Exercise 5

Shapley

Shapley values tell us how to distribute the "payout" among features fairly. In this exercise, we are going to implement methods to calculate the Shapley values in a theoretical and a practical setting. After finishing this exercise, you have a fundamental understanding of how Shapley works, how to implement a basic version, and how to interpret the results.

- You can get a bonus point for this exercise if you pass at least 85% of the tests. Code is automatically tested after pushing. If your tests fail, either your code is wrong, or you solved the task differently. In the second case, we will manually check your solution.
- Three collected bonus points result in a 0.33 increase of the final grade.
- You are allowed to work in groups with up to three people. You do not need to specify your ErgebnisPIN as it is already collected from the entrance test.
- Follow the 'README.md' for further instructions.
- Finish this exercise by 17th November, 2021 at 11:59 pm.

---

1 Cooperative Games

To understand the basics of Shapley, we implement the original Shapley algorithm for cooperative games. Given a set of players P with p players, important notations are explained briefly:

- $S \subseteq P$ forms a coalition with $0 \leq |S| \leq p$ players.
- $v(S) : 2^{|P|} \mapsto R$ describes the payout achieved by coalition S.
- ϕ_j is the shapley value of player $j \in P$.

Associated file: *tasks/cooperative_game.py*.

1.1 Original Implementation

Return the shapley value based on the original implementation from player $j \in P$ in function *get_shapley* with the following equation:

$$\phi_j = \sum_{S \subseteq P - \{j\}} \frac{|S|!(|P| - |S| - 1)!}{|P|!}(v(S \cup \{j\}) - v(S)). \tag{1}$$

Make sure you round the value. For testing purposes, also all combinations without player j should be returned.

1.2 Implementation via Orders

Next, we want to calculate the shapley value via order permutations. Use the following equation the complete function *get_shapley_by_order*:

$$\phi_j = \frac{1}{|P|!} \sum_{\tau \in \Pi} (v(Pre(\tau, j) \cup \{j\}) - v(Pre(\tau, j))). \tag{2}$$

$Pre(\tau, j)$ is the set of players before player j joins the coalition. The order is given by τ. Since mathematically speaking, Equation 1 and 2 are the same, the same result is expected. To that means, also round the value exactly as you did before.

To identify if you solved the task via orders, return a list of sets, including all players, before j is added for all permutation orders. Abort the algorithm after M steps.

1.3 Axioms

In this subsection, we want to identify whether the Shapley values or specific sets fulfill the axioms. Please complete the functions *check_symmetry*, *check_dummy*, *check_additivity*, *check_efficiency*, all of which return a Boolean value.

1.3.1 Symmetry

Two features with the same contribution have the same payouts. For $S \subseteq P - \{j, k\}$ and $v(S \cup \{j\}) - v(S) = v(S \cup \{k\}) - v(S)$, the following equation is satisfied:

$$v(S \cup \{j\}) - v(S) = v(S \cup \{k\}) - v(S). \tag{3}$$

1.3.2 Dummy

Player $j \in S \subseteq P$ with no contribution has also no payout:

$$v(S) = v(S \cup \{j\}) \tag{4}$$

1.3.3 Additivity

If two games v_1 and v_2 are the result of v then the two equations with $S \subseteq P - \{j\}$ are fulfilled:

$$v(S) = v_1(S) + v_2(S) \tag{5}$$

$$v(S \cup \{j\}) - v(S) = v_1(S \cup \{j\}) - v_1(S) + v_2(S \cup \{j\}) - v_2(S) \tag{6}$$

1.3.4 Efficiency

Player contributions add up to the total payout of the game:

$$\sum_{j=1}^{p} \phi_j = v(P) \tag{7}$$

2 Model-Agnostic

The implementation from Task 1 is working nicely but only for sets. In this section, we implement the Shapley algorithm for all predictive models using the estimation implementation.
Associated file: *tasks/model_agnostic.py*.

2.1 Preparation

The estimation algorithm is based on order permutations. In the function *merge*, we are implementing a step, which is called later on. Having two arrays $A = (a_1, \ldots, a_p)$ and $B = (b_1, \ldots, b_p)$, make sure you achieve the following behavior:

$$\mathbf{x}_{+j} = (a_1, \ldots, a_{j-1}, a_j, b_{j+1}, \ldots, b_p) \tag{8}$$

$$\mathbf{x}_{-j} = (a_1, \ldots, a_{j-1}, b_j, b_{j+1}, \ldots, b_p) \tag{9}$$

2.2 Estimation Algorithm

Perform now the estimation algorithm for an arbitrary model. In every step you have to sample a random instance $z \in X$ using *np.random.choice*. Use then *np.random.permutation* to get a permutation order for both given point x and random instance z. Those two steps are important to ensure correct testing. Otherwise follow the steps on Slide 7 in the lecture and use the function *merge* from the previous task.

Also, integrate M in your algorithm. M aborts the algorithm prematurely after M steps. If M is not given use the number of entries in X or return zero if no steps are requested.

Warning: Remember to permutate the arrays back before predicting.