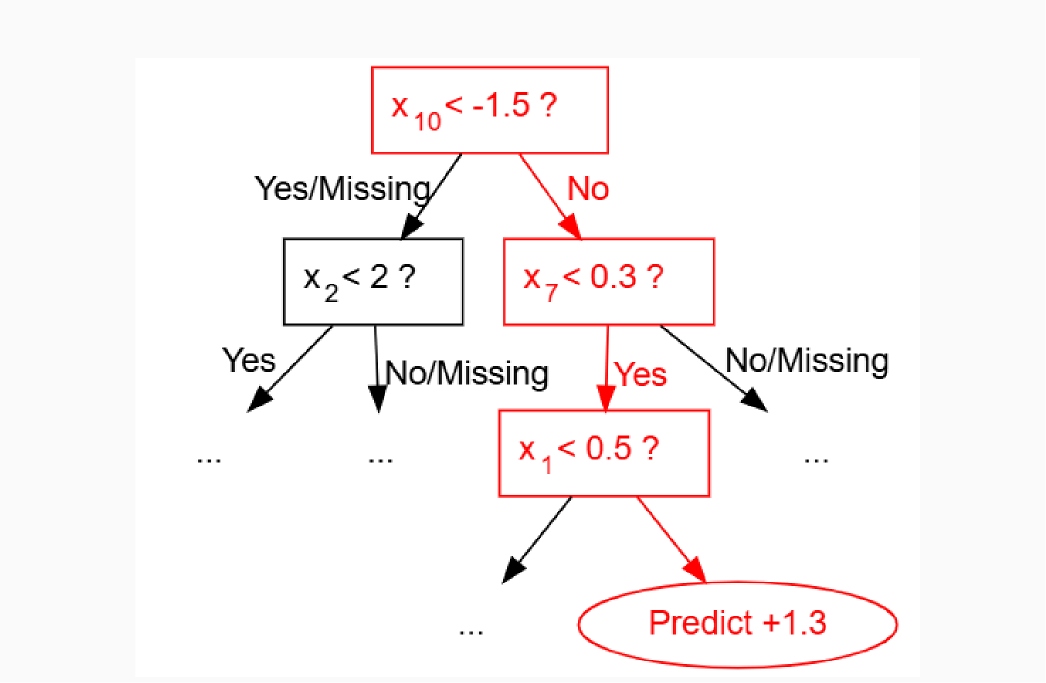
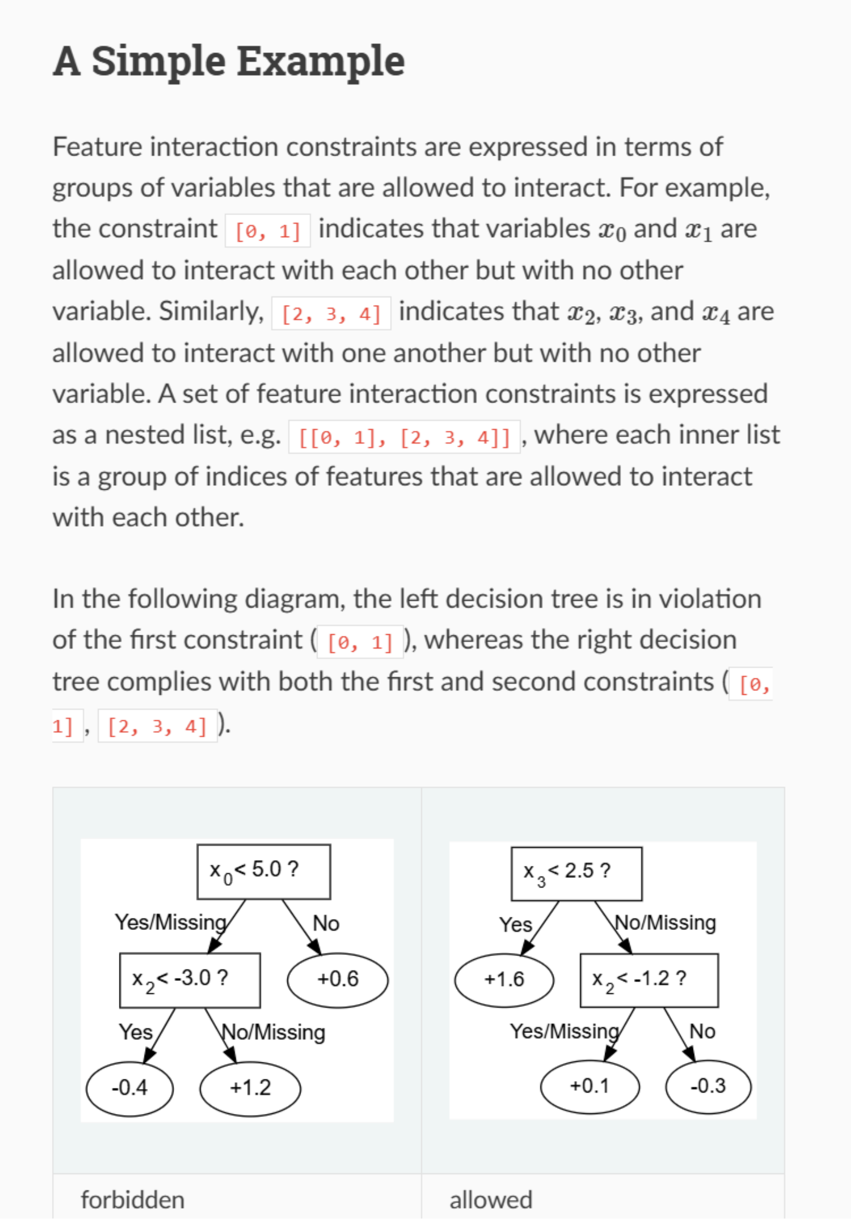
Feature Interaction Constraints

- Decision tree used to discover interaction among independent variables
    - Variables on the same path interact with each other since the condition of a child is predicted based on the condition of the parent



- When the tree depth is larger than one, variables may react on the basis of minimizing training loss
- Feature interaction constraints allow users to choose which variables are allowed to interact
- Benefits
    - Predictive performance
    - Less noise, better generalization
    - More control for the user, they can exclude certain things

A Simple Example



A Simple Example

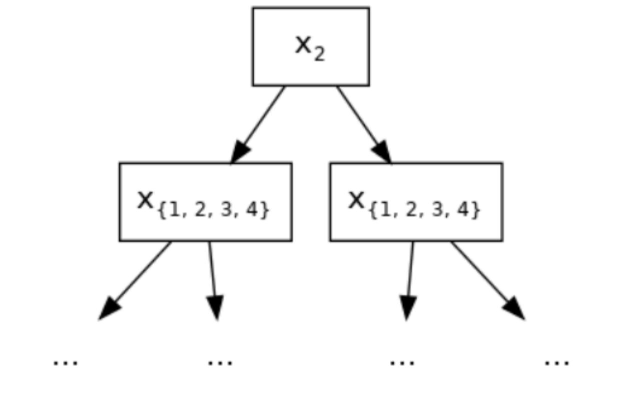
Feature interaction constraints are expressed in terms of groups of variables that are allowed to interact. For example, the constraint `[0, 1]` indicates that variables x_0 and x_1 are allowed to interact with each other but with no other variable. Similarly, `[2, 3, 4]` indicates that x_2, x_3, and x_4 are allowed to interact with one another but with no other variable. A set of feature interaction constraints is expressed as a nested list, e.g. `[[0, 1], [2, 3, 4]]`, where each inner list is a group of indices of features that are allowed to interact with each other.

In the following diagram, the left decision tree is in violation of the first constraint (`[0, 1]`), whereas the right decision tree complies with both the first and second constraints (`[0, 1]`, `[2, 3, 4]`).

forbidden

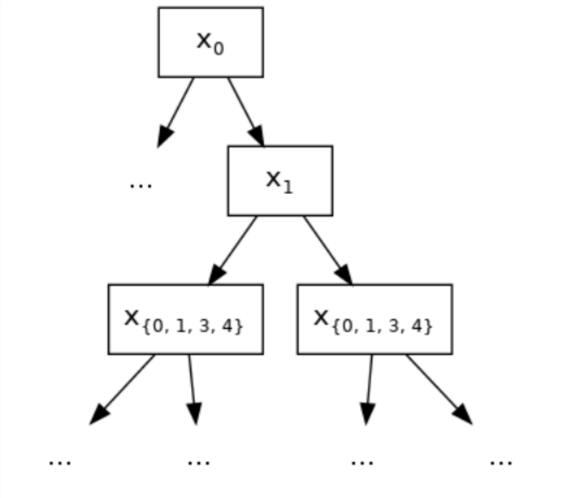
allowed

- In XGB, `params_constrained['interaction_constraints'] = '[[0, 2], [1, 3, 4], [5, 6]]'`
- This constrains the parameters from the list of params accordingly
- You can also use the feature name instead of the index
- Constraints can also be combined, for example, if [1,2] and [2,3,4], the below image would still be legal.



`{1, 2, 3, 4}` *represents the sets of legitimate split features.*

For one last example, we use `[[0, 1], [1, 3, 4]]` and choose feature `0` as split for the root node. At the second layer of the built tree, `1` is the only legitimate split candidate except for `0` itself, since they belong to the same constraint set. Following the grow path of our example tree below, the node at the second layer splits at feature `1`. But due to the fact that `1` also belongs to second constraint set `[1, 3, 4]`, at the third layer, we are allowed to include all features as split candidates and still comply with the interaction constraints of its ascendants.



`{0, 1, 3, 4}` *represents the sets of legitimate split features.*