Deadline: Th. December 19th , 13:15 Drop your printed or legible handwritten submissions into the boxes at Samelsonplatz. Alternatively upload a `.pdf` file via LearnWeb. (e.g. exported Jupyter notebook)

Convention: Always use splitting rules of the form "$x_i <$ value" for numerical and "$x_i =$ cat." for categorical features. Draw the "true" node to the left and the "false" node to the right!

1. Decision Trees (12 points)

In the lecture you have seen the following image of a partition

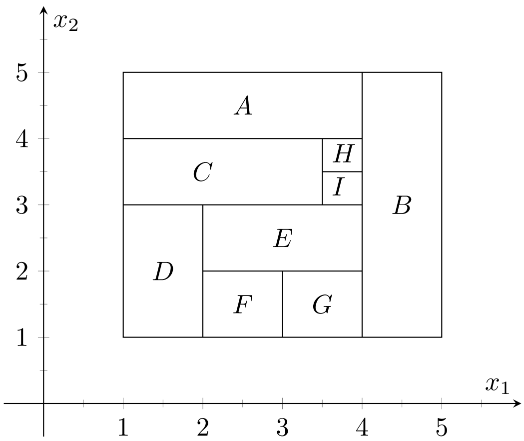

Figure 1: Partition

A. [4p] Construct a decision tree (without explicitly training) which realizes this partition.

B. [4p] We want to predict gender of an elephant given its weight and species (Asian elephant `Elephas maximus` or African elephant `Loxodonta africana`).

Weight	Species	Gender
5500kg	African	male
3500kg	African	female
3400kg	Asian	male
2700kg	Asian	female

Table 1

By hand, train a Decision Tree using the **Information Gain** splitting criterion on the data-set provided by Table 1. Draw the learned tree.

C. [2p] Draw a **minimal depth** Decision tree that solves the classification problem from 1B.

D. [2p] Explain why the learning algorithm does not find the low depth solution. How could one modify the learning procedure or the model such that it does learn a decision tree of depth ≤ 2 for this task?

2. Decision Tree – Programming (12 points)

A. Implement a Decision Tree for Classifier for problems with numerical data in the form of a `scikit-learn` estimator. You will have to implement 2 classes: the model class itself and a tree class. A rough outline is given below (you don't have to stick to it, it is merely a suggestion).

1. A model class with 3 methods: `fit(X, Y)` to fit the model to the data, `predict(X)` to compute the prediction and `score(X, Y)` which computes the accuracy of the prediction. (you are not required to implement any extra options so you can skip `__init__`)

```
import numpy as np

class decision_tree:
    def fit(self, X, Y):
        self.tree = Tree().split(X, Y)
        return self

    def predict(self, X):
        return self.tree(X)

    def score(self, X, Y):
        Yhat = self.predict(X)
        return accuracy of the prediction
```

2. A Tree class that has a method for to split, using the Gini-index as the splitting criterion. As a stopping criterion, simply keep splitting until either there is only 1 sample left or all samples belong to the same class.

```
import numpy as np

class Tree:
    def __call__(self, X):
        if self.is_leaf:
            # return majority class label (from training data)
        # else: obtain results from child nodes (recursion!)
        return prediction

    def split(self, X, Y):
        if self.stop_criterion(X, Y):
          # make the node a leaf
        # else: determine the best split and recurse
        self.rule = best splitting rule
        split = self.rule(X)
        self.left = Tree().split(X[split],  Y[split])
        self.right = Tree().split(X[~split], Y[~split])
        return self

    @staticmethod
    def split_criterion(split, X, Y):
        # split should be a boolean array indicating wether the data satisfies
    the selected rule or not
        return gini index of the split

    @staticmethod
    def stop_criterion(X, Y):
        # implement the stopping criterion. keep splitting until either all data
     belongs to the same class or there is only 1 sample left
        return True/False

    @staticmethod
    def _make_rule(idx, val):
        # return the splitting rule (univariate splits for numerical data)
        return lambda X: X[:, idx] < val
```

B. Compare your own implementation against `sklearn.tree.DecisionTreeClassifier` by evaluating them on both the Iris and Wisconsin Breast Cancer datasets. You can load these datasets via `sklearn.datasets.load_breast_cancer` and `sklearn.datasets.load_iris`.

Use `sklearn.model_selection.train_test_split` with the settings `test_size=0.3` and `random_state=2019` to create the training/test splits.