Artificial Intelligence Seminar

Andrea Masi, Victor Badenas

Master in Artificial Intelligence (MAI)

May 2021

Contents

Chapter 1

Previous Knowledge

Before going into details of all the different implementation and applications of SHAP, it's important give a good explanation about the most important concepts needed to know for understanding why the algorithm works and why it has quickly become one of the most used Explainable Artificial Intelligence method.

1.1 Additive Feature Attribution Methods

The best explanation of a simple model is the model itself; it perfectly represents itself and is easy to understand. For complex models, such as ensemble methods or deep networks, we cannot use the original model as its own best explanation because it is not easy to understand.
Instead, we must use a *simpler explanation model*, that use simplified inputs x' that map to the original inputs through a mapping function $x = h_x(x')$, which we define as any interpretable approximation of the original model.

All additive feature attribution methods have an explanation model that is a linear function of binary variables:

$$g(z') = \phi_0 + \sum_{j=1}^{M} \phi_j z'_j$$

Methods with explanation models matching the previous expression (that will be explained later in section 2.1) attribute an effect ϕ_i to each feature, and summing the effects of all feature attributions approximates the output f(x) of the original model [1].

In other words, we can say that feature attribution refers to the fact that the change of an outcome to be explained (e.g., a class probability in a classification problem) with respect to a baseline (e.g., average prediction probability for that class in the training set) can be attributed in different proportions to the model input features.

1.2 Shapley Values

The Shapley value is a solution concept in cooperative game theory, named after Lloyd Shapley. To each cooperative game it assigns a unique distribution (among the players) of a

total surplus generated by the coalition of players.

For example: a group of players cooperates and obtains a gain from the cooperation. The contribution of the players to the overall gain is not usually uniform, some contribute more than others. The players that contribute more hold more bargaining power (for example threatening to destroy the whole surplus). The Shapley value answers the question of how important is each player to the overall cooperation, and what payoff can he or she expect?

In Machine Learning models, Shapley values can be defined analogously to the above description. In this case, the game would be the prediction task for the instance, the players are the features and finally the gain is the prediction for this instance minus the average prediction for all instances. Shapley values allows us to numerically describe the contribution of the features to an output of a model.

1.2.1 Nomenclature

1. A set of N players
2. A function v that maps subsets of players (simplified features) to real numbers (gains) $v : 2^N \rightarrow \mathbb{R}$, where $v(\emptyset) = 0$.
3. A coalition of players noted S for which $v(S)$ is the worth of the coalition of players and $v(x),\ x = \{p_1, ..., p_n\}\ where\ p_i \in S$ describes the expected payoff for the group of players x.
4. $\phi(v)$ denotes the shapley value for the the gain function v

1.2.2 Computation example

We start from a model trained to predict apartment prices based on 2 features: 'park' (nearby, far), 'area' (numerical), 'floor' (cardinal/numerical) and 'cat' (allowed, banned). The average prediction for all apartments is 310.000€. Let's compute the coalition for the combination 'park-nearby', 'cat-banned' and 'area-50' and let's simulate that the 'floor' variable is not in the coalition. In order to do that we have to perform two steps:

- To simulate the absence of the 'floor' feature, a random instance of the dataset is extracted and the 'floor' attribute is used to predict the first value f.i. 310.000€.
- The 'cat-banned' is removed from the coalition and replaced with a random value of the cat allowed or banned and predict the price f.i. 320.000€.
- Compute the difference between the two values.

The goal is to explain the difference between the instance prediction and the average prediction. For example, to evaluate the contribution of the 'cat-banned' in a coalition with 'park-nearby', 'area-50' and 'floor-2nd'. Given a set of simplified features (each feature is either included or excluded), all combinations of feature coalitions are evaluated with and without the cat-banned feature and the difference is computed. After all the combinations are computed, the Shapley value is the weighted average of all the marginal contributions. The feature values of features that are not in a coalition with random feature values from the apartment dataset to get a prediction from the machine learning model.

1.2.3 Properties

1. **Efficiency**: The sum of the Shapley values of all players equals the value of the grand coalition, so that all the gain is distributed among the agents.

2. **Symmetry**: If i and j are two players who are equivalent in the sense that the addition of each player to a set contributes to the same gain $v(S \cup \{i\}) = v(S \cup \{j\})$, then $\phi_i(v) = \phi_j(v)$ their Shapley values are the same.

3. **Linearity**: Shapley values have to satisfy:

   - $\phi_i(v + w) = \phi_i(v) + \phi_i(w)$
   - $\phi_i(av) = a * \phi_i(v)$

4. **Null player**: If the gain is the same for a player group independent if another player is in the group, the Shapley value for that player is 0. If $v(S \cup \{i\}) = v(S)$ then $\phi_i(v) = 0$

5. **Anonymity**: The labeling of the players doesn't affect the assignation of their gain

6. **Marginalism**: The Shapley value can be defined as a function which uses only the marginal contributions of player i as the arguments.

1.2.4 Advantages

The main advantages of the Shapley values are:

1. The difference between the prediction and the average prediction is **fairly distributed**. This property distinguishes the Shapley value from other methods such as LIME. LIME does not guarantee that the prediction is fairly distributed among the features. The Shapley value might be the only method to deliver a full explanation. In situations where the law requires explainability the Shapley value might be the only legally compliant method, because it is based on a solid theory and distributes the effects fairly.

2. The Shapley value allows **contrastive explanations**. Instead of comparing a prediction to the average prediction of the entire dataset, you could compare it to a subset or even to a single data point. This contrastiveness is also something that local models like LIME do not have.

3. The Shapley value is the only explanation method with a **solid theory**. Methods like LIME assume linear behavior of the machine learning model locally, but there is no theory as to why this should work.

1.2.5 Disadvantages

The main disadvantages of Shapley values are:

1. The Shapley value requires **a lot of computing time**. In 99.9% of real-world problems, only the approximate solution is feasible. An exact computation of the Shapley value is computationally expensive because there are 2^k possible coalitions of the feature values and the "absence" of a feature has to be simulated by drawing random instances, which increases the variance for the estimate of the Shapley values estimation. The exponential number of the coalitions is dealt with by sampling coalitions and limiting the number of iterations M. Decreasing M reduces computation time, but increases the variance of the Shapley value. There is no good rule of thumb for the number of iterations M. M should be large enough to accurately estimate the Shapley values, but small enough to complete the computation in a reasonable time.

2. The Shapley value **can be misinterpreted**. The Shapley value of a feature value is not the difference of the predicted value after removing the feature from the model training. The interpretation of the Shapley value is: Given the current set of feature values, the contribution of a feature value to the difference between the actual prediction and the mean prediction is the estimated Shapley value.

3. The Shapley value returns a simple value per feature, but **no prediction model** like LIME. This means it cannot be used to make statements about changes in prediction for changes in the input, such as: "If I were to earn €300 more a year, my credit score would increase by 5 points."

4. Like many other permutation-based interpretation methods, the Shapley value method suffers from **inclusion of unrealistic data** instances when features are correlated. To simulate that a feature value is missing from a coalition, we marginalize the feature.

1.2.6 Estimation

To estimate a Shapley value, all the sets of features have to be evaluated with and without the feature to extract the Shapley value. The computation of the exact solution grows exponentially with the number of features so usually a Monte-Carlo sampling is used:

$$\hat{\phi}_j = \frac{1}{M} \sum_{m=1}^{M} \left(\hat{f}(x_{+j}^m) - \hat{f}(x_{-j}^m) \right) \tag{1.1}$$

where $\hat{f}(x_{+j}^m)$ is the prediction for x, but with a random number of feature values replaced by feature values from a random data point z, except for the respective value of feature j. The x-vector x_{-j}^m is almost identical to x_{+j}^m, but the value x_j^m is also taken from the sampled z.

1.2.7 Single Shapley value estimation

Algorithm 1:

Result: Shapley value for the value of the j-th feature
initialization;
Required: Number of iterations M, instance of interest x, feature index j, data matrix X, and machine learning model f

for *m=1,...,M:* **do**
Draw random instance z from the data matrix X;
Choose a random permutation o of the feature values;
Order instance x: $x_o = (x_{(1)}, \ldots, x_{(j)}, \ldots, x_{(p)})$;
Order instance z: $z_o = (z_{(1)}, \ldots, z_{(j)}, \ldots, z_{(p)})$;
Construct two new instances:

- With feature j: $x_{+j} = (x_{(1)}, \ldots, x_{(j-1)}, x_{(j)}, z_{(j+1)}, \ldots, z_{(p)})$
- Without feature j: $x_{-j} = (x_{(1)}, \ldots, x_{(j-1)}, z_{(j)}, z_{(j+1)}, \ldots, z_{(p)})$

Compute marginal contribution: $\phi_j^m = \hat{f}(x_{+j}) - \hat{f}(x_{-j})$;
end
Compute Shapley value as the average: $\phi_j(x) = \frac{1}{M} \sum_{m=1}^{M} \phi_j^m$

Chapter 2

SHAP

SHAP (SHapley Additive exPlanations) by Lundberg and Lee [1] is a method to explain individual predictions. SHAP is based on the previously explained game theoretically optimal Shapley values. The SHAP authors proposed **Kernel SHAP**, an alternative, kernel-based estimation approach for Shapley values inspired by LIME and the local surrogate models. They also proposed **Tree SHAP**, an efficient estimation approach for tree-based models.

In this chapter we will discuss the definition of SHAP as well as the two main variants proposed by the authors: Kernel SHAP and Tree SHAP. Additionally we will be explaining **Linear SHAP** and **Deep SHAP** implementations.

2.1 Definition

The goal of **SHAP** is to explain the prediction of an instance x_i by computing the contribution of each feature to the prediction. The SHAP explanation is based on the Shapley values, where the feature values of a data instance act as players in a coalition. Shapley values tell us how to fairly distribute the gain (the model's prediction in this case) among the features. A player can be an either an individual feature value or a group of feature values. For example in the case of images, pixels are grouped to super pixels or regions that will be added or removed from the simplified feature vector. One innovation of SHAP is that the Shapley value explanation is represented as an additive feature attribution method, a linear model defined as:

$$g(z') = \phi_0 + \sum_{j=1}^{M} \phi_j z'_j$$

where g is the explanation model, $z' \in \{0,1\}^M$ is the simplified feature vector, M is feature combinations and ϕ_j is the Shapley value for the feature combination j.

The simplified feature vector $z_j \in \{0,1\}^M$ contains the information for the inclusion or exclusion of dataset features. In the simplified feature vector, 1 means that the corresponding feature value is "present" and 0 that it is "absent". To compute the Shapley values for

a given instance, we will have to set some features to present and some to absent.

2.2 How to interpret the plots

This section is very visual and dependant on the python package. Because of that, we propose to read the following notebook as a demonstration for this step.

2.3 Model-Agnostic Approximation: Kernel SHAP

KernelSHAP estimates for an instance x the contributions of each feature value to the prediction.

KernelSHAP consists of 5 steps:

- Sample coalitions $z'_k \in \{0,1\}^M, \quad k \in \{1,\dots,K\}$ (1 = feature present in coalition, 0 = feature absent).
- Get prediction for each z'_k by first converting z'_k to the original feature space and then applying model f: $f(h_x(z'_k))$
- Compute the weight for each z'_k with the SHAP kernel.
- Fit weighted linear model.
- Return Shapley values ϕ_k, the coefficients from the linear model.

We can create a random coalition by generating a binary vector $z \in \{0,1\}^P$ where P is the number of features in the dataset. For example, the vector of (1,0,1,0) means that we have a coalition of the first and third features. The K sampled coalitions become the dataset for the regression model. The target for the regression model is the prediction for a coalition. However, the model itself has not been trained in with the simplified features, they need to be translated to the dataset's domain. To do that, we need a function that performs this operation. That operation will be denoted $h_x(z') = z$, where $h_x : \{0,1\}^M \to \mathbb{R}^p$.

The function h_x maps 1's to the corresponding value from the instance x that we want to explain and it maps 0's to a random instance of the dataset. This means that we consider "feature value is absent" and "feature value is replaced by random feature value from data" as equal statements. For instance, if start from the iris dataset and extract an instance such as:

$$x_i = \begin{bmatrix} sepal\_length \\ sepal\_width \\ petal\_length \\ petal\_width \end{bmatrix} = \begin{bmatrix} 5.1 \\ 3.5 \\ 1.4 \\ 0.2 \end{bmatrix}$$

and we try to find the coalitions for sepal_length and petal_length. In this case z_i would be (1, 0, 1, 0) and the features values for the dataset would be (5.1, *random_value_from_dataset*, 1.4, *random_value_from_dataset*).

Sampling from the marginal distribution means ignoring the dependence structure between present and absent features. So we are assuming that the variables are independent between them. The estimation puts too much weight on unlikely instances and results can become unreliable. If the absent feature values would be sampled from the conditional distribution, then the resulting values are no longer Shapley values.

The big difference to LIME is the weighting of the instances in the regression model. LIME weights the instances according to how close they are to the original instance. The more 0's in the coalition vector, the smaller the weight in LIME. SHAP weights the sampled instances according to the weight the coalition would get in the Shapley value estimation. Small coalitions (few 1's) and large coalitions (i.e. many 1's) get the largest weights. The intuition behind it is:

- We learn most about individual features if we can study their effects in isolation.
- If a coalition consists of a single feature, we can learn about the features' isolated main effect on the prediction.
- If a coalition consists of all but one feature, we can learn about this features' total effect (main effect plus feature interactions).
- If a coalition consists of half the features, we learn little about an individual features contribution, as there are many possible coalitions with half of the features.

To achieve Shapley compliant weighting, the following SHAP kernel is used:

$$\pi_x(z') = \frac{(M-1)}{\binom{M}{|z'|}|z'|(M-|z'|)}$$

Where M is the number of possible features in the dataset. $|z'|$ represents the number of present features in instance z'.

We provide an example notebook for the KernelSHAP. In the notebook includes an SVM training of the well known Iris dataset and the explanation with `KernelExplainer` from the `shap` python package.

2.4 Model-Dependant Approximations

Kernel SHAP improves the sample efficiency of model-agnostic estimations of SHAP values, so restricting the attention to specific model types helps develop faster model-specific approximation methods.

2.4.1 Linear SHAP

One of the simpler SHAP approximations for a specific model is the *Linear SHAP*. For linear models, if we assume input feature independence, SHAP values can be approximated directly

from the model's weight coefficients. The single feature representation of a Linear Regressor follows the following expression:

$$f(x) = \sum_{j=1}^{M} w_j x_j + b$$

From that expression, we can infer that the bias in the expression is the ϕ_0 of the shapley value computation as it is the only value not dependant on the input feature.

$$\phi_0(f, x) = b$$

Also, we can infer that the value for each shapley value is then computed as prediction for which the equation has been translated to assign a shapley value of 0 to the mean of the feature's value. To do that, a 0 bias version of the model is used with the mean of the feature subtracted to the input.

$$\phi_i(f, x) = w_j(x_j - E[x_j])$$

These equations lead to an intuitive explanation of a linear model in which each shapley value denotes the contribution of each feature to the output of the model with respect to the mean of the dataset.

An implementation of the method can be found in `Explainers.Linear` in the `shap` python package. Using this library we created a notebook with an example using a Linear Regression model.

2.4.2 Tree SHAP

Several common feature attribution methods for tree ensembles are inconsistent, meaning they can lower a feature's assigned importance when the true impact of that feature actually increases. This can prevent the meaningful comparison of feature attribution values. In contrast, SHAP values consistently attribute feature importance, better align with human intuition, and better recover influential features [2].
Therefore, in 2018 Tree SHAP was proposed, as the first algorithm to compute exact SHAP values for tree-based machine learning models such as decision trees, random forests and gradient boosted trees.

SHAP values are theoretically optimal, but like other model agnostic feature attribution methods, they can be challenging to compute, Tree SHAP reduces the complexity of computing exact SHAP values from $O(TL2^M)$ to $O(TLD^2)$ where 'T' is the number of trees, 'L' is the maximum number of leaves in any tree, 'M' is the number of features, and 'D' is the maximum depth of any tree.

This exponential reduction in complexity allows predictions from previously intractable models with thousands of trees and features to now be explained in a fraction of a second.

We can compute the SHAP values for a tree applying algorithm 2 by estimating $E[f(x)|x_S]$ and then using Equation 1.1. It finds $E[f(x)|x_S]$ by recursively following the decision path

for x if the split feature is in S, and taking the weighted average of both branches if the split feature is not in S:

Algorithm 2:

Result: Estimating $E[f(x)|x_S]$ in $O(TL2^M)$

1: **procedure** EXPVALUE(x, S, $tree = \{v, a, b, t, r, d\}$)
2: **procedure** G(j) ▷ Define the G procedure which we will call on line 10
3: **if** $v_j \neq internal$ **then** ▷ Check if node j is a leaf
4: **return** v_j ▷ Return the leaf's value
5: **else**
6: **if** $d_j \in S$ **then** ▷ Check if we are conditioning on this feature
7: **return** G(a_j) **if** $x_{d_j} \leq t_j$ **else** G(b_j) ▷ Use the child on the decision path
8: **else**
9: **return** $[\text{G}(a_j) \cdot r_{a_j} + \text{G}(b_j) \cdot r_{b_j}]/r_j$ ▷ Weight children by their coverage
10: **return** G(1) ▷ Start at the root node

In the previous algorithm *tree* contains the information of the tree:

- 'v' is a vector of node values;
- For internal nodes, we assign the value '*internal*';
- The vectors 'a' and 'b' represent the left and right node indexes for each internal node;
- The vector 't' contains the thresholds for each internal node, and d is a vector of indexes of the features used for splitting in internal nodes;
- The vector 'r' represents the cover of each node (i.e., how many data samples fall in that sub-tree).

This algorithm is actually a simpler and slower version of another more complex. The latter achieves same results but in polynomial time instead of exponential time and with just some main differences in the implementation.
The intuition of the polynomial time algorithm is to recursively keep track of what proportion of all possible subsets flow down into each of the leaves of the tree. This is similar to running algorithm 2 simultaneously for all 2^M subsets S in Equation 1.1.

Also for Tree SHAP, we provide an example notebook, where we include a practical example and some of the most interesting new data visualization belonging to the implementation of Tree Shap in `TreeExplainer` `shap` python package.

2.4.3 Deep SHAP (DeepLIFT + Shapley values)

As mentioned before, model-dependant approximations like Deep SHAP promise to improve computational performance while extracting extra knowledge on the compositional nature of deep networks. Deep SHAP incorporates another algorithm, DeepLIFT (Learning Important FeaTures), that also tries to interpret neural networks' "black box" nature, improving its flexibility and reliability.

DeepLIFT is a recursive prediction explanation method for deep learning presented in 2017 by A. Shrikumar et al.. It attributes to each input x_i a value $C_{\Delta x_i \Delta t}$, called contribution scores, that represents the effect of that input being set to a reference value as opposed to its original value. This means that for DeepLIFT, the mapping $x = h_x(x')$ converts binary values into the original inputs, where 1 indicates that an input takes its original value, and 0 indicates that it takes the reference value [1]. In other words, DeepLIFT aims to explain the difference in the output from some reference output in terms of the difference of the input from some reference input. If the output of a neuron for a given input has a difference Δ_t with its reference output, we can assign contribution scores to the differences of the activations of neurons in any intermediate layer with their reference state (i.e. Δx_i) such as following '*summation-to-delta*' property [3]:

$$\sum_{i=1}^{n} C_{\Delta x_i \Delta t} = \Delta_t \tag{2.1}$$

Since DeepLIFT is an additive feature attribution method that satisfies local accuracy and missingness, we know that Shapley values represent the only attribution values that satisfy consistency. Therefore, adapting DeepLIFT to become a compositional approximation of SHAP values, leading to Deep SHAP was a simple consequence.

In short, Deep SHAP can efficiently compute SHAP values for the simple network components if they are linear functions, max pooling, or an activation function with just one input, this composition rule enables a fast approximation of values for the whole model. Deep SHAP avoids the need to heuristically choose ways to linearize components. Instead, it derives an effective linearization from the SHAP values computed for each component [1].

Finally for Deep SHAP, we prepared an example notebook, where we include a practical example and some of the most interesting new data visualization belonging to the implementation of Deep Shap in `DeepExplainer` `shap` python package.

Chapter 3

Diary

In this chapter we will document the time and task management strategies used during the development and the research for this project. The work done for the subject has been divided in the following subtasks:

1. Previous Knowkledge (Table 3.2)
   (a) Additive Feature Attribution Method
   (b) Shapley Values
2. SHAP (Table 3.3)
   (a) Definition
   (b) How to interpret the plots
   (c) Model Agnostic Approximation: Kernel SHAP
   (d) Model dependant Approximation:
      i. Linear SHAP
      ii. Tree SHAP
      iii. Deep SHAP
   (e) Jupyter Notebooks [4]:
      i. Kernel SHAP
      ii. Linear SHAP
      iii. Tree SHAP
      iv. Deep SHAP
      v. Other datasets and alternative explanations

In the following tables we document the time assignations done for each of the tasks. As a rough estimation, we started to develop the project on May 29^{th} and we segmented the work as represented in Table 3.1. The times devoted to each task are documented in Table 3.2 and Table 3.3 where NA means not applicable and 1d of work refers to a working day (8h).

Week	Global Task
29/03 - 4/04	Background Research
5/04 - 18/04	Main document writing and explanations
18/04 - 25/04	Jupyter Notebooks and sample code
26/04 - 2/05	Presentation

Table 3.1: Week-wise planning and task asignation

Previous Knowledge Subtask	Asignee	Documentation Time	Explanation Time	Presentation Time
Additive Feature Attribution Method	Andrea Masi	1d	2h	2h
Shapley Values	Victor Badenas	3d	2d	1d

Table 3.2: Diary: Previous Knowkledge

SHAP Subtask	Asignee	Documentation Time	Explanation Time	Presentation Time
Definition	Victor Badenas	2d	3h	2h
How to interpret the plots Notebook	Victor Badenas	1h	3h	1h
Model Agnostic Approximation: Kernel SHAP	Victor Badenas	2d	1d	2h
Kernel SHAP Notebook	Victor Badenas	NA	4h	NA
Model dependant Approximation: Linear SHAP	Andrea Masi	1d	1h	1h
Linear SHAP Notebook	Victor Badenas	NA	2h	NA
Model dependant Approximation: Tree SHAP	Andrea Masi	4d	5h	1h
Tree SHAP Notebook	Andrea Masi	NA	3h	NA
Tree SHAP example Notebook	Andrea Masi	NA	3h	NA
Model dependant Approximation: Deep SHAP	Andrea Masi	4d	5h	1h
Deep SHAP Notebook	Andrea Masi	NA	1d	NA
New Datasets and Alternative Experiments: Kernel SHAP	Victor Badenas	NA	2h	NA
New Datasets and Alternative Experiments: Linear SHAP	Victor Badenas	NA	3h	NA
New Datasets and Alternative Experiments: Tree SHAP	Andrea Masi	NA	1h	NA
New Datasets and Alternative Experiments: Deep SHAP	Andrea Masi	NA	3h	NA

Table 3.3: Diary: SHAP

3.1 Conclusions

The SHAP Explainers and alternatives are very useful in order to properly explain the reasoning behind a prediction of a model. It is quite computationally expensive to compute, however it delivers very accurate and visual results for the user to interpret. We find the visual tools in the SHAP python package in particular very useful to include non numerical explanations in papers and articles, making them easier on the eyes as well as provide some meaningful insight on the model. This will disambiguate some of the aspects of a ML model making it more understandable and clear.

Additionally, explainers like the `DeepExplainer` on the SHAP python package can be very useful for detecting why a classification model, for instance, is performing the classifications in a certain way. The visual tool in the python package also gives very interesting information, where each of the classes can be seen to detect a pattern that can be easily identified in the plots.

Other kids of explainers can be really useful as well to debug the correct functionality of models that have not been tested exhaustively in a certain domain, allowing the researcher to make sure that the models are good, not only in the domain that they are being tested

on, but in a more general scenario.

We also would like to mention that one particular and very interesting explainer has not been exploited due to lack of time, `KernelExplainer` (This allows an entire dataset to be used as the background distribution, instead of just a reduced number of samples).

Finally we would like to state that mostly for sure in our future works we will make use of SHAP for performing explanations and make our models more interpretable and make our findings more concise.

Bibliography

[1] Scott M. Lundberg and Su-In Lee. "A Unified Approach to Interpreting Model Predictions". In: (2017).

[2] Gabriel G. Erion Scott M. Lundberg and Su-In Lee. "Consistent Individualized Feature Attribution for Tree Ensembles". In: (2018).

[3] Avanti Shrikumar - Peyton Greenside - Anshul Kundaje. "Learning Important Features Through Propagating Activation Differences". In: (2019).

[4] Andrea Masi & Victor Badenas. *Google Drive notebook repositories*. URL: https://drive.google.com/drive/folders/1oNLP-BD9QhU7RIF0OIavjRBHBIdt2kmW?usp=sharing.

[5] Scott M. Lundberg et al. "Explainable AI for Trees: From Local Explanations to Global Understanding". In: (2019).

[6] Scott Lundberg. *An introduction to explainable AI with Shapley values*. URL: https://shap.readthedocs.io/en/latest/example_notebooks/overviews/An%20introduction%20to%20explainable%20AI%20with%20Shapley%20values.html.

[7] Christoph Molnar. *Interpretable ML book - Chapter 5.9: Shapley Values (2021)*. URL: https://christophm.github.io/interpretable-ml-book/shapley.html.

[8] Christoph Molnar. *Interpretable ML book - Chapter 5.10: Shap (2021)*. URL: https://christophm.github.io/interpretable-ml-book/shap.html.

[9] Christoph Molnar. *Interpretable ML book - Chapter 5.10: Kernel Shap (2021)*. URL: https://christophm.github.io/interpretable-ml-book/shap.html#kernelshap.

[10] Christoph Molnar. *Interpretable ML book - Chapter 5.10: Tree Shap (2021)*. URL: https://christophm.github.io/interpretable-ml-book/shap.html#treeshap.

[11] Janis Klaise & Alexandru Coca. *Tree SHAP/Theoretical overview*. URL: https://github.com/SeldonIO/alibi/blob/a8efe1bdd8164bef28c3c22687caf4231ce587a5/doc/source/methods/TreeSHAP.ipynb.

[12] Hugh Chen & Scott Lundberg & Su-In Lee. *Understanding Shapley value explanation algorithms for trees*. URL: https://hughchen.github.io/its_blog/index.html.

[13] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. "Learning important features through propagating activation differences". In: *International Conference on Machine Learning*. PMLR. 2017, pp. 3145–3153.

[14] Scott Lundberg. *Shap Python package*. URL: https://github.com/slundberg/shap.

[15] 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.

[16] Scott M. Lundberg et al. "From local explanations to global understanding with explainable AI for trees". In: *Nature Machine Intelligence* 2.1 (2020), pp. 2522–5839.

[17] Scott M Lundberg et al. "Explainable machine-learning predictions for the prevention of hypoxaemia during surgery". In: *Nature Biomedical Engineering* 2.10 (2018), p. 749.