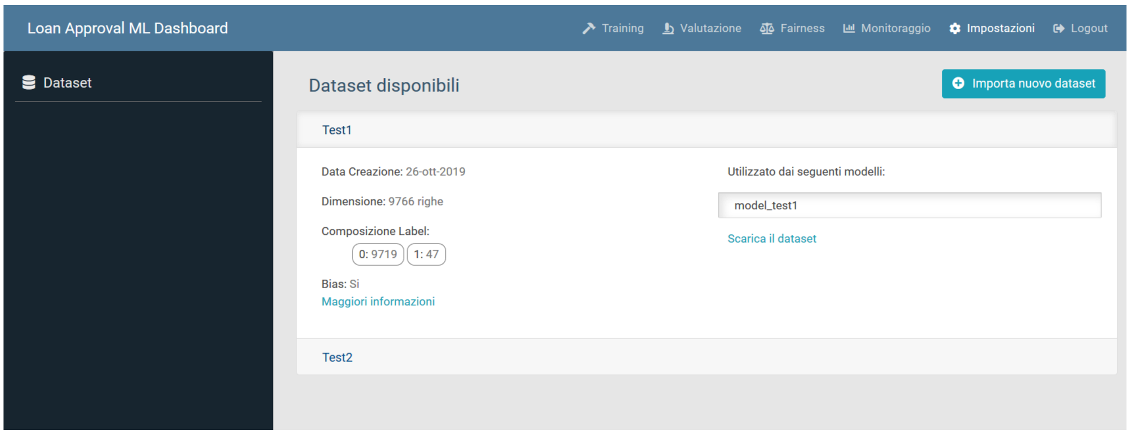
4.4 The model management application's interface

Section 4.3 focused on the most important components of the architecture of the model management application. It provided an overview of the whole project's structure, as well as some insights on how the most relevant features of the application have been designed and implemented. This section's objective instead is to present the views that compose the interface of the software program and to describe how the different views are structured. The focus here is not on presenting the technical implementation of the interface, which is rather straightforward once the backend correctly provides the desired information. Instead, this section's intent is to provide an overview of the application from the average user's perspective.

4.4.1 Dataset and model setup

Once the user accesses the model management application, he must provide a dataset from which one or more machine learning models will be trained. Figure 4.5 depicts the view that is presented to the user. By clicking on the blue button on the top right corner of the page, the user is presented with a modal he can use to provide a name for his dataset and the dataset itself. Once the dataset has been loaded into the system, its associated tab is shown on the interface, along with some diagnostic information, such as the size of the dataset and if the system detected the possible presence of biases in the dataset. The system also offers the possibility to download the dataset csv file.

After the dataset has been loaded, in another view the user can select the provided dataset and train a new model by clicking on the "train" button. The system tests several machine learning models, and the results are presented to the



Dataset disponibili

Test1

Data Creazione: 26-ott-2019

Dimensione: 9766 righe

Composizione Label: 0: 9719 1: 47

Bias: Si

Utilizzato dai seguenti modelli: model_test1

Test2

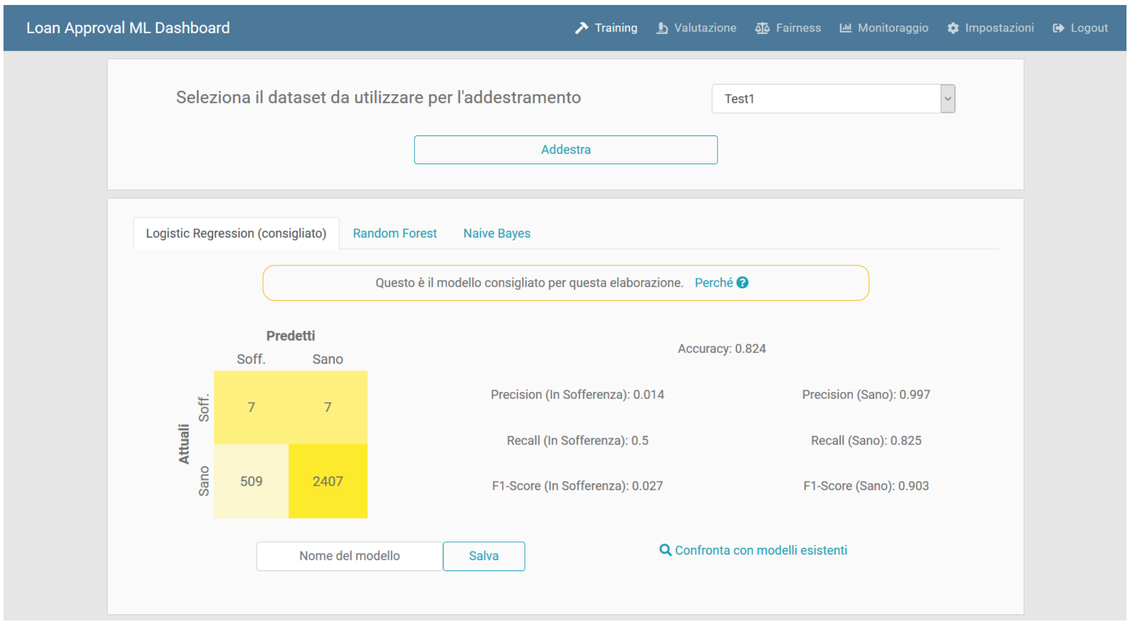
Figure 4.5: The dataset setup view of the the model management application

user in different tabs. The system ranks the trained models based on the f1-score metric and initially suggests to the user the model with the highest f1-score. The system also allows the user to compare the performance of the selected model with the performance of the models already in the system. Finally, the user can store the currently selected model by providing a name for the new model and by clicking on the “store” button on the bottom left of the screen. Figure 4.6 presents a screenshot of the described view.

4.4.2 Requesting a prediction and the associated explanation

To request a prediction, the user must fill the provided form (Figure 4.7), which contains the attributes selected for this test, as well as a drop-down list from which the model selected for the prediction can be chosen. By clicking on the “verify” button, the prediction result and explanation are generated. As Figure 4.8 shows, the final output, along with its probability, is presented in the top-left box, while in the right box an explanation is provided. In the example, the algorithm used for generating the explanation is SHAP. Finally, the user can provide a feedback by clicking on one of the two buttons contained in the bottom-left box.

One thing that has been omitted from the previous description is the checkbox next to the “nationality” field. When a user trains a new model, a second model without the sensitive attribute is trained, and this “unbiased” version of the model can be used for the prediction by deselecting the “nationality” checkbox. This fairness approach follows the “fairness through unawareness” criterion presented in Section 3.1, and was the original naive strategy followed to solve the fairness issue. This notion of fairness was later substituted with the concept of *independence*, and the checkbox feature, while deprecated, is left for flexibility and testing purposes.



Seleziona il dataset da utilizzare per l'addestramento: Test1

Addestra

Logistic Regression (consigliato) | Random Forest | Naive Bayes

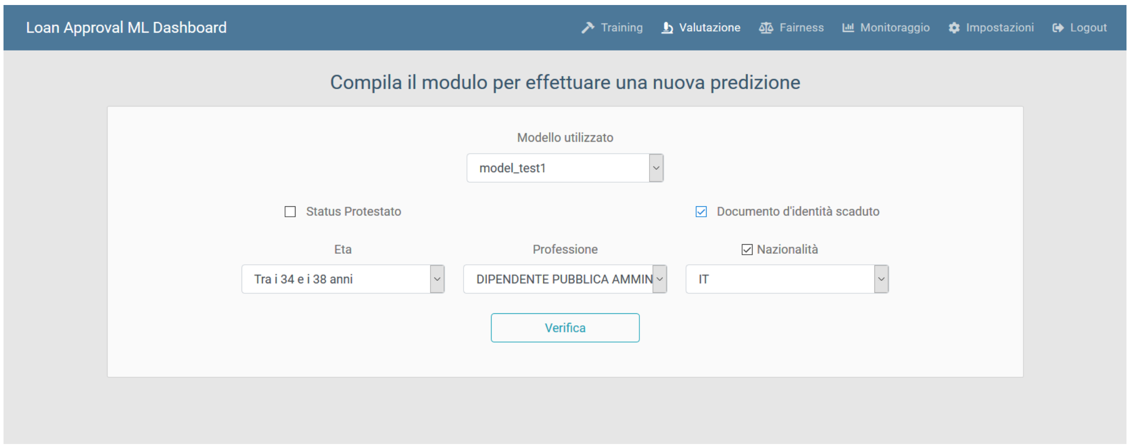
Questo è il modello consigliato per questa elaborazione. Perché

		Predetti: Soff.	Predetti: Sano
Attuali	Soff.	7	7
Attuali	Sano	509	2407

Accuracy: 0.824

Precision (In Sofferenza): 0.014	Precision (Sano): 0.997
Recall (In Sofferenza): 0.5	Recall (Sano): 0.825
F1-Score (In Sofferenza): 0.027	F1-Score (Sano): 0.903

Figure 4.6: The model training interface provided by the model management application



Compila il modulo per effettuare una nuova predizione

Modello utilizzato: model_test1

☐ Status Protestato

☑ Documento d'identità scaduto

Eta: Tra i 34 e i 38 anni

Professione: DIPENDENTE PUBBLICA AMMIN

☑ Nazionalità: IT

Figure 4.7: The model management application's form to request a new prediction

4.4.3 Bias detection and mitigation

By navigating to the "fairness" section, the user is presented with the privilege class division for the most recently uploaded dataset (Figure 4.9). The "nationality" partitioning is obtained by applying the procedure described in Algorithm 1. From the first tab in the navigation menu on the left, the user can choose the dataset or

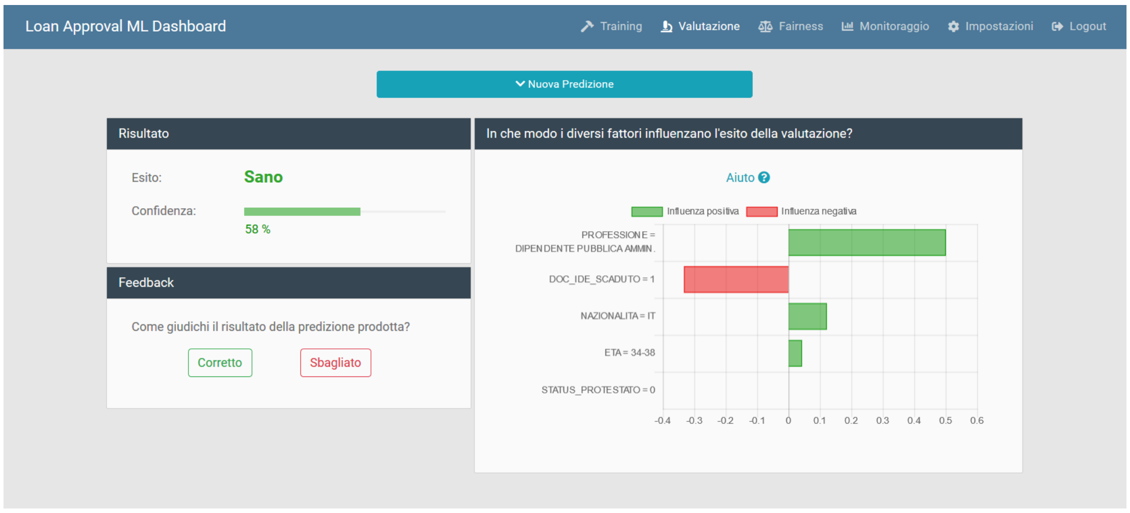
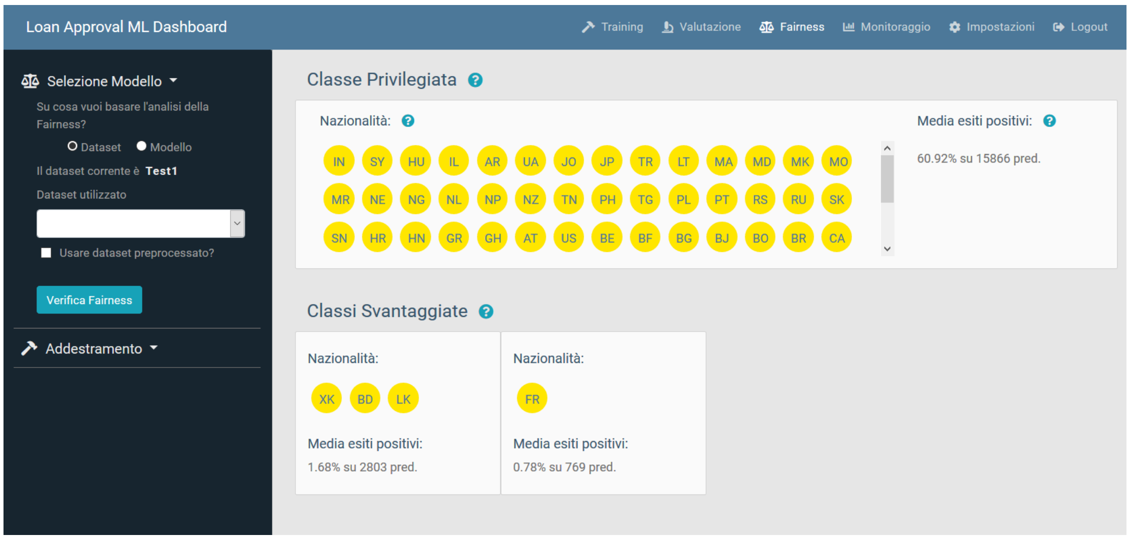


Figure 4.8: An example of a prediction generated by the model management application

model to analyse. Moreover, by selecting the “training” tab, the user can request the system to train a new version of the model by exploiting the *Reweighing* bias mitigation algorithm. Once the training is completed, the model is permanently stored in the system, along with its previous version, and can be selected in the “evaluate” section of the interface. A comparison between the prediction generated by the biased version of a given model and the prediction generated for the same instance by the unbiased version of the model is provided in Figure 4.10.

4.5 Comparison with existing solutions

The model management application presented in this chapter has been developed with the dual goal of providing a management platform to support a loan approval process, and laying out the foundation for a more general model management suite. The latter intent is made evident by the generic description that was provided for most of the structures and features presented. Of course, the current version of the application has been developed around the loan approval use case, and some of its parts have been hard coded to fit the case. However, there are already several features that are mostly independent from the specific context. For instance, the standardised explanation framework presented in Section 4.3.4 is not bound to any dataset schema. This section’s focus is on comparing the features already available in the tool (considered in their most general form) with the functionalities provided by other solutions presented in this work (Section 2.4 and Section 3.3).


Loan Approval ML Dashboard

Training | Valutazione | Fairness | Monitoraggio | Impostazioni | Logout

Selezione Modello

Su cosa vuoi basare l'analisi della Fairness?

○ Dataset ● Modello

Il dataset corrente è **Test1**

Dataset utilizzato

☐ Usare dataset preprocessato?

Verifica Fairness

Addestramento

Classe Privilegiata

Nazionalità: IN SY HU IL AR UA JO JP TR LT MA MD MK MO MR NE NG NL NP NZ TN PH TG PL PT RS RU SK SN HR HN GR GH AT US BE BF BG BJ BO BR CA

Media esiti positivi: 60.92% su 15866 pred.

Classi Svantaggiate

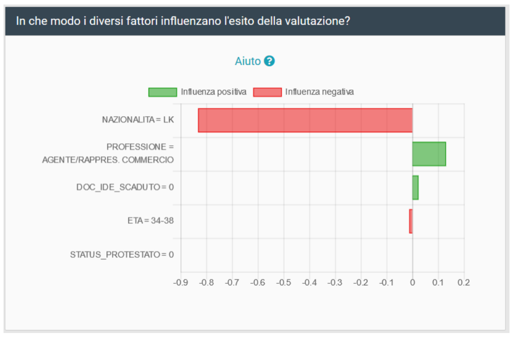
Nazionalità: XK BD LK

Media esiti positivi: 1.68% su 2803 pred.

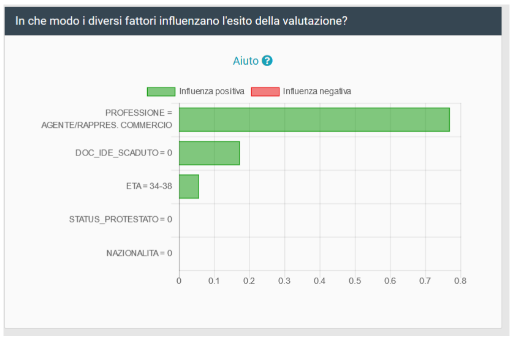
Nazionalità: FR

Media esiti positivi: 0.78% su 769 pred.

Figure 4.9: Bias detection view of the model management application



(a)



(b)

Figure 4.10: A comparison between the prediction's explanations generated by the unfair (left) and fair (right) models for the same instance.

Model Management Application vs OpenScale

The most obvious comparison is between the model management application and IBM's Watson OpenScale service. Both tools are meant to provide a suite to monitor the performance of the available models, interpret their results, and react to unfair behaviours of the models. While OpenScale does not directly provide the capability to manage the user's datasets and train custom models, these tasks are achieved by integrating OpenScale with the rest of the Watson environment. Indeed, the suite to which OpenScale belongs includes other tools that assist the user during all the steps related to developing his custom machine learning model, in a way that its much more flexible with respect to the one detailed by the model management application. Moreover, OpenScale is able to provide its management services for different types of classifiers, namely image, multi-class, and binary classifiers.

On the other hand, the model management application has the advantage of providing a variety of interpretability algorithms (LIME, SHAP, Anchor), most of which are interchangeable with minimal effort. The tool also introduces a different approach to detecting the bias. One of OpenScale's greatest limitations is that the privileged and unprivileged groups must be selected in advance when setting up the fairness monitor. This operation may become cumbersome as the cardinality of the sensitivity attribute grows. Moreover, the user may not be aware of which value belongs to which group. On the contrary, the approach detailed in Algorithm 1 doesn't require any setup by the user.

There's also a number of factors that might limit the application of a commercial tool such as OpenScale. The first, and in many cases the most important argument against OpenScale, is the expensiveness of the tool. Other issues are related to the lack of control over what happens inside OpenScale and the requirement to have the training data stored on IBM's cloud. Finally, some limitations are related to OpenScale being a recent addition to IBM's services. The tool's first commercial version was released around April 2019, and a new beta version is underway. This means that there are several issues that puts a practical limitation to the afore mentioned flexibility. For instance, at the time when the tool was tested the machine learning package *scikit-learn* was not fully supported.

Comparison with other tools

Though the only direct "competitor" of the model management application is Open-Scale, there are other tools which presents some similarities with the application described in this chapter. The open source libraries Skater and ELI5 presented in Section 2.3 both offer a number of utilities to provide model interpretability. While the model management application doesn't introduce any original implementation of the interpretability algorithms provided, its approach based on the combination of the *Explainer* and *Configuration* classes offers an easier and more flexible access to a wider array of interpretability algorithms.

The Google What If Tool (WIT) is another service which provides further insights on how a model behaves. The main difference with other solutions like Open-Scale or the model management application presented here is the approach chosen to interpret the model's behaviour. The visual approach followed by WIT leverages the predictions obtained from a test set to provide customisable graphs which offer a better understanding of the relationship between different attributes and the predicted label. The tool is also useful for comparing different instances and for observing how each prediction changes as its attributes values are modified. While WIT is great for a data scientist who is trying to better understand his model, most users want a straightforward explanation for why a certain result was obtained. On the other hand, the model management application was designed with the goal of providing interpretations which can be effortlessly understood even by non-expert.

Chapter 5

Conclusion and future work

In recent years, machine learning applications have become increasingly ubiquitous. The pattern recognition capabilities provided by these strategies has proved to be one of the best ways to deal with the growing amount of information that is available for analysis. This fact led machine learning techniques to be adopted in increasingly high risk fields, like medicine or finance. Moreover, as machine learning started to be applied in loan approval or school admission processes, the impact these strategies has on people's lives became evident. Hence, this surging popularity must be backed up by an evolution of the tools and techniques that allow people to monitor, interpret and react to the machine learning models' behaviour.

This work presented an analysis of the most popular solutions that have been introduced to meet the need for trustworthy artificial intelligent systems, which leverage machine learning strategies to provide support to decision processes. The problem was framed into two separate topics: the demand for interpretable machine learning models, and the need for these models to treat different groups of people equally. The proposed techniques and tools were then applied to a loan approval process, with the goal of developing an easy to use platform for model management capable of supporting the whole life cycle of a machine learning model. The resulting model management application provides the user the tools to train its own machine learning models, monitor their performance, examine their predictions, and inspect the model's decisions for unfair behaviour. These features were designed with simplicity in mind. Indeed, the main target of the presented system are non-experts who don't have the technical background or the time required to understand the technicalities of machine learning, or to analyse and interpret complex graphical representations, but still need to trust the output or their system and react accordingly if the model doesn't behave as expected.

The presented model management application, while capable of supporting the manager or data scientist from the training of a machine learning model to its dismissal, is an early stage tool with plenty of room for improvement. From an architectural point of view, the fact that back-end and front-end are strictly dependent one from the other is a huge limitation. To overcome this issue, in future iterations the tool might be rewritten according to the REST paradigm to provide better independence between back-end and front-end. Defining a view-independent API will also help the process of generalising the current architecture of the back-end. Another drawback of the current implementation is indeed its dependency from the specific use case. By generalising the architecture, the structures presented in Chapter 4 may be leveraged to provide the foundation for an open source tool capable of offering a standardised access to several model interpretability algorithms as well as bias detection functionalities. The interface, in turn, might be improved in terms of capabilities and flexibility by introducing a front-end web framework like Angular. In terms of features provided, the model management application can be extended by providing a finer grained control over the datasets and models managed by the user, and by introducing the support for a greater number of machine learning and model interpretability algorithms.

The steps described here provide a roadmap for evolving the model management application. Of course, future versions will also be affected by feedback from the customer. The tool's main goal is to simplify the management of machine learning models and build the user's trust over machine learning techniques. Hence, interactions with the final user are crucial to understand which are the main limitations of the system, and will provide further insights for extending the presented application.

Bibliography

[1] URL: https://www.kaggle.com/c/titanic/overview.

[2] URL: https://pair-code.github.io/what-if-tool/.

[3] URL: https://pair-code.github.io/facets/.

[4] URL: https://github.com/slundberg/shap.

[5] URL: https://github.com/oracle/Skater.

[6] URL: https://github.com/marcotcr/lime.

[7] URL: https://www.propublica.org/article/machine-bias-risk-assessments-in-criminal-sentencing.

[8] URL: https://aif360.mybluemix.net/.

[9] URL: https://github.com/IBM/AIF360.

[10] URL: https://github.com/oracle/Skater.

[11] Solon Barocas, Moritz Hardt, and Arvind Narayanan. *Fairness and Machine Learning*. http://www.fairmlbook.org. fairmlbook.org, 2019.

[12] Rachel K. E. Bellamy et al. "AI Fairness 360: An Extensible Toolkit for Detecting, Understanding, and Mitigating Unwanted Algorithmic Bias". In: *CoRR* abs/1810.01943 (2018). arXiv: 1810.01943. URL: http://arxiv.org/abs/1810.01943.

[13] D. Biddle. *Adverse Impact and Test Validation: A Practitioner's Guide to Valid and Defensible Employment Testing*. Gower, 2005. ISBN: 9780566086540. URL: https://books.google.it/books?id=q7zZ8h5X3nQC.

[14] Muhammad Bilal Zafar et al. "Fairness Beyond Disparate Treatment &amp; Disparate Impact: Learning Classification without Disparate Mistreatment". In: *arXiv e-prints*, arXiv:1610.08452 (2016), arXiv:1610.08452. arXiv: 1610.08452 [stat.ML].

[15] L. Breiman et al. *Classification and Regression Trees*. Monterey, CA: Wadsworth and Brooks, 1984.

[16] Leo Breiman. “Random Forests”. In: *Mach. Learn.* 45.1 (Oct. 2001), pp. 5–32. ISSN: 0885-6125. DOI: 10.1023/A:1010933404324. URL: https://doi.org/10.1023/A:1010933404324.

[17] Flavio Calmon et al. “Optimized Pre-Processing for Discrimination Prevention”. In: *Advances in Neural Information Processing Systems 30*. Ed. by I. Guyon et al. Curran Associates, Inc., 2017, pp. 3992–4001. URL: http://papers.nips.cc/paper/6988-optimized-pre-processing-for-discrimination-prevention.pdf.

[18] Mark W. Craven and Jude W. Shavlik. “Extracting Tree-structured Representations of Trained Networks”. In: *Proceedings of the 8th International Conference on Neural Information Processing Systems*. NIPS’95. Denver, Colorado: MIT Press, 1995, pp. 24–30. URL: http://dl.acm.org/citation.cfm?id=2998828.2998832.

[19] Amit Dhurandhar et al. “Explanations based on the Missing: Towards Contrastive Explanations with Pertinent Negatives”. In: *CoRR* abs/1802.07623 (2018). arXiv: 1802.07623. URL: http://arxiv.org/abs/1802.07623.

[20] Bradley Efron and Robert J. Tibshirani. *An Introduction to the Bootstrap*. Monographs on Statistics and Applied Probability 57. Boca Raton, Florida, USA: Chapman & Hall/CRC, 1993.

[21] Michael Feldman et al. “Certifying and removing disparate impact”. In: *arXiv e-prints*, arXiv:1412.3756 (2014), arXiv:1412.3756. arXiv: 1412.3756 [stat.ML].

[22] Aaron Fisher, Cynthia Rudin, and Francesca Dominici. *All Models are Wrong, but Many are Useful: Learning a Variable’s Importance by Studying an Entire Class of Prediction Models Simultaneously*. 2018. arXiv: 1801.01489 [stat.ME].

[23] Sorelle A. Friedler, Carlos Scheidegger, and Suresh Venkatasubramanian. “On the (im)possibility of fairness”. In: *CoRR* abs/1609.07236 (2016). arXiv: 1609.07236. URL: http://arxiv.org/abs/1609.07236.

[24] Moritz Hardt, Eric Price, and Nathan Srebro. “Equality of Opportunity in Supervised Learning”. In: *CoRR* abs/1610.02413 (2016). arXiv: 1610.02413. URL: http://arxiv.org/abs/1610.02413.

[25] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning: data mining, inference and prediction*. 2nd ed. Springer, 2009. URL: http://www-stat.stanford.edu/~tibs/ElemStatLearn/.

[26] High-Level Expert Group on AI. *Ethics guidelines for trustworthy AI*. eng. Report. Brussels: European Commission, Apr. 2019. URL: https://ec.europa.eu/digital-single-market/en/news/ethics-guidelines-trustworthy-ai.

[27] Faisal Kamiran and Toon Calders. “Data preprocessing techniques for classification without discrimination”. In: *Knowledge and Information Systems* 33.1 (2012), pp. 1–33. ISSN: 0219-3116. DOI: 10.1007/s10115-011-0463-8. URL: https://doi.org/10.1007/s10115-011-0463-8.

[28] Faisal Kamiran, Asim Karim, and Xiangliang Zhang. “Decision Theory for Discrimination-Aware Classification”. In: *Proceedings of the 2012 IEEE 12th International Conference on Data Mining*. ICDM ’12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 924–929. ISBN: 978-0-7695-4905-7. DOI: 10.1109/ICDM.2012.45. URL: http://dx.doi.org/10.1109/ICDM.2012.45.

[29] Emilie Kaufmann and Shivaram Kalyanakrishnan. “Information Complexity in Bandit Subset Selection”. In: *Proceedings of the 26th Annual Conference on Learning Theory*. Ed. by Shai Shalev-Shwartz and Ingo Steinwart. Vol. 30. Proceedings of Machine Learning Research. Princeton, NJ, USA: PMLR, 2013, pp. 228–251. URL: http://proceedings.mlr.press/v30/Kaufmann13.html.

[30] Scott M Lundberg and Su-In Lee. “A Unified Approach to Interpreting Model Predictions”. In: *Advances in Neural Information Processing Systems 30*. Ed. by I. Guyon et al. Curran Associates, Inc., 2017, pp. 4765–4774. URL: http://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions.pdf.

[31] Scott M Lundberg et al. “Explainable AI for Trees: From Local Explanations to Global Understanding”. In: *arXiv preprint arXiv:1905.04610* (2019).

[32] Christoph Molnar. *Interpretable Machine Learning. A Guide for Making Black Box Models Explainable*. https://christophm.github.io/interpretable-ml-book/. 2019.

[33] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993. ISBN: 1-55860-238-0.

[34] David Reinsel, John Gantz, and John Rydning. *DataAge 2025 - The digitization of the world*. IDC, 2018.

[35] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. “Anchors: High-Precision Model-Agnostic Explanations”. In: *AAAI Conference on Artificial Intelligence (AAAI)*. 2018.

[36] Marco Túlio Ribeiro, Sameer Singh, and Carlos Guestrin. “"Why Should I Trust You?": Explaining the Predictions of Any Classifier”. In: *CoRR* abs/1602.04938 (2016). arXiv: 1602.04938. URL: http://arxiv.org/abs/1602.04938.

[37] Lloyd S. Shapley. *A Value for n-Person Games*. 1953, pp. 307–317.

[38] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. “Learning Important Features Through Propagating Activation Differences”. In: *CoRR* abs/1704.02685 (2017). arXiv: 1704.02685. URL: http://arxiv.org/abs/1704.02685.

[39] M. Stone. “Cross-Validatory Choice and Assessment of Statistical Predictions”. In: *Journal of the Royal Statistical Society. Series B (Methodological)* 36.2 (1974), pp. 111–147. ISSN: 00359246. URL: http://www.jstor.org/stable/2984809.

[40] Erik Štrumbelj and Igor Kononenko. “Explaining prediction models and individual predictions with feature contributions”. In: *Knowledge and Information Systems* 41.3 (2014), pp. 647–665. ISSN: 0219-3116. DOI: 10.1007/s10115-013-0679-x. URL: https://doi.org/10.1007/s10115-013-0679-x.

[41] Samuel Yeom and Michael Carl Tschantz. “Discriminative but Not Discriminatory: A Comparison of Fairness Definitions under Different Worldviews”. In: *CoRR* abs/1808.08619 (2018). arXiv: 1808.08619. URL: http://arxiv.org/abs/1808.08619.

[42] Brian Hu Zhang, Blake Lemoine, and Margaret Mitchell. “Mitigating Unwanted Biases with Adversarial Learning”. In: *CoRR* abs/1801.07593 (2018). arXiv: 1801.07593. URL: http://arxiv.org/abs/1801.07593.