KU LEUVEN

AI for Business

From data to decisions

Wouter Verbeke
Bart Baesens
Johannes De Smedt
Jochen De Weerdt
Hans Weytjens

January 23, 2025

Part 2: Predictive Analytics

CHAPTER 1

Explainable Artificial Intelligence (XAI)

Bart Baesens, Wouter Verbeke, Johannes De Smedt, Jochen De Weerdt, Hans Weytjens

Corresponding author: Bart.Baesens@kuleuven.be

1.1 CHAPTER OBJECTIVES

At the end of this chapter, you should be able to:

- identify different types of XAI techniques.
- understand basic visualization techniques such as cluster plots, Sankey diagrams and traffic lights.
- measure and visualize feature importance using permutation-based feature importance, partial dependence plots and individual conditional expectation (ICE) plots.
- represent regression-based models using a scorecard approach.
- use meta decision trees to represent black box models.
- understand how LIME, anchors, Shapley values, and counterfactuals work.
- understand how nomograms, Grad-CAM, and decision tables can be used for XAI.

1.2 INTRODUCTION

The goal of Explainable Artificial Intelligence (XAI) is to make AI systems transparent, interpretable, and understandable to humans [Molnar, 2022]. As AI applications continue to grow in healthcare, business, and law, understanding the ”why” behind AI decisions becomes essential for trust, accountability, and fairness. The need for XAI is further amplified by regulations such as the EU AI Act proposed by the EU Parliament and Council, the US AI Risk Management Framework as put forward by the National Institute of Standards and Technology (NIST) and the Provisions on the Management of Deep Synthesis Technologies as introduced by Cyberspace Administration of China (CAC). All these regulations share a focus on transparency, accountability, and human control and hence strongly encourage the adoption of explainable methods in AI development.

XAI bridges the gap between complex machine learning models and human comprehension by providing insights into the models' decision-making processes. However, it is important to be aware that the level of required model interpretation highly depends upon the AI application and business user. White box i.e.

interpretable models are especially important in settings such as credit risk modeling and healthcare where insight into the dynamics of a model's decision-making process is essential for justification or remedial action purposes. On the other hand, AI applications such as response modeling, recommender systems, ad targeting or gaming can be perfectly supported with black box AI techniques without any additional explanatory facilities.

A distinction can be made between different types of XAI techniques. Model-agnostic approaches (e.g., LIME, Anchors, Shapley values, see below) offer flexibility by working across different AI techniques without being tied to a specific model formulation or architecture. These methods prioritize generalizability, making them ideal for comparing and interpreting different models. On the other hand, model-specific approaches leverage the unique properties of a particular AI technique, often providing more precise or tailored insights but at the cost of general applicability. Popular examples are p-values which are typically only available for regression-based techniques, feature importance measures provided by random forests or the support vectors of an SVM.

Another distinction relates to the granularity of the explanations provided. Local explanation methods provide detailed insights into individual predictions at the observation level, making them particularly valuable for applications requiring case-by-case analysis. In contrast, global explanation methods aim to explain the overarching behavior of a model, offering a broader understanding of its decision-making patterns and biases. In this chapter, we further elaborate on this and discuss key concepts, techniques and challenges in XAI.

1.3 BASIC VISUALIZATION TECHNIQUES

To facilitate model interpretation, one commonly resorts to good visualization methods aiming to reduce cognitive overload and thus complexity by having users interact with data and/or analytical models using visual tools. The aim is to help AI model developers, data scientists, and business users to explore and better understand data and AI models. Remember the famous saying: "A picture is worth a thousand words"!

In Figure 1.1 you can see an example of a cluster plot. It shows the correlations between various currencies, market indices, and commodity prices. Using creative visualization, the figure reveals plenty of information. The cell values in the upper half contain the correlations. The color coding is based on the correlation. Red is -1, Green is +1 and Yellow is 0. In the lower half, you can see a scatter plot where each dot represents the price on a particular day. To the left, you can also see a dendrogram from a hierarchical clustering. The securities are clustered based on their correlation.

Figure 1.2 gives another example of the power of visualization. It depicts Minard's map illustrating the advance and retreat of Napoleon's Grande Armée into Russia in 1812 and 1813 [Minard, 1869]. Remember, Napoleon decided to attack Russia since it didn't want to cease trading with Britain. His army was decimated due to a combination of the Russian winter, lack of supplies, and the Russian army's scorched-earth tactics. The graph essentially combines six different types of data:

- Geographic data as you can see cities and rivers on it
- Path data as it shows the army's movements
- The direction of movement where gold shows troops moving forward and black retreat
- The number of troops as represented by the width of the path where each millimeter represents 10.000 men.
- The temperature and time indicated at the bottom

Yet another powerful example of visualization is a Sankey diagram as shown in Figure 1.3. It illustrates the user journey through a website, starting from four main traffic sources: Organic Search, Paid Search, Social Media, and Direct traffic. The flow then branches into three types of landing pages (Homepage, Product

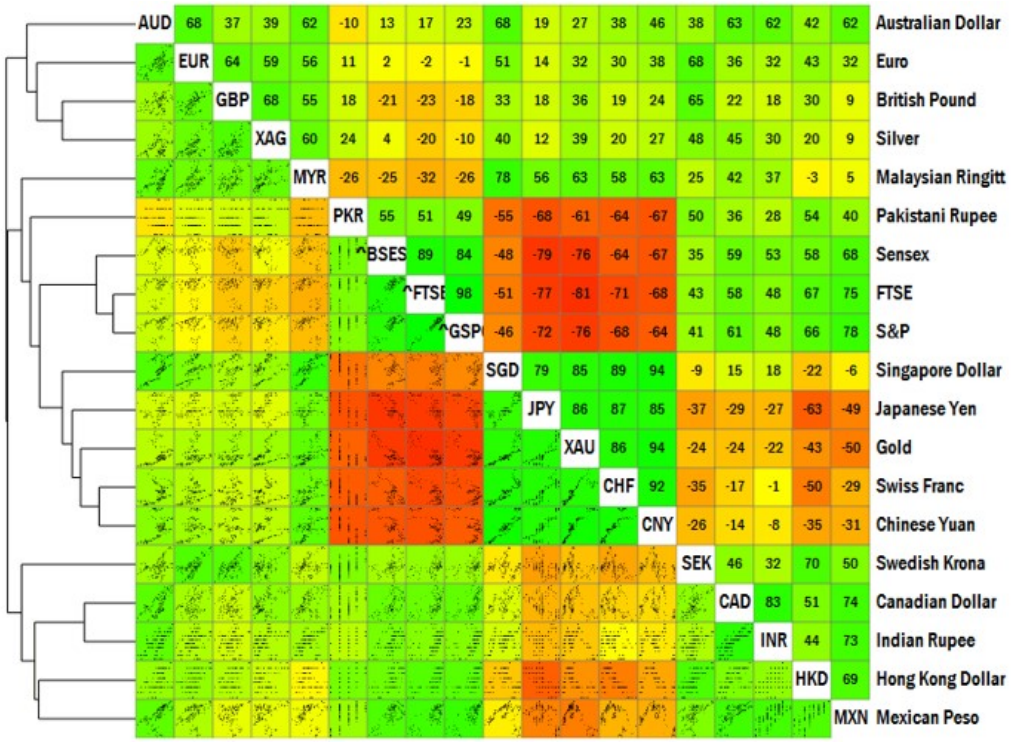


Figure 1.1: Cluster plot (taken from `http://blog.gramener.com/18/visualising-securities-correlation`).

Pages, and Blog Posts), followed by user interactions including Product Views, Add to Cart actions, and Newsletter Signups. Finally, the diagram shows how these interactions culminate in two ultimate outcomes: successful Purchases or Abandoned sessions. The width of each flow represents the volume of traffic, with thicker lines indicating higher user numbers, while different colors are used to distinguish between the various stages of the journey, making it easier to track how users move through the site. Plots like these are also commonly used by web analytics tools like Google Analytics.

Sankey
Sankey diagrams are named after Irish Captain Matthew Henry Phineas Riall Sankey (1853-1925), who first used this visualization type in 1898 while analyzing the energy efficiency of steam engines. Originally, Sankey created this diagram format to show the energy flows in steam engines, demonstrating heat, steam, and mechanical power losses at each stage of the engine's operation.

A basic and intuitive way of visually representing the output of an AI model is based on a traffic light indicator coding as shown in Figure 1.4. Colors are assigned to the various outcomes of a model, for example, representing the probability of a customer being a good payer or a defaulter. When using logistic regression one can bin the outcome probabilities into categories and assign colors to each category. Note that not every color needs to have equidistant intervals in terms of the probability obtained. The green zone indicates that the customer is low risk, whereas the red zone indicates that the customer is likely to default. The colors in between can then be interpreted accordingly. The arrow shifts from left to right and indicates the color assigned to the customer under study. In the figure, the customer has a good score with low default probability. It can then be left to the consideration of the business user to decide where the cut-off color (e.g., for loan approval) or grey zone is situated.

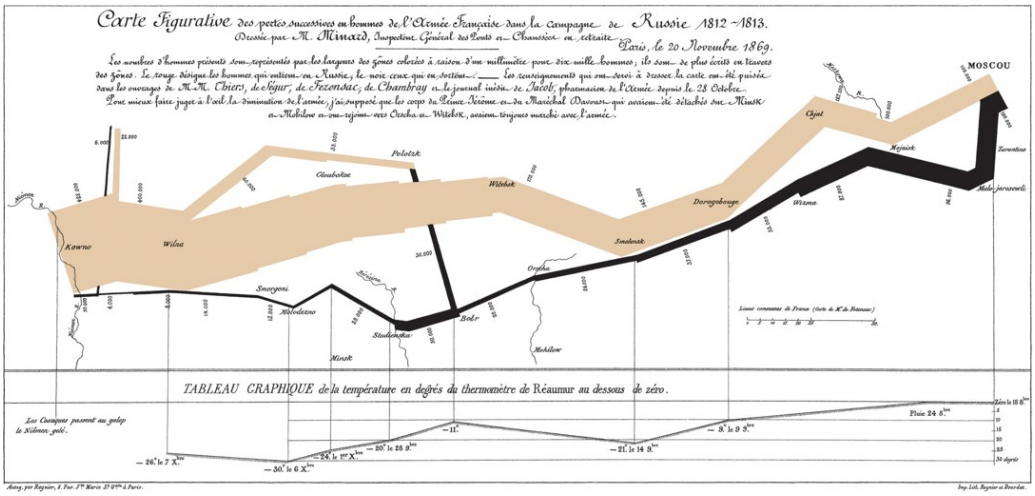


Figure 1.2: Visualisation of retreat of Napoleon's Grande Armée.

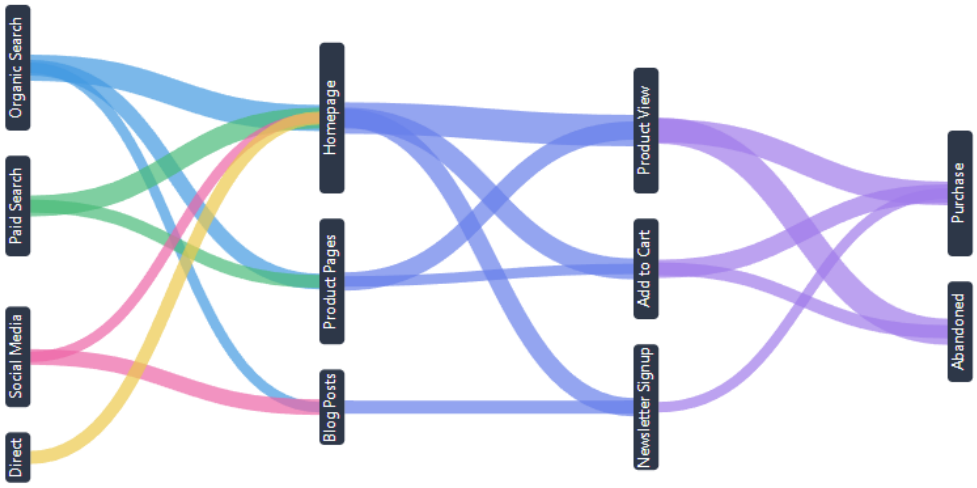


Figure 1.3: Example Sankey plot in web analytics.

1.4 FEATURE IMPORTANCE

1.4.1 Basic Idea

One of the basic tools to interpret black box models is to find out which features are important according to the AI model. Different techniques exist to do this. A first way is by inspecting the mean decrease in impurity (see section **??**) as is commonly done in tree-based ensembles. Obviously, this is model-specific since it will be different between gradient-boosted models and random forests. We can also gauge the importance of a feature by looking at the position of the feature in the trees constructed. Features nearer to the top are obviously more important. Again, note that this is model-dependent. The idea of drop feature importance is to drop a feature from the machine learning model, train it again, and inspect the impact on model performance. This requires multiple retraining runs of the model and can thus be slow. The method, however, is model agnostic. A more powerful method to evaluate feature importance is permutation importance which is generally considered as a better approach and is also model agnostic. We elaborate on this in what follows.

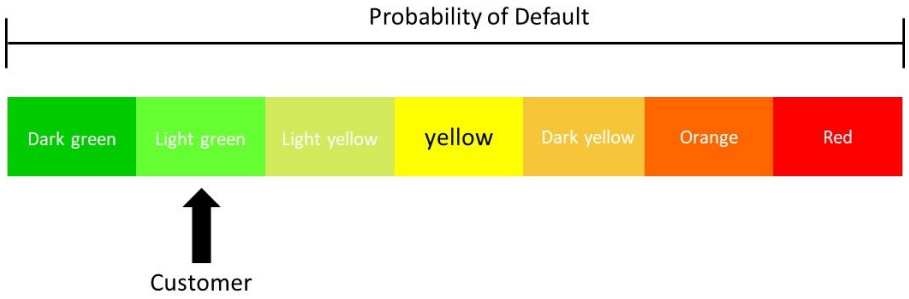


Figure 1.4: Traffic light indicator approach.

1.4.2 Permutation-based feature importance

The idea of permutation-based feature importance is very simple [Breiman, 2001]. It starts from a trained machine learning model on a given data set and a performance measure of interest. You can think of accuracy, area under the ROC curve or even profit. We then randomly permute or shuffle the values for the feature under study. Next, we use the trained model to predict the observations again. The feature importance then corresponds to the difference between the baseline performance measure on the original data and the performance measure on the permuted data set. Note that the shuffling also breaks down the interaction effect, i.e. the way two or more features jointly influence the outcome. You can see an illustration in Figure 1.5. Suppose we start from a data set for churn prediction. It has the well-known RFM features, R for Recency, F for Frequency, M for Monetary, together with Age and Debt. We can then gauge the impact of the recency feature by permuting it as shown in Figure 1.5.

R	F	M	Age	Debt	Churn
1	10	41	23	55	Y
2	11	14	45	31	N
3	12	45	31	31	N
4	13	64	35	14	Y

Permuting R

R	F	M	Age	Debt	Churn
4	10	41	23	55	Y
3	11	14	45	31	N
1	12	45	31	31	N
2	13	64	35	14	Y

Figure 1.5: Permutation-based feature importance.

By repeating this exercise for all features, you obtain a feature importance plot as shown in Figure 1.6 where it can be clearly seen that the frequency is the most important feature.

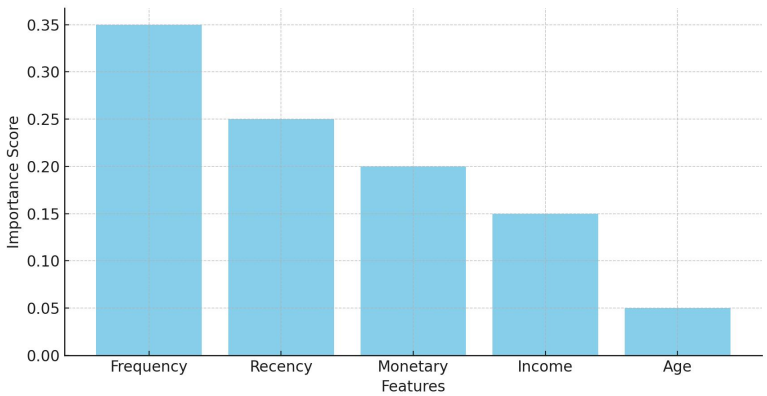
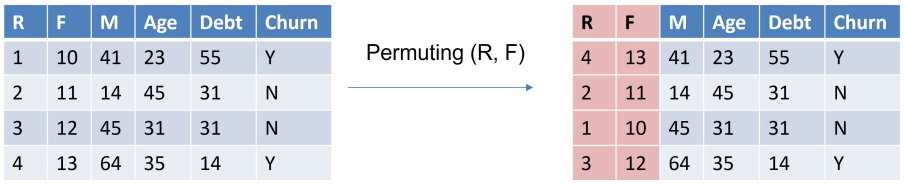


Figure 1.6: Feature importance plot.

Permutation-based feature importance indicates which features are important, but not to which extent

they affect the outcome or prediction. The procedure can also be easily used for feature selection by retraining a model on the top *N* most important features only as done in the Boruta R package. Do note however that correlated features are likely to share importance which may lead to misinterpretation. Hence, it is still important to check for correlated features. It is also possible to investigate multiple features at once to zoom in on interaction effects. This can be done by permuting two or more features together as illustrated in Figure 1.7 . Note that it perfectly makes sense to assess the drop in predictive power (e.g.,



R	F	M	Age	Debt	Churn
1	10	41	23	55	Y
2	11	14	45	31	N
3	12	45	31	31	N
4	13	64	35	14	Y

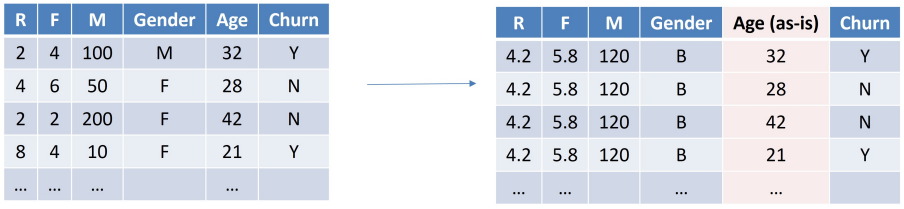
R	F	M	Age	Debt	Churn
4	13	41	23	55	Y
2	11	14	45	31	N
1	10	45	31	31	N
3	12	64	35	14	Y

Figure 1.7: Permutation-based feature importance: checking two features together.

in terms of accuracy, AUC or profit) on both the training set (used to train the model) and the test set (used for model evaluation), which represents unseen data. For the training set, permutation-based feature importance allows us to learn and understand how the model has actually relied on each feature. By doing permutation-based feature importance on the test set, we can verify whether the feature rankings are similar and there was no overfitting of the model on the training set.

1.4.3 Partial Dependence plots

The aim of partial dependence plots is to understand how a feature impacts the outcome of a machine learning model [Friedman, 2001]. The core idea is to keep the feature under study as-is and impute all others with the median (for continuous features) or mode (for categorical features) across all observations as illustrated in Figure 1.8. The trained model is then used to predict the new data set. We then plot the predicted probabilities for the values of the feature under study. Note that this procedure can also be implemented by defining a grid-based range over the feature under study between its observed minimum and maximum.



R	F	M	Gender	Age	Churn
2	4	100	M	32	Y
4	6	50	F	28	N
2	2	200	F	42	N
8	4	10	F	21	Y
...	...	...		...	

R	F	M	Gender	Age (as-is)	Churn
4.2	5.8	120	B	32	Y
4.2	5.8	120	B	28	N
4.2	5.8	120	B	42	N
4.2	5.8	120	B	21	Y
...	...		...	...	

Figure 1.8: Data set for partial dependence plot.

Figure 1.9 illustrates a partial dependence plot of the purchase probability of a product versus age. You can see that the probability first increases and then drops after the age of 50. This could be the case for products such as life insurance, financial planning, or family cars.

In terms of interaction effects, it is important to note that the absence of evidence does not mean evidence of absence! In fact, the partial dependence plot can be easily used to our benefit to detect interaction effects. More specifically, to detect interaction effects we can contrast patterns observed in the data with a partial dependence plot. Let's give an example of this. Say we categorize the age feature and look at the percentage of churners in each of the categories. Assume now that according to the data, when plotting churn in function of age, the churn rate drops for customers between 30 and 40 years old. Furthermore, assume that according to the partial dependence plot, the churn probability stays constant for customers between 30 and 40 years. This clearly hints at an interaction effect between age and another feature. Hence,

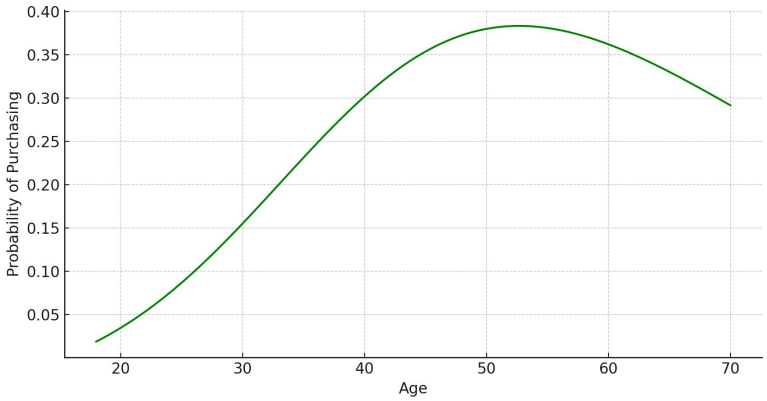


Figure 1.9: Partial Dependence Plot for age.

it is important to jointly inspect both the data and the partial dependence plots during model development and interpretation.

Like permutation importance, it is possible to keep more than one feature as-is whilst replacing the others with their median and mode. Note however that this is usually harder to visualize. Popular visualization methods are contour and 3D plots. Obviously, this quickly becomes infeasible for higher dimensions.

1.4.4 Individual Conditional Expectation (ICE) plots

The principle of individual conditional expectation or ICE plots is very similar to the idea of partial dependence plots we discussed before [Goldstein et al., 2015]. The key idea is to not replace features with the median or mode and hence keep every feature as is. Instead, we create new observations based on the values of the feature under study. It is also possible to define a grid-based range over the feature under study between its observed minimum and maximum.

You can see this illustrated in Figure 1.10. Say we again start from a data set with the RFM features, R for Recency, F for Frequency and M for Monetary, complemented with Age. We keep the RFM features to their original values. However, we add more values for age. We then replicate each observation but with different values of age as illustrated in the table to the right. You can see that every observation is replicated 4 times in our example.

R	F	M	Age
2	4	100	32
4	6	50	28
2	2	200	42
8	4	10	21
...	...	...	...

R	F	M	Age				
2	4	100	21	28	32	42	...
4	6	50	21	28	32	42	...
2	2	200	21	28	32	42	...
8	4	10	21	28	32	42	...
...	...	...	...				

R	F	M	Age
2	4	100	21
2	4	100	28
2	4	100	32
2	4	100	42
4	6	50	21
4	6	50	28
4	6	50	32
4	6	50	42
2	2	200	21
2	2	200	28
2	2	200	32
2	2	200	42
8	4	10	21
8	4	10	28
8	4	10	32
8	4	10	42
...	...	...	...

Figure 1.10: Data set for ICE plot with four customers and varying age values.

Every original observation now leads to multiple observations or rows in the modified data set. We then let the machine learning model predict over all these observations. For each distinct value of the feature under study, we obtain multiple predictions, allowing us to plot multiple lines. In Figure 1.11 you can see this illustrated once more. We then plot four lines for the table to the right.

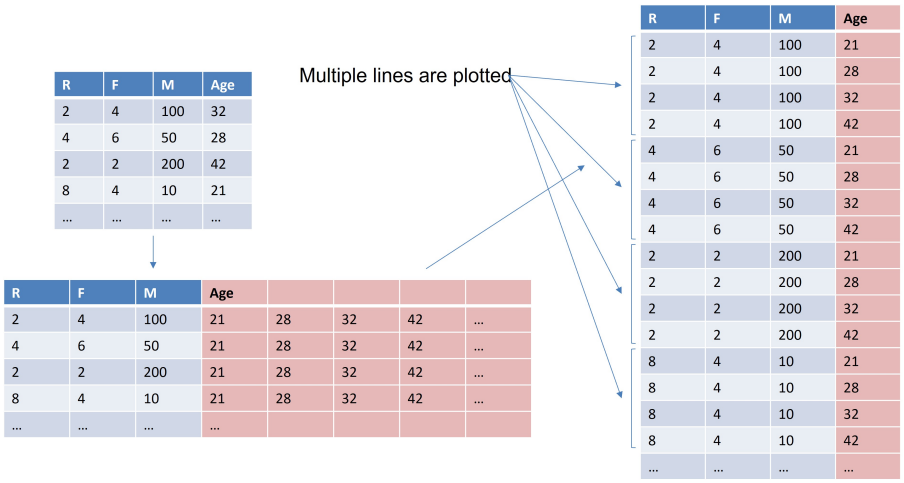


Figure 1.11: Data set for ICE plot.

Figure 1.12 illustrates how the probability of subscribing to a product varies with age for several individuals, each represented by a unique line. Note that Base Age represents the original value in the data set. The differences in the lines arise due to individual variability, feature interactions, non-linear effects, and model complexity. The various lines can be summarized into an average line (shown as the dashed line in Figure 1.12) depicting the overall effect of the feature. Hence, ICE plots can be used both as local as well as global XAI methods.

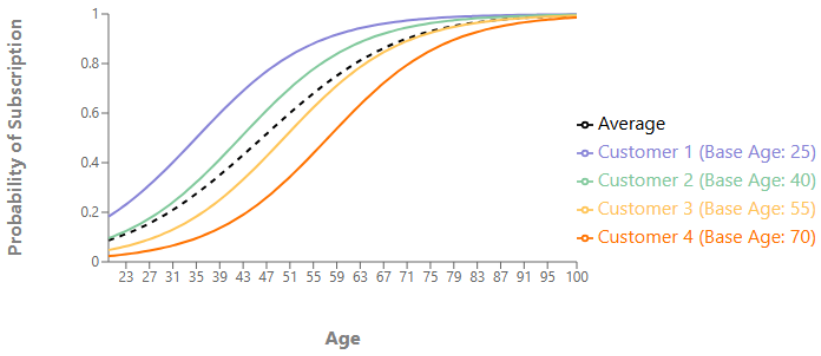


Figure 1.12: Example ICE plot for four observations.

ICE plots are really well suited to show the behavior of a feature across the entire data set. It is however important to note that ICE plots show individual effects whereas PDP plots show average effects. It is recommended practice to inspect ICE plots for the important features. The same remark we made about training versus test set when discussing partial dependence plots also applies here. In other words, for the training set, ICE plots allow us to learn and understand how the model has actually relied on each feature. By doing ICE plots on the test set, we can verify whether the feature impacts are similar and there was no overfitting of the machine learning model on the training set.

Partial dependence and ICE plots should be a key tool to any AI model developer working with black box models and interested in understanding the impact of features. Both are essential tools to bridge

the gap between complex machine learning models, such as XGBoost or (deep) neural networks, with great predictive accuracy and interpretability. They allow to validate the model, bridge the gap towards the business decision makers, and provide explanations to end users.

1.5 SCORECARD APPROACH

A popular way of representing regression based models is by assigning points to characteristics which are then added to an overall score indicating, e.g., the creditworthiness of a customer. Consider, e.g., a logistic regression model, where the inputs have been categorised and coded using weights of evidence coding,

$$P(y = 1 \mid \mathbf{x}) = \frac{1}{1 + \exp(-(w_0 + w_1 woe_1 + \ldots + w_n woe_n))}. \tag{1.1}$$

in which $P(y = 1 \mid \mathbf{x})$ is ..., w_i is ...

Remember weights of evidence coding assumes that each variable has been categorised and is modeled based on the good-bad odds in each category (see Chapter **??**).

Reformulating the logistic regression classifier (see section **??**) in terms of the log odds gives:

$$\log(\frac{P(y = 1 \mid \mathbf{x})}{1 - P(y = 1 \mid \mathbf{x})}) = w_0 + w_1 woe_1 + \ldots + w_n woe_n. \tag{1.2}$$

We prefer to work with the log(odds) since this is linear in the input features. However, we could also linearly recalibrate the log(odds) to the score range we like in the following way:

$$score = \log(odds) * \alpha + \beta. \tag{1.3}$$

The parameters α and β need to be estimated by specifying two points on the line. E.g., we could require that when the odds are 10:1, 500 points should be assigned and 50 extra points for double odds. This would give

$$\begin{cases} 500 = & \log(10) * \alpha + \beta \\ 550 = & \log(20) * \alpha + \beta, \end{cases} \tag{1.4}$$

or,

$$\begin{cases} \alpha = & \frac{50}{\log(2)} \approx 72.13 \\ \beta = & 500 - \log(10) * \alpha \approx 333.09 \end{cases} \tag{1.5}$$

The score can then be computed as follows:

$$\begin{aligned} score &= \log(odds) * \alpha + \beta \\ &= (\textstyle\sum_{i=1}^{n}(w_i woe_i) + w_0) * \alpha + \beta \\ &= (\textstyle\sum_{i=1}^{n}(w_i woe_i + \frac{w_0}{n})) * \alpha + \beta \\ &= \textstyle\sum_{i=1}^{n}((w_i woe_i + \frac{w_0}{n}) * \alpha + \frac{\beta}{n}) \end{aligned} \tag{1.6}$$

In other words, the scorecard points for attribute i are then calculated as $((w_i woe_i + \frac{w_0}{n}) * \alpha + \frac{\beta}{n})$. This is illustrated in Table 1.1, where $w_0 = 1$, 500 points are assigned when the odds are 10:1 and 50 extra points for double odds. A 32-year-old male credit applicant, with an income of 1500 would then receive 142 + 101 + 187= 430 points. The advantage of this points-based scorecard is that it is very clear how the different characteristics contribute to the overall credit score. This also allows to give advice to credit applicants on how to improve their score.

Scorecards in credit risk
Scorecards are very commonly used in credit risk [Baesens et al., 2016]. In application scoring, scorecards are used as explanatory models to decide upon credit approval or rejection. In case of the latter, the scorecard provides an intuitive interpretation of why credit was denied as well as recommendations on how to improve the credit score.

Characteristic	Category	WOE	w_i	Points
Age	< 30	-0,25	0,88	119
	[30,40]	0,10	0,88	142
	> 40	0,65	0,88	177
Gender	Female	0,75	-0,35	116
	Male	1,35	-0,35	101
Income	< 1000	-1,87	0,90	14
	[1000,10000]	0,80	0,90	187
	>10000	2,03	0,90	267

Table 1.1: Example of a points-based scorecard.

1.6 META DECISION TREES

The idea of meta decision trees is to train a classical decision tree to predict a black box model's target rather than the original target variable in the data. In doing so, the decision tree basically tries to learn the behavior of the black model in an interpretable way. Obviously, it will not be perfectly capable in doing so as a decision tree typically assumes decision boundaries orthogonal to the axes (see Chapter **??**X) but a good approximation of the black box model's functioning can already be useful from an interpretability perspective. The approach comes with an extra benefit. In fact, as the tree is grown the data used to make the splitting decisions becomes more sparse. To circumvent this and add more meaningful splits to the tree the black box model can be used as an oracle to generate more labeled data which will allow to make more powerful splits. One can then calculate the fidelity of the decision tree as the percentage of observations it predicts in the same way as the black box model. The steps are described in Algorithm 1. The fidelity measures the percentage of observations which are predicted (e.g., classified) in the same way by both the model and the meta decision tree. Obviously, lower/higher values indicate a worse/better approximation of the model's reasoning. Obviously, it is important to benchmark this against a decision directly estimated from the original targets.

Algorithm 1 Meta Decision Tree.

```
Input: Original dataset with features and target
Output: Meta decision tree model approximating the black box model
black_box_model ← train_black_box_model(original_data)
new_targets ← black_box_model.predict(original_data)
meta_decision_tree ← train_decision_tree(original_data, new_targets)
while more_data_needed do
    new_data ← generate_new_data()
    new_labels ← black_box_model.predict(new_data)
    append_to_training_set(new_data, new_labels)
    meta_decision_tree ← retrain_decision_tree(current_tree, new_data, new_labels)
end while
fidelity ← calculate_fidelity(meta_decision_tree.predict(original_data), new_targets)
```

Trepan
Trepan (short for Trepanation) is a popular meta decision tree algorithm designed to extract decision trees from trained neural networks, thereby improving their interpretability without sacrificing predictive accuracy [Craven and Shavlik, 1996]. The method operates by recursively querying the neural network to create a decision tree that approximates the network's decision boundaries. By generating synthetic examples and prioritizing splits that maximize information gain, Trepan ensures that the extracted tree captures the logic of the neural network. In [Baesens et al., 2003], we successfully applied Trepan for extracting decision trees from neural networks for credit risk evaluation.

1.7 LIME

LIME is another XAI technique and stands for Local Interpretable Model-agnostic Explanations [Ribeiro et al., 2016]. Essentially, LIME implements a "local surrogate" model to provide predictions. More specifically, LIME helps to explain single individual predictions. It is model-agnostic and works with tabular as well as other types of data (e.g. text, images).

Assume that we have trained a black-box classifier $f()$ and want to explain a given instance $\mathbf{x}_i$. Assume that the prediction of the black box classifier equals $f(\mathbf{x}_i) = p(y \mid \mathbf{x}_i) = 0.78$ for the case of binary classification and we want to know why? The main idea of LIME is as follows. First, create a new dataset containing permuted samples around $\mathbf{x}_i$ and have the black box model provide predictions for these. In a second step, train a local, interpretable model on this data set and use it to offer explanations. This model starts by taking a weighted sample based on the proximity of the permuted instances around $\mathbf{x}_i$. LASSO regression or a regression tree are commonly used for this purpose as they are considered to give highly interpretable models.

You can see this illustrated in Figure 1.13. Assume we start from a binary classification data set with 2 features: x_1 and x_2. We train a random forest classifier which results in the decision boundary illustrated in the figure to the right. We now select an instance we wish to explain. Let's say we would like to explain the

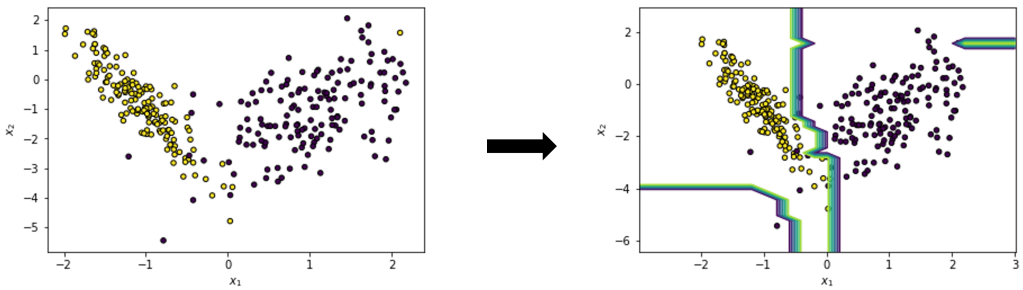


Figure 1.13: Example data set and trained random forest classifier.

red observation in Figure 1.14. We first create a permuted data set which we obtain by uniform sampling for tabular data as shown in Figure 1.15. The observations sampled around the red instance under study are all weighted: the further from the instance the lower the weight, as you can see from the smaller size of the blue dots in the figure to the right. Note that most LIME implementations use an exponential smoothing kernel to do this, which is similar to t-SNE (see Chapter **??**). We can then have the black box model, which is a random forest in our case, make its predictions or thus calculate its probabilities as shown in Figure 1.16. Next, we sample from the perturbed data set based on the weights. You can see the result of this in Figure 1.17. A simple machine learning model is then trained on this perturbed data set. In our case, we use LASSO regression (see Chapter **??**). Note that the output of this regression model is now a continuous value representing the predicted probabilities of the black box model as shown in Figure 1.18. This provides us with a simple, local decision boundary which can be easily inspected. From our figure, it is clear to see that variable x_2 is not important. The line going through x_1 equals 0 is the most important since the class on

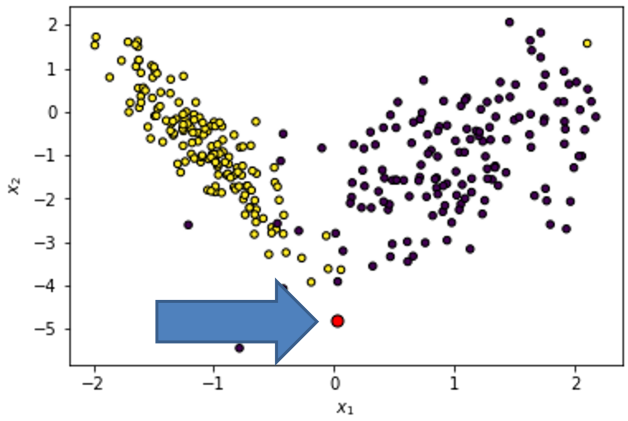


Figure 1.14: Observation to explain using LIME.

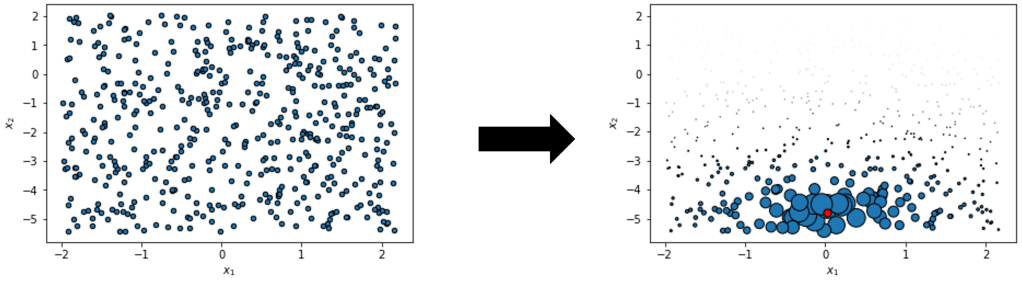


Figure 1.15: Permuted data set (left) and weighted according to distance to observation to explain (right).

the right side of it is the other class. You can clearly see in Figure 1.19 that the decision boundary of the "explanatory model" approximates the decision boundary of the random forest model around the instance under study.

LIME can also be applied to non-tabular data such as text or images. Non-tabular data changes how the perturbation is performed. In case of text data, new texts are created by randomly removing words from the original text. More specifically, if the dataset is represented with binary features for each word (bag of words, see section **??**), this can be done in a very straightforward way. For images, perturbing individual pixels does not make a lot of sense. Instead, groups of pixels in the image are perturbed at once by "blanking" them (removing them from the image). These groups are called "superpixels", based on interconnected pixels with similar coloring. They can be found using for example a k-means clustering procedure.

Note that defining a neighborhood around an instance to define the weights is not straightforward. For example, the distance measure or bandwidth of the exponential smoothing kernel can heavily impact the results. As a simple alternative, one can also select the k nearest neighbors around the instance under study, but do note that we then still need to decide upon an appropriate k and a distance metric. Choosing the simple model is also somewhat arbitrary. For example, the tuning of the regularization parameter of the LASSO regression can impact the results (see Chapter **??**). Furthermore, LIME's explanations can be unstable, meaning that multiple runs on the same instance might produce different feature importance rankings due to the random sampling in its perturbation process. The technique can be computationally expensive since it needs to generate numerous perturbed samples and get predictions from the black-box model for each instance we want to explain. Additionally, LIME assumes local linearity around the instance being explained, which may not hold true in complex models where the decision boundary is highly nonlinear, potentially leading to misleading explanations. The main advantage is that LIME works on tabular data, text and images.

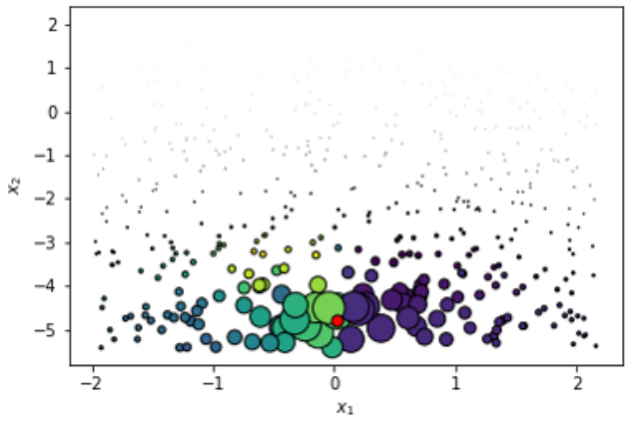


Figure 1.16: Random forest predictions for permuted data set.

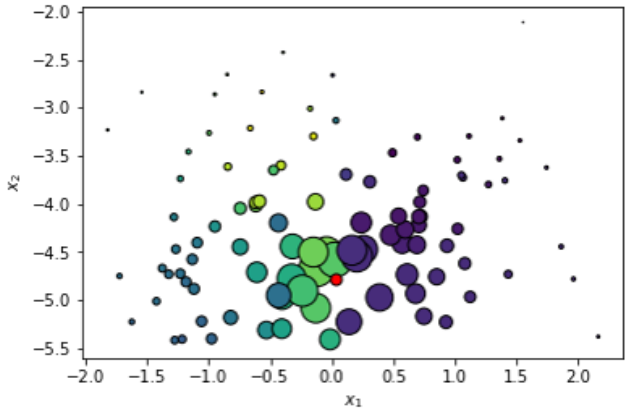


Figure 1.17: Sampling from permuted data set based on weights (larger scale).

1.8 ANCHORS

Anchors provide local model explanations by identifying a subset of feature-value conditions (also called anchors) that are highly influential in a model's prediction for a specific instance [Ribeiro et al., 2018]. The method was developed by the same team of researchers as LIME. These conditions effectively capture the key factors driving the model's decision by acting as a rule that anchors the prediction, offering a clear and intuitive understanding of why the model behaved as it did.

For instance, consider a model predicting whether a person is a "Good Payer" for a loan. An example anchor might be: "If Age $>$ 30 and Income $>$ 50k", then the model predicts Good Payer with 95% confidence." This anchor communicates that as long as these conditions are met, the model's prediction is very likely to remain consistent, regardless of how other input features vary. This provides a human-readable explanation that focuses on the most critical conditions.

The process of generating anchors begins with a particular observation and its corresponding model prediction. The algorithm systematically explores possible feature-value conditions, such as "Age $>$ 30" or "Income $>$ 50k," and evaluates their influence by testing whether the model's prediction remains unchanged when the other features of the input are perturbed. This involves creating synthetic instances where all features except the ones fixed by the condition are randomly altered, and the model's prediction is recorded. Conditions that consistently preserve the prediction across these perturbations are considered meaningful because they indicate the model's reliance on these specific factors.

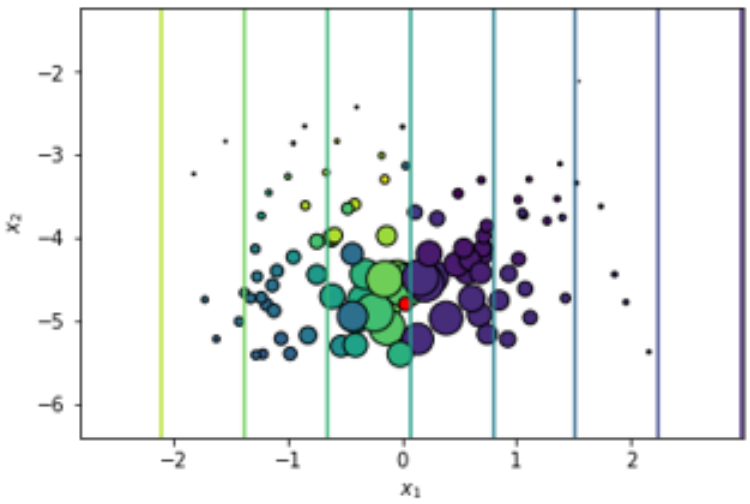


Figure 1.18: LASSO regression predictions on permuted data set.

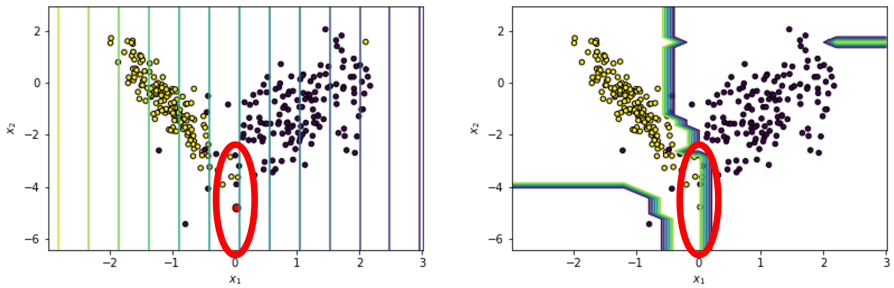


Figure 1.19: Random forest versus Lasso regression prediction.

The goal is to identify a set of feature-value conditions combined into an If-Then rule that explains the prediction with high precision. Precision measures how consistently the anchor conditions lead to the same prediction as the original instance. In addition to precision, the method also considers coverage, which measures the proportion of possible input variations for which the anchor conditions hold. A good anchor achieves both high precision and sufficient coverage, ensuring that the explanation is not only accurate but also broadly applicable to similar cases. By combining these properties, anchors provide a localized and interpretable explanation for individual predictions.

While LIME provides explanations through local linear approximations, Anchors takes a different approach by finding rules that are sufficient to ensure a prediction. Where LIME might say 'these features contributed positively or negatively to the prediction,' Anchors says 'these conditions are sufficient to guarantee this prediction with this probability (certainty).' This makes Anchors particularly useful when you need high-precision explanations and are willing to trade off some model coverage for certainty in your explanations, as the rules produced by Anchors are more likely to be stable across similar instances.

1.9 SHAPLEY VALUES

Shapley values are both a local and global XAI technique used to interpret feature contributions at observation or model level. This is however not the origin of the Shapley value, since Shapley was an economist interested in game theory for which he was awarded the Nobel prize [Shapley, 1953]. Therefore, let us first focus on the game-theoretical concept of the Shapley value to get a grasp of how it works.

Shapley
Lloyd Stowell Shapley (1923–2016) was a renowned American mathematician and economist, celebrated for his contributions to game theory and economics. Born in Cambridge, Massachusetts, Shapley demonstrated exceptional mathematical talent from an early age. He attended Harvard University but interrupted his studies during World War II to serve in the U.S. Army Air Corps, where he earned a Bronze Star for his contributions to breaking Soviet weather codes. Shapley's most significant contribution to economics was the development of the Shapley value, a method for fairly distributing rewards or costs among players in cooperative games. Published in 1953, this concept became a cornerstone of cooperative game theory, addressing how individuals in a group should divide resources based on their marginal contributions to the group. Beyond the Shapley value, he made substantial contributions to the theory of matching markets, as exemplified by the Gale-Shapley algorithm for stable matchings, which earned him the Nobel Memorial Prize in Economic Sciences in 2012, shared with Alvin Roth.

The original context of Shapley is to assess each players' contribution in a cooperative game. Let's take a simple example: an indoor soccer team (futsal) with players A, B, C, etc. The goal is to find out how much each player contributes to the team. In game theory this would be the payoff, but we could also call it the marginal gain/contribution of each player. We assume that we have a characteristic function to assess the value of a soccer team. Assume that adding player A will give the team a value of 6: $v(A) = 6$. Now, we can try to complete the team by adding extra players. Adding extra players will increase the value of the team. However, players can have overlapping skill sets. Say that adding player B to player A gives us a value of the characteristic function of 10: $v(A, B) = 10$. Hence, B has a marginal contribution of 4. Adding player C increases the value to 12, $v(A, B, C) = 12$, so C contributes only 2 to the team. However, if this small value is due to player B already being in the team before C, then we should take into account what would happen if C came in first. This is the only fair way to address all player's payoffs. So the sequence might as well have been $v(A) = 6$, $v(A, C) = 10$, $v(A, C, B) = 12$ in which case the contributions of C and B would have been 4 and 2 respectively. In other words, if we want to be perfectly fair in our assessment, we should look at all possible team configurations we can form with player C to know what value he really adds to the team. We can then average player C's contribution across all possible teams.

Building upon these insights, it is straightforward to see that only the players in the team before C matter, not the order in which these were added. Whether A was added first to the team or B doesn't matter. This means we only need to know the value of adding C to a team with A and B once and weigh it with the number of teams we can form with A and B. Another intuition is that all players that come after C are not important. We do need to take into account that we could add players D and E in a different order and it would not impact the marginal contribution of adding C to a team having already A and B. In other words, $v(A, B, C) - v(A, B)$ is the same for the sequences ABCDE, ABCED, BACDE and BACED

Let's now translate this intuition to a mathematical formula with ϕ_j representing the contribution of player j and $x_1, \ldots, x_p$ the players in the team:

$$\phi_j = \sum_{S \subseteq \{x_1, \ldots, x_p\} \setminus \{x_j\}} \frac{|S|!(p - |S| - 1)!}{p!} \left[v_{S \cup \{j\}}(x_{S \cup \{j\}}) - v_S(x_S) \right] \tag{1.7}$$

Let's elaborate on this formula into more detail. We have a team with p players and we want to know the payoff for player j among those p players. This means we have to look at all possible subsets where we can add player j to. Put differently, we take sets of players out of all p players, not including player j since this is the one we are adding to the team. So let's work this out again for player C in our soccer team. In other words, we consider all possible teams with players A, B, D and E. This can be teams of all possible sizes: AC, ABC, BDC, AEDC, etc. Note that also the empty set is a subset: this is the value of a team with only C in it. We saw earlier that the order of the players in the subset did not matter. However, if we want to weigh the contribution of C over all possible teams he can be added to, then we need some combinatorics to determine the weights. Again, recall that C has the same contribution whether the team was AB or BA before he was added. All permutations of players in the team before C are given by $|S|!$. The number of

residual players left after adding C to the subset equals $p - |S| - 1$. So the factorial of this term gives all permutations how we can add these players. p! is a normalization term and corresponds to the total number of teams possible taking the order into account. This is the denominator of the weights. The actual contribution of C (respectively j) is then the value of the team with C minus the value of the team without C. We just need to calculate this value for all possible teams. In other words, the Shapley value of player C is his marginal value over all possible sub-teams and weighted over all possible permutations of these teams.

Let us now translate the problem to a machine learning (ML) context [Lundberg and Lee, 2017a]. Suppose you have trained a (black box) ML model on your data. We want to know how much each feature contributes to the model's predictions for a specific observation. Here $\hat{f}()$ is the estimated function by your machine learning model (e.g., XGBoost, random forest, etc.). X_S is a subspace of all features. To obtain the estimate of the model over the subset of features, you will have to integrate out the other features that are not in the subset (i.e. marginalize for the features that are not in the model) as follows:

$$v(X_S) = \int \hat{f}(x_1, \ldots, x_p)\, dP_{x \notin S} - E_X(\hat{f}(X)) \tag{1.8}$$

Remember $v(X_S)$ represents the value function for a subset S of features. The integral calculates the expected prediction when we keep the features in subset S fixed at their original values and average over all possible values of the other features (those not in S) according to their distribution P. $E_X(f(X))$ is the global average prediction or baseline. The difference between both these terms gives us the marginal contribution of the features in subset S. Note that if you subtract two value functions for different subsets, the baseline term actually disappears.

All possible subsets of feature values have to be evaluated with and without feature j to calculate the exact Shapley value.

$$\hat{\varphi}_j = \frac{1}{M} \sum_{m=1}^{M} \left(\hat{f}(x_{+j}^m) - \hat{f}(x_{-j}^m) \right) \tag{1.9}$$

Monte Carlo sampling becomes necessary because computing exact Shapley values requires evaluating all possible feature combinations, which grows exponentially with the number of features. For instance, with just 20 features, we would need to evaluate 2^{20} (over 1 million) combinations for each prediction. This makes exact computation computationally infeasible for most real-world applications. For every feature j, we sample M times a random permutation of the other features and calculate the difference in predicted value for a specific observation. The Shapley value is then the average of all these samples. The number of Monte Carlo samples (M) represents a trade-off between computation time and estimation accuracy. In practice, M is often chosen based on: the number of features (more features typically require larger M), the desired precision of the estimates and computational constraints. Typical values of M range from 100 to 1000 samples.

Let's work out an example of how to calculate the Shapley value. Say we have an observation x, Bart, with features age, income, Rec for Recency, Freq for Frequency, Mon for Monetary and debt. Assume we want to determine the importance of F (feature j) for Bart as depicted in Table 1.2.

Customer	age	income	Rec	Freq	Mon	debt
Bart (=x)	44	2500	5	2	1000	100
Tim (=z)	36	3000	10	4	500	50

Table 1.2: Data for Bart (x) and Tim (z).

We start an iterative procedure for m starting at 1 and going to M. We draw a random instance z from our data. Say this is Tim. Tim could have the same or different target value (e.g., churn). For Shapley value calculations, we want the random instance z (i.e., Tim) to be representative of the overall data distribution,

including the target variable. We then choose a random permutation of the features, say R, debt, F, age, M and income. Table 1.2 illustrates the values of these features for Bart and Tim. We then construct two new instances. The first one, x_{+j}, includes all the values of R, Debt and F for Bart, and the values of age, M and income for Tim. The second instance, x_{-j}, includes the values of R and debt for Bart and the values of F, age, M and income for Tim as shown in Table 1.3.

	Rec	**debt**	**Freq**	**age**	**Mon**	**income**
Bart (x_{+j})	5	100	2	**36**	**500**	**3000**
Bart (x_{-j})	5	100	**4**	**36**	**500**	**3000**

Table 1.3: Data for Bart with variables x_{+j} and x_{-j}.

We then calculate φ_j^m as the difference in the prediction of the machine learning model for both x_{+j} and x_{-j}

$$\varphi_j^m = \hat{f}(x_{+j}) - \hat{f}(x_{-j}). \tag{1.10}$$

Once the iteration from $m = 1$ to M has finished, the importance of feature j for x can be calculated as the average of the φ_j^m values

$$\varphi_j(x) = \frac{1}{M} \sum_{m=1}^{M} \varphi_j^m. \tag{1.11}$$

To visualize Shapley values and their contributions effectively, Shapley plots are often used. To create these plots, we first calculate the Shapley values for each feature and observation using either the exact method or a Monte Carlo approximation as described earlier. These plots highlight the individual contributions of features to the prediction for a given observation or across multiple observations in a dataset.

One common type of Shapley plot is the force plot, which shows the positive and negative contributions of features relative to a baseline prediction for one particular observation. This is illustrated in Figure 1.20 for an XGBoost classifier trained on the publicly available Taiwanese credit card data set (see `https://archive.ics.uci.edu/`). Note that PAY_0 indicates the number of months payment is delayed (0, 1, 2, etc), LIMIT_BAL is the credit limit assigned, PAY_AMT1 is the most recent payment, BILL_AMT1 is the outstanding balance and AGE is the age of the customer. The force plot visualizes how each feature

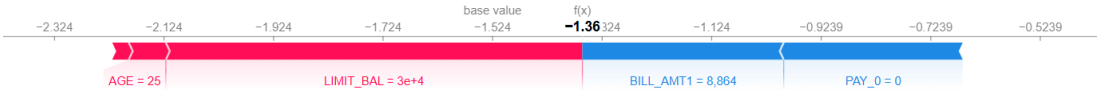


Figure 1.20: Example force Shapley plot.

contributes to the model's log odds for a specific observation. Starting from the base value -1.254, which represents the average prediction of the model, the contributions of individual features are shown as arrows pointing towards the final prediction (-1.36 in our case) for the observation. Features with red arrows positively contribute, pushing the prediction higher, while blue arrows negatively contribute, lowering the prediction. For example, in our credit risk model, a high PAY_0 (history of late payments) contributes positively (increasing the risk of default), while a high LIMIT_BAL (credit limit) has a negative contribution, indicating lower risk. The magnitude of each arrow reflects the strength of the feature's impact on the prediction.

Another widely used plot is the summary plot, which aggregates Shapley values across multiple observations to display the global importance and distribution of each feature's impact as illustrated in Figure 1.21. In the figure, each dot represents a SHAP value for a specific feature and observation, with the position on the x-axis indicating the magnitude and direction of the feature's impact. Features that increase the

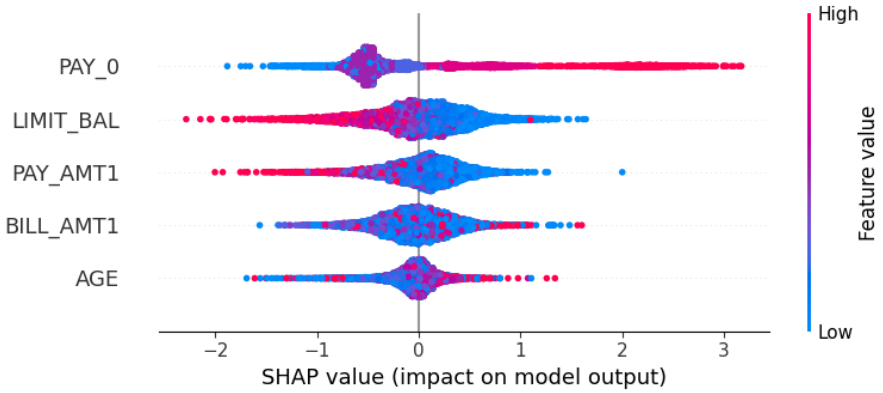


Figure 1.21: Example summary Shapley plot.

prediction are shown on the positive side, while features that decrease it appear on the negative side. The colors of the dots represent the feature values: red dots correspond to high feature values, and blue dots represent low feature values. For example, in our credit risk model, a high PAY_0 (red) has a strong positive impact on default risk, while a high LIMIT_BAL (red) reduces default risk. This plot highlights which features are most important globally and how their effects vary across the dataset.

To further enhance the computation and interpretation of Shapley values, various extensions have been proposed. TreeExplainer is specifically optimized for tree-based models like XGBoost, LightGBM, and Random Forests, leveraging their internal structure to compute exact Shapley values in polynomial time rather than relying on the exponential complexity of the original formulation [Lundberg et al., 2020]. KernelSHAP [Lundberg and Lee, 2017b] approximates Shapley values using a specially weighted local linear regression. It improves upon the original SHAP framework by using a kernel function that assigns weights to perturbed samples based on the size of their feature coalitions, ensuring that the resulting coefficients are exact Shapley values. While computationally more intensive than TreeSHAP, KernelSHAP provides consistent estimates for any model type and handles feature interactions naturally.

To summarize Shapley values are capable of providing consistent and fair attributions across all possible feature combinations and their applicability to both local (individual prediction) and global (model level) explanations. However, Shapley values have weaknesses, such as their computational complexity, especially for models with many features, which often necessitates approximations like Monte Carlo sampling. Another challenge arises in the presence of multicollinearity, where features are highly correlated. In such cases, Shapley values may distribute contributions ambiguously among correlated features, leading to explanations that are harder to interpret. For example, if two features provide similar information, Shapley values may split their contributions arbitrarily, obscuring the true importance of each feature. Addressing multicollinearity requires careful preprocessing, feature selection, or domain expertise to interpret results in context. Finally, while Shapley values offer stronger theoretical guarantees, they are computationally more expensive and may be harder to explain to non-technical audiences than LIME's linear approximations or Anchors' If-Then rules.

1.10 COUNTERFACTUALS

Counterfactuals are a model agnostic and local XAI technique because it focuses on a specific observation and its neighborhood in the input space [Wachter et al., 2017]. More specifically, it tries to answer the question: "What minimum changes in the input features would alter the prediction to a desirable class?". In Figure 1.22, the observation Bart is classified as a bad payer (red) because his income and age are positioned below the decision line. The idea of counterfactuals is now to determine the minimal changes in Bart's features (income and age) needed to shift his classification to a good payer (green). Two options are

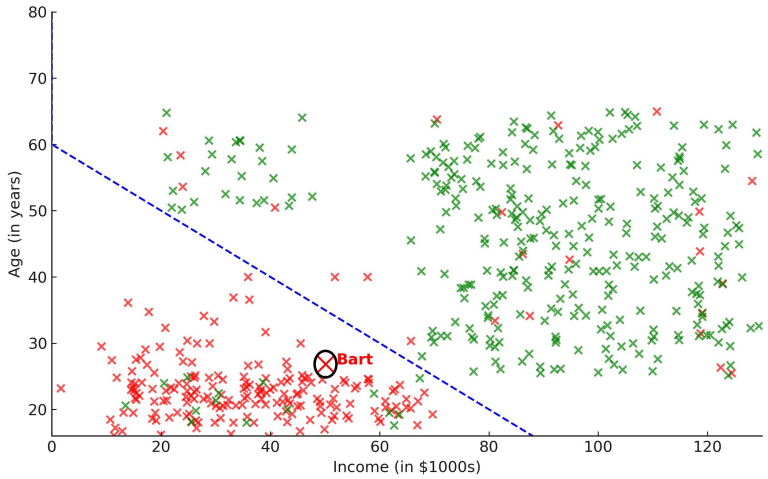


Figure 1.22: Example of counterfactuals.

available; either his income needs to increase given his age, or he has to wait until his age increases whilst keeping his income. An increase in both is also an option. Examples of counterfactuals in other applications could be: ”What lifestyle adjustments might reduce my risk of a disease?”, ”What qualifications would make my profile a match for a job?”, etc. Counterfactuals can help highlight scenarios where decisions could seem unfair or biased, helping to refine the decision-making process.

Counterfactuals can be computed by solving an optimization problem that identifies the minimal changes needed in the input features to achieve a desired outcome. This typically involves defining a loss function balancing multiple objectives: (1) proximity, ensuring the counterfactual is close to the original observation by minimizing the (e.g., Euclidean, Jaccard, see Chapter **??**) distance between the two; (2) validity, ensuring the counterfactual successfully shifts the prediction to the desired class; and (3) sparsity, encouraging changes in as few features as possible to enhance interpretability. Gradient-based optimization methods are often used when the model is differentiable, allowing iterative updates to the input features until the desired prediction is achieved. For non-differentiable models, heuristic or search-based approaches can be employed [Mothilal et al., 2020]. Constraints can also be integrated into the optimization to ensure feasibility, such as restricting changes to plausible values (e.g., income must remain positive, age is between 18 and 90). These techniques allow counterfactual explanations to be generated effectively across different types of models, providing actionable and interpretable insights [Karimi et al., 2021].

DiCE [Mothilal et al., 2020] extends traditional counterfactual explanations by generating a diverse set of counterfactual examples instead of just one. The key insight is that different users may prefer different ways to achieve the same desired outcome. For example, one loan applicant might prefer to increase their income, while another might find it easier to reduce their debt. DiCE uses a diversity loss function along with the traditional proximity and sparsity objectives to generate multiple counterfactuals that are both feasible and different from each other. It also introduces the concept of feasibility weights to encourage the generation of counterfactuals that respect real-world constraints and are actionable. For instance, when suggesting changes to improve a loan application, DiCE can account for the fact that age cannot decrease and income cannot become negative.

Counterfactuals can be generated for both classification and regression problems. Contrary to LIME and Shapley values, counterfactuals focus on actionability: instead of explaining why a prediction was made, they show how to change it. For example, where SHAP might tell us ”your income had a negative impact of -0.3 on the loan decision” and LIME might say ”low income was the main reason for rejection”, counterfactuals provide actionable feedback like ”increasing your income by $10,000 would change the decision to approval”.

		Original data		Contribution log odds		Points	
	Estimate	**Min**	**Max**	**Min**	**Max**	**Min**	**Max**
Intercept	-2.5						
Income	-0.03	10k	100k	-0.3	-3.0	-7.5	-75
FICO	-0.002	300	850	-0.6	-1.7	-15	-42.5
Debt-to-Income	0.08	0%	50%	0	4	0	100

Table 1.4: Nomogram summary statistics.

1.11 NOMOGRAMS

A nomogram is a graphical XAI tool used to represent a statistical model, such as a linear/ logistic regression, in a way that allows for easy calculation of a predicted outcome [Allcock, 1963] using a visually attractive format. At its core, a nomogram consists of several parallel scales representing different predictors, allowing users to trace lines between them to compute model predictions. Let's assume we start with a logistic regression model as follows:

$$P(\text{Default}) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 \text{Income} + \beta_2 \text{Fico Score} + \beta_3 \text{Debt-to-Income Ratio})}} \tag{1.12}$$

The log odds formulation then becomes:

$$\log(\text{Odds}(\text{Default})) = \beta_0 + \beta_1 \text{Income} + \beta_2 \text{Fico Score} + \beta_3 \text{Debt-to-Income Ratio} \tag{1.13}$$

Let's say the estimated coefficients are: $\beta_0 = -2.5$ (Intercept), $\beta_1 = -0.03$ (Income in \$1000s), $\beta_2 = -0.002$ (Fico Score) and $\beta_3 = 0.08$ (Debt-to-Income Ratio, as a percentage). The predictor ranges are as follows: Income: \$10k to \$100k, Fico score between 300 and 850 and Debt-to-Income ratio between 0% and 50%. In other words, the minimum/maximum contributions to the log odds are: Income ($-0.03 \times 10 = -0.3$, $-0.03 \times 100 = -3.0$), Fico score ($-0.002 \times 300 = -0.6$, $-0.002 \times 850 = -1.7$) and Debt-to-Income ratio ($0.08 \times 0 = 0.0$, $0.08 \times 50 = 4.0$). Debt-to-Income ratio has the highest maximum contribution or range effect of 4. To create a user-friendly point system, we assign 100 points to this maximum range and scale all other contributions proportionally using a scaling factor of $100/4 = 25$. This transformation results in point ranges that maintain the relative impacts of each predictor: Income ranges from -7.5 to -75 points (total range of 67.5 points), Fico score from -15 to -42.5 (total range of 27.5 points), and Debt-to-Income from 0 to 100 points (total range of 100 points). This is summarised in Table 1.4.

The spacing between values on each scale is carefully constructed to preserve the linear relationship in the log odds scale. You can see this illustrated in Figure 1.23. To use the nomogram, one draws straight lines between the parallel scales to compute predictions. The scales are designed so that points can then be summed graphically by following connecting lines as shown in red. The probability scale at the bottom incorporates both the intercept term (β_0) and the logistic transformation, allowing direct reading of predicted probabilities.

Let's assume we now wish to calculate the points for a customer with characteristics, income = \$50k, Fico score = 700 and Debt-to-Income ratio = 40% as highlighted by the red tick marks in Figure 1.23. The total points become: $-0.03 \times 50 \times 25$ (Income) + $-0.002 \times 700 \times 25$ (Fico score) + $0.08 \times 40 \times 25$ (Debt-to-Income) = 7.5. The probability of default can then be computed as:

$$P(\text{Default}) = \frac{1}{1 + e^{-(-2.5 + 7.5/25)}} \tag{1.14}$$

which equals 9.98% as shown on the bottom axis.

Nomograms can also be used to represent linear regression models. The key difference from logistic regression nomograms is that we don't need a logistic transformation at the end - the scales can be linear throughout. This makes linear regression nomograms somewhat simpler to construct and interpret.

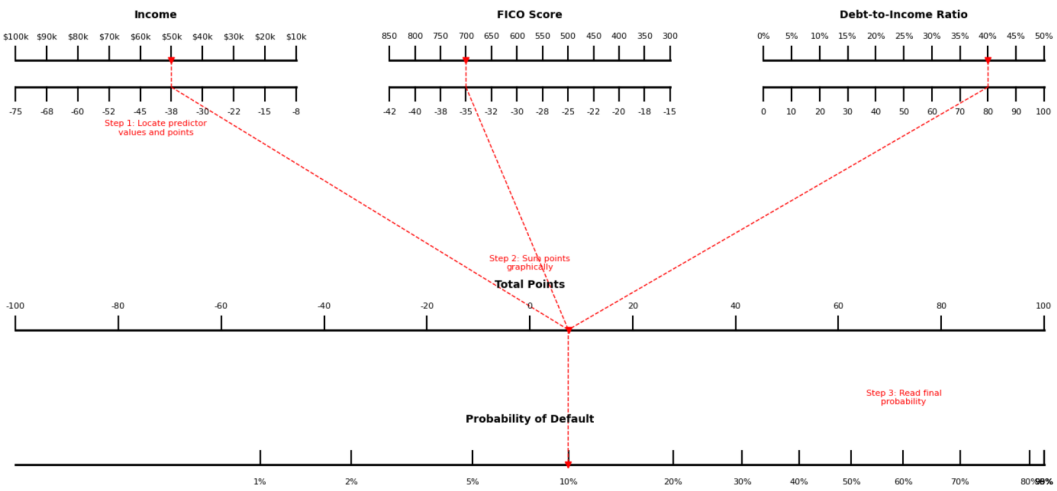


Figure 1.23: Example nomogram.

Nomograms offer several advantages for model interpretation and practical use. Their design allows users to visually understand the relative impact of various predictors, as the length of each scale reflects the predictor's maximum possible contribution to the model. The alignment of scales also enables quick "what-if" analyses by moving connecting lines to different values. To further improve interpretability, color codes can be added, as shown in Figure 1.24 with lighter colors indicating higher probabilities of default [Van Belle and Van Calster, 2015].

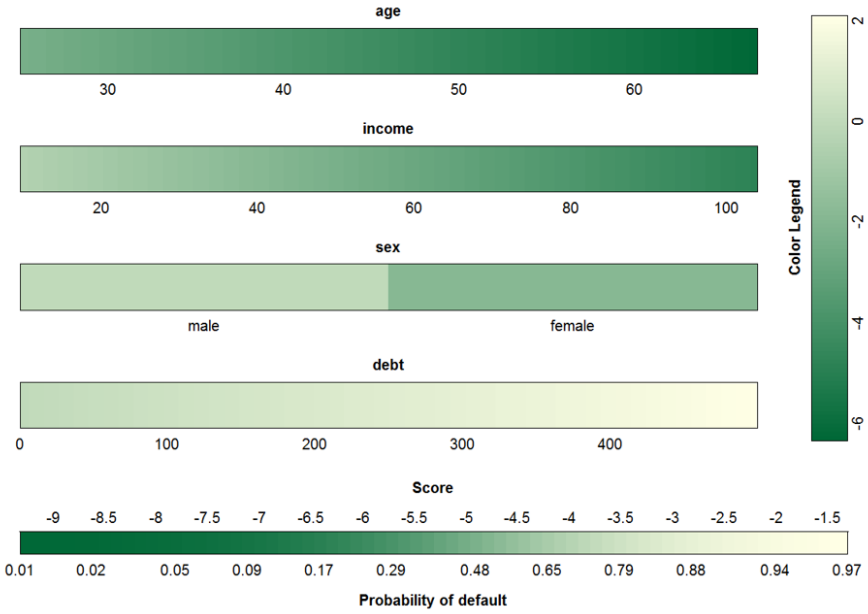


Figure 1.24: Example of colored nomogram.

Nomograms come with several limitations. They are primarily designed for linear and logistic regression models, making them unsuitable for complex non-linear models like neural networks or gradient boosting machines. They also struggle to represent interaction effects between features, as the parallel scales assume feature independence. While nomograms can support "what-if" scenarios through manual adjustment of feature values, counterfactuals automatically identify the minimal changes needed to achieve a desired outcome.

1.12 GRAD-CAM

Gradient-weighted Class Activation Mapping (Grad-CAM) is a powerful visualization technique that helps understand which parts of an input image are most important for a convolutional neural network's (CNN, see section **??**) classification decision [Selvaraju et al., 2017]. The technique works by examining the gradients flowing into the final convolutional layer of the network, which contains rich spatial information about the target object.

The process begins with a forward pass through the network, where the input image is processed through multiple convolutional layers, each followed by a ReLU activation function to introduce non-linearity and emphasize positive feature contributions. As shown in Figure 1.25, each layer extracts increasingly complex features. For a smiley image, early layers might detect basic edges, while deeper layers recognize more complex patterns like shapes and facial features. During this forward pass, ReLU ensures only positive activations are propagated through the network, helping to create sparse, meaningful feature representations.

After obtaining the prediction (say, "Happy smile" with 85% confidence), Grad-CAM calculates the gradients of the target class score with respect to the feature maps of the final convolutional layer as these are the most discriminatory. These gradients are globally average-pooled (meaning each feature map is reduced to a single value by taking the average of all its values) to obtain weights that represent the importance of each feature map for the target class. This averaging process helps identify which feature maps are most important for the target class by summarizing the overall presence of each feature across the entire image. The final step involves multiplying each feature map by its corresponding importance weight and combining them to create a coarse heatmap. A ReLU function is then applied to the weighted sum to eliminate any negative influences, ensuring we only visualize features that positively contribute to the target class prediction. This heatmap is upsampled to the original image size and overlaid on the input image, typically using a color scheme where warmer colors (red) indicate regions of high importance and cooler colors (blue) indicate less important regions. For our smiley example, the resulting visualization shows intense activation around the mouth clearly highlighting the smile.

Prior to Grad-CAM, saliency maps were a fundamental approach to visualization in deep neural networks [Simonyan et al., 2013]. These maps compute the gradient of the output with respect to the input image, highlighting which pixels need to be changed the least to affect the prediction the most. However, vanilla saliency maps often produced noisy visualizations and could be sensitive to small perturbations in the input image, making them less reliable for interpretation purposes. Additionally, they typically highlighted edges rather than the semantic concepts that actually drove the network's decision, limiting their utility for understanding higher-level more meaningful feature representations. Grad-CAM's approach of using the final convolutional layer's feature maps allows it to capture higher-level semantic information while maintaining spatial relationships. This makes it particularly effective for tasks requiring interpretable explanations, such as medical imaging diagnosis or autonomous vehicle decision-making, where understanding the network's focus areas is crucial for building trust and validating model behavior.

Despite its widespread adoption, Grad-CAM has several notable limitations. First, because it relies on the final convolutional layer, it can miss fine-grained details that were captured in earlier layers but lost due to pooling operations, resulting in coarse localization maps that may not precisely identify small or intricate features. Additionally, Grad-CAM can struggle with multi-class scenarios where multiple objects of interest are present in the image, as it tends to focus on the most discriminative features for a single class. The technique also assumes that the importance of a feature map can be adequately represented by a single global weight, which may not hold true when different spatial regions of the same feature map contribute differently to the classification decision [Selvaraju et al., 2017].

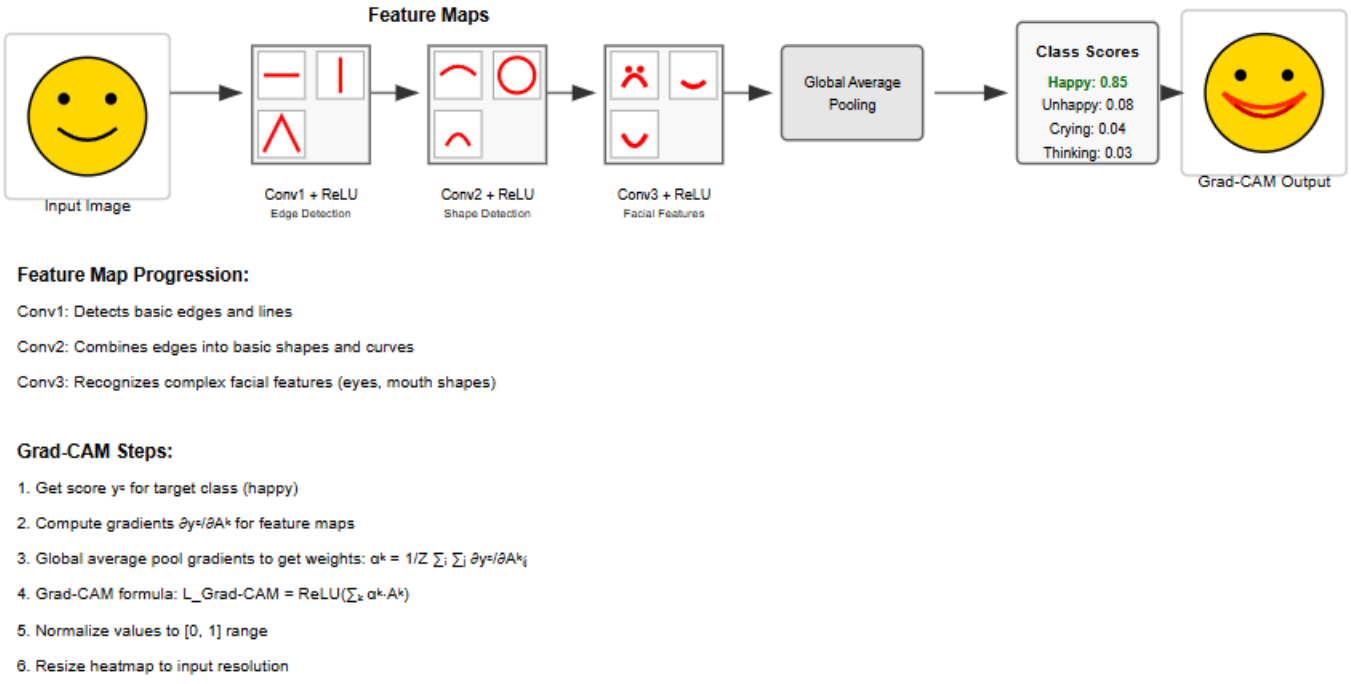


Figure 1.25: Grad-CAM illustrated.

Grad-CAM for Royal Navy warship classification
In [Baesens et al., 2024], we trained convolutional neural networks for classifying images of Royal Navy warships as destroyers, submarines, minesweepers, etc. We used Grad-CAM to facilitate the interpretation of the trained networks. Figure 1.26 displays some examples of Grad-CAM heatmaps for some of the ships in our data set made using Python's grad-cam package.

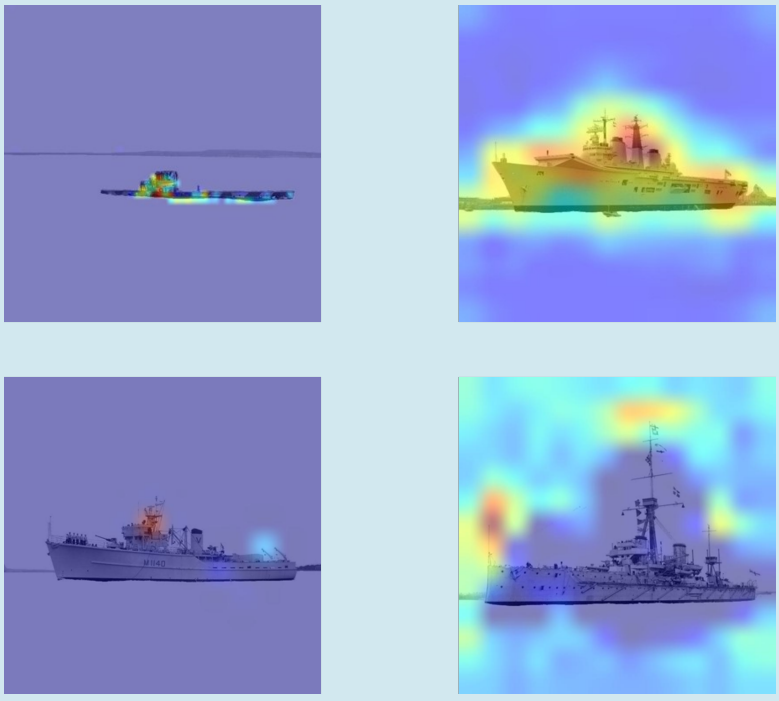


Figure 1.26: Example Grad-CAM images. Top Left: HMS E11 submarine ($\sim$ 1914); Top Right: HMS Ark Royal carrier (1993); Bottom Left: HMS Gavington minesweeper ($\sim$ 1952); Bottom Right: HMS Dreadnought battleship (1906).

1.13 DECISION TABLES

Origin of decision tables
Decision tables were developed in the mid-1950s, with significant work done at General Electric (GE). The primary person credited with inventing decision tables is Charles H. Davidson, who worked at GE and developed them while working on the UNIVAC I computer system. Significant contributions were also made by two of our former KU Leuven colleagues, prof. Maurice Verhelst and prof. Jan Vanthienen.

Decision tables are a very practical rule-representation format with a lot of potential in an XAI setting. A decision table (DT) is a tabular representation that is used to describe and analyze decision situations, where the state of a number of conditions jointly determines the execution of a set of actions. The conditions correspond to the antecedents of the rules whereas the actions correspond to the outcome classes. In the example decision table shown in Figure 1.27, the conditions are: Owns property, Years client and Savings amount. The actions are application is good or applicant is bad. Each condition entry describes a relevant

1. Owns property?	yes				no			
2. Years client	<= 3		> 3		<= 3		> 3	
3. Savings amount	low	high	low	high	low	high	low	high
1. Applicant=good	-	x	x	x	-	x	-	x
2. Applicant=bad	x	-	-	-	x	-	x	-
	1	2	3	4	5	6	7	8

Figure 1.27: Example Decision Table.

subset of values, called a state, for a given condition subject or attribute, or contains a hyphen symbol ('-') if its value is irrelevant within the context of that column.This is an opportunity to simplify our table. In our updated example decision table in Figure 1.28, you can see that when owns property is no, years client becomes irrelevant as indicated by the dash symbol. Every column in the entry part of the DT thus comprises a classification rule, indicating what actions apply to a certain combination of condition states. In our example in Figure 1.28, you can see that the decision table corresponds to five classification rules.

1. Owns property?	yes			no	
2. Years client	<= 3		> 3	-	
3. Savings amount	low	high	-	low	high
1. Applicant=good	-	x	x	-	x
2. Applicant=bad	x	-	-	x	-
	1	2	3	4	5

Figure 1.28: Example Decision Table.

If each column contains only simple states, so no contracted or irrelevant entries, the table is called an expanded DT. Otherwise, the table is called a contracted DT. Table contraction can be achieved by combining columns, which lead to the same action configuration. The number of columns in the contracted table can then be further minimized by changing the order of the conditions. It is obvious that a DT with a minimal number of columns is preferred because it provides a more parsimonious and comprehensible representation of the extracted knowledge than an expanded DT.

In Figure 1.29 you can see an example of a decision table. The table is the fully expanded table. The area above the double-line is the condition part and the area below the double-line is the action part. The decision table models eight classification rules. As an example, IF Owns property is Yes AND Years client $\leq$ 3 AND Savings amount is Low, THEN the Applicant is bad. When you inspect this table more closely, it can be seen that contraction is possible. As an example, consider rules 3 and 4. They both have Owns property = Yes, AND Years client $>$ 3. They only differ in terms of Savings amount. However, both rules classify the Applicant as good, so it means that Savings amount is not relevant to make the classification here. Both rules 3 and 4 can be contracted, which results in a hyphen symbol indicating that Savings amount is irrelevant here. The same applies when you look at rules 5, 6, 7, and 8. Here you can see similar patterns occurring in the action part of the table, indicating that Years client is not relevant, which results in further contraction. This creates the contracted table in Figure 1.30, which has only five rules as opposed to the eight rules that we started from. The decision table can then be further minimized by changing the order of the conditions as shown in Figure 1.31. In other words, by starting from Savings amount, then Owns property, and finally Years client, a decision table with four rules can be obtained. This is a very compact, interpretable and user-friendly representation of the IF-THEN rules. It also shows that if Savings amount is high, then all other conditions become irrelevant and the Applicant will be classified as good.

1. Owns property?	yes				no			
2. Years client	<= 3		> 3		<= 3		> 3	
3. Savings amount	low	high	low	high	low	high	low	high
1. Applicant=good	-	x	x	x	-	x	-	x
2. Applicant=bad	x	-	-	-	x	-	x	-
	1	2	3	4	5	6	7	8

Figure 1.29: Fully expanded Decision Table.

1. Owns property?	yes			no	
2. Years client	<= 3		> 3	-	
3. Savings amount	low	high	-	low	high
1. Applicant=good	-	x	x	-	x
2. Applicant=bad	x	-	-	x	-
	1	2	3	4	5

Figure 1.30: Contracted Decision Table.

1. Savings amount	low			high
2. Owns property?	yes		no	-
3. Years client	<= 3	> 3	-	-
1. Applicant=good	-	x	-	x
2. Applicant=bad	x	-	x	-
	1	2	3	4

Figure 1.31: Minimized Decision Table.

The rule set depicted in Figure 1.32 is a set of IF-THEN rules for credit scoring. Although the rule set looks intuitive, representing it as a decision table as shown in Figure 1.33 adds further interpretability. For

example, as soon as the Savings Account balance is bigger than 12.40 euro, the Applicant is considered good. This is not obvious when you inspect the original rule set.

IF **Term** > 12 Months AND **Purpose** = Cash Provisioning AND **Savings Account** ≤ 12.40 Euro AND **Years Client** ≤ 3 THEN **Applicant** = Bad

IF **Term** > 12 Months AND **Purpose** = Cash Provisioning AND **Owns Property** = No AND **Savings Account** ≤ 12.40 Euro THEN **Applicant** = Bad

IF **Purpose** = Cash Provisioning AND **Income** > 719 Euro AND **Owns Property** = No AND **Savings Account** ≤ 12.40 Euro AND **Years Client** ≤ 3 THEN **Applicant** = Bad

IF **Purpose** = Second-Hand Car AND **Income** > 719 Euro AND **Owns Property** = No AND **Savings Account** ≤ 12.40 Euro AND **Years Client** ≤ 3 THEN **Applicant** = Bad

IF **Savings Account** ≤ 12.40 Euro AND **Economic Sector** = Sector C THEN **Applicant** = Bad

Default class: **Applicant** = Good

Figure 1.32: Example rule set for credit scoring.

1. Savings Account	<= 12.40 Euro												> 12.40 Euro
2. Economical sector	Sector C	other											-
3. Purpose	-	cash provisioning							second-hand car			other	-
4. Term	-	<= 12 months				> 12 months			-			-	-
5. Years client	-	<= 3			> 3	<= 3		> 3	<= 3			> 3	-	-
6. Income	-	<= 719 Euro		> 719 Euro		-		-	<= 719 Euro		> 719 Euro		-	-
7. Owns Property	-	-		Yes	No	-		-	Yes	No			-	-
1. applicant = good	-	x		x	-	x	-	x	-	x	x	-	x	x
2. applicant = bad	x	-		-	x	-	x	-	x	-	-	x	-	-
	1	2		3	4	5	6	7	8	9	10	11	12	13	14

Figure 1.33: Decision table for rule set in Figure 1.32.

In Figure 1.34 you can see another example of a decision table for churn prediction in a Telco setting, again from earlier research as cited below. A very useful feature of decision tables is that they easily enable you to verify the patterns that are extracted from data with the knowledge of the business expert. One way of doing this is by removing all the classifications in the action part of the decision table and presenting it to the business expert as such. You can then ask the business expert to make the classifications according to his or her business knowledge or experience. The obtained classifications can be contrasted with the classifications in the original table as they were obtained from the data. This will enable you to find the unexpected, potentially interesting patterns.

Decision tables are very useful for visualizing rules in an intuitive and user-friendly way and for checking rules for completeness and anomalies. The rules can be obtained in various ways. They can be obtained from a decision tree, inferred from a business expert or originate from both. When you use decision tables, it becomes easy to identify conflicts between different rules. Let's illustrate this with an example. In Figure 1.35 you can see four rules in a churn prediction setting. The first rule is IF Avg Usage $<$ 25 AND Internat Plan = Y AND Service Calls $>$ 3, THEN Churn. Despite the fact that there are only four rules with three conditions at most, it is not that straightforward to verify whether the rules are conflicting or if they include all

1. Savings Account	<= 12.40 Euro													> 12.40 Euro
2. Economical sector	Sector C	other												-
3. Purpose	-	cash provisioning							second-hand car				other	-
4. Term	-	<= 12 months				> 12 months			-				-	-
5. Years client	-	<= 3			> 3	<= 3		> 3	<= 3			> 3	-	-
6. Income	-	<= 719 Euro		> 719 Euro		-	-	-	<= 719 Euro		> 719 Euro		-	-
7. Owns Property	-	-	Yes	No	-	-	Yes	No	-	Yes	No	-	-	-
1. applicant = good	-	x	x	-	x	-	x	-	x	x	-	x	x	x
2. applicant = bad	x	-	-	x	-	x	-	x	-	-	x	-	-	-
	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Figure 1.34: Decision table for churn prediction in Telco.

possible input combinations. To check this, we can represent the rules as a decision table. In Figure 1.36

IF **Avg Usage** < 25 AND **Internat Plan** = Y AND **Service Calls** > 3 THEN **Churn**

IF **Avg Usage** < 25 AND **Internat Plan** = N THEN **Churn**

IF **Avg Usage** ≥ 25 AND **Internat Plan** = Y THEN **Not Churn**

IF **Avg Usage** < 25 AND **Service Calls** ≤ 3 THEN **Not Churn**

Figure 1.35: Rule set for churn prediction in Telco.

you can see the corresponding decision table. It can be easily seen that rules R2 and R4 are conflicting. In the case when Avg Usage < 25, Internat Plan is No, and Service Calls ≤ 3, the rules assign conflicting classes. Clearly, this problem should be solved. In other words, the rule set is not exclusive. Another problem arises in the last two columns. Here, you can see that if Avg Usage is bigger than or equal to 25 and Internat Plan is no, then no classifications are assigned as indicated by the missing entries. This illustrates that the rule set is not exhaustive. In other words, using decision tables, it is very easy to check whether rules are exclusive and exhaustive.

1. Avg Usage	< 25				≥ 25			
2. Internat Plan	Y		N		Y		N	
3. Service Calls	≤ 3	> 3	≤ 3	> 3	≤ 3	> 3	≤ 3	> 3
1. Churn	-	x	x	x	-	-	.	.
2. Not Churn	x	-	x	-	x	x	.	.
Contributing rule(s):	R4	R1	R2 R4	R2	R3	R3		

Figure 1.36: Decision Table for rule set in Figure 1.35.

1.14 SUMMARY

Table 1.5 summarizes the XAI methods discussed in this chapter, categorizing them based on their model-specificity (agnostic or specific) and whether they provide local, global, or combined insights. The table clearly highlights the diversity of XAI tools and emphasizes the trade-offs between local and global interpretability depending on the method used.

Method	Model agnostic/specific	Local versus Global
Traffic light indicator method	Model agnostic	Local
Permutation based feature importance	Model agnostic	Global
Partial Dependence plots	Model agnostic	Global
Individual Conditional Expectation (ICE) plots	Model agnostic	Local + Global
Scorecard approach	Model specific (logistic regression)	Global
Meta decision trees	Model agnostic	Global
LIME	Model agnostic	Local
Anchors	Model agnostic	Local
Shapley	Model agnostic	Local + Global
Counterfactuals	Model agnostic	Local
Nomograms	Model specific (regression models)	Global
Grad-CAM	Model specific (convolutional neural networks)	Local
Decision tables	Model specific (rule based models)	Global

Table 1.5: Explanation methods and their characteristics.

1.15 CONCLUDING REMARKS

Explainable Artificial Intelligence (XAI) plays a critical role in bridging the gap between complex AI/machine learning models and human understanding, fostering trust, transparency, and accountability in AI-driven decisions. This chapter highlighted a variety of XAI techniques, from feature importance methods to local and global explanation approaches such as LIME, Shapley values, and counterfactuals. Methods like meta decision trees and scorecards provide practical ways to make models accessible to non-technical schooled decision makers. Also visual analytics like traffic light approaches, nomograms, Grad-CAM and decision tables can contribute to better interpretation of AI models.

Note that there often exists an inherent trade-off where more explainable models may sacrifice some predictive performance compared to their black-box counterparts. This trade-off is particularly evident in highly complex domains like computer vision and natural language processing, where simpler, interpretable models may struggle to capture intricate patterns. For critical applications in healthcare and finance, practitioners must carefully weigh the benefits of model transparency against potential losses in accuracy.

While XAI methods enhance interpretability, challenges remain, including computational complexity, multicollinearity in features, and the need for domain expertise. Furthermore, it is important to note that any XAI method is always an approximation and hence never a perfect mapping of a model's internal functioning. In other words, model subtleties are often lost when an XAI layer is added on top of it.

Despite these limitations, XAI continues to be a vital area of research and applications, ensuring AI systems are not only accurate but also fair, reliable, and comprehensible. By leveraging these techniques, we can better align AI systems with ethical, regulatory, and business objectives.

Bibliography

Harry John Allcock. *The nomogram: The theory and practical construction of computation charts*. Pitman, London, 1963.

B. Baesens, D. Roesch, and H. Scheule. *Credit Risk Analytics: Measurement Techniques, Applications, and Examples in SAS*. John Wiley & Sons, 2016.

Bart Baesens, Rudy Setiono, Christophe Mues, and Jan Vanthienen. Using neural network rule extraction and decision tables for credit-risk evaluation. *Management Science*, 49(3):312–329, 2003. doi: 10.1287/mnsc.49.3.312.12739.

Bart Baesens, Amy Adams, Rodrigo Pacheco-Ruiz, Ann-Sophie Baesens, and Seppe Vanden Broucke. Explainable deep learning to classify royal navy ships. *IEEE Access*, 12:1774–1785, 2024.

Leo Breiman. Random forests. *Machine learning*, 45:5–32, 2001.

Mark W. Craven and Jude W. Shavlik. Extracting tree-structured representations of trained networks. In *Advances in Neural Information Processing Systems (NIPS)*, volume 8, pages 24–30. MIT Press, 1996.

Jerome H. Friedman. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29 (5):1189–1232, 2001.

Alex Goldstein, Adam Kapelner, Justin Bleich, and Emil Pitkin. Peeking inside the black box: Visualizing statistical learning with plots of individual conditional expectation. *Journal of Computational and Graphical Statistics*, 24(1):44–65, 2015.

Amir-Hossein Karimi, Gilles Barthe, Bernhard Schölkopf, and Isabel Valera. Algorithmic recourse: From counterfactual explanations to interventions. *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency (FAccT '21)*, pages 353–362, 2021.

Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 4768–4777, 2017a.

Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. *Advances in Neural Information Processing Systems*, 30, 2017b.

Scott M Lundberg, Gabriel Erion, and Su-In Lee. From local explanations to global understanding with explainable ai for trees. *Nature machine intelligence*, 2(1):56–67, 2020.

Charles Joseph Minard. Carte figurative des pertes successives en hommes de l'armée française dans la campagne de russie 1812-1813, 1869. Lithograph, 62 x 30 cm.

Christoph Molnar. *Interpretable Machine Learning*. 2 edition, 2022. URL `https://christophm.github.io/interpretable-ml-book`.

Ramaravind K Mothilal, Amit Sharma, and Chenhao Tan. Explaining machine learning classifiers through diverse counterfactual explanations. In *Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency*, pages 607–617. ACM, 2020.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Why should i trust you?: Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1135–1144. ACM, 2016.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Anchors: High-precision model-agnostic explanations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE international conference on computer vision*, pages 618–626, 2017.

Lloyd S. Shapley. A value for n-person games. *Contributions to the Theory of Games (AM-28), Volume II*, 28:307–317, 1953.

Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013.

Vanya Van Belle and Ben Van Calster. Visualizing risk prediction models. *PLOS ONE*, 10:1–16, 07 2015. doi: 10.1371/journal.pone.0132614. URL `https://doi.org/10.1371/journal.pone.0132614`.

Sandra Wachter, Brent Mittelstadt, and Chris Russell. Counterfactual explanations without opening the black box: Automated decisions and the gdpr. *Harvard Journal of Law & Technology*, 31:841–887, 2017.