Deadline: Sunday January 3rd, 23:59 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)

Given is the following training data, the target variable *PlayTennis* with possible values *yes* and *no* needs to be predicted for different Saturdays depending on the attributes of the respective mornings.

Day	Outlook	Temp.	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

A. [5p] Create two binary decision trees using the method introduced in the lecture („greedy strategy"). You can stop after the first two splits. Use the a) Information Gain and b) Gini Index as the split quality criterion, respectively.

B. [3p] The decision tree in Figure 1 was learned without regularization. How would the tree look like if one of the following regularization methods was applied.

- Minimum number of points per cell is set to 4.
- Maximum number of cells is set to 2.
- Maximum depth is set to 3.

Draw all three resulting trees.

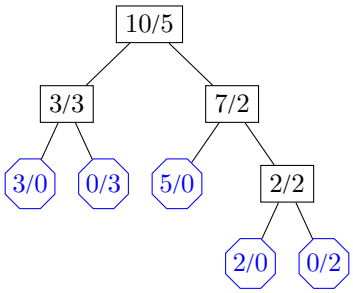

Figure 1: Decision tree for 1B

C. [4p] We want to predict gender of an elephant given its weight and species (Asian elephant `Elephas maximus` or African elephant `Loxodonta africana`). Draw a **minimal depth** Decision tree that solves this classification problem.

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

Table 1

2. Decision Tree – Programming (8 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 decisiontree:
    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.isleaf:
            # return majority class label (from training data)
        # else: obtain results from child nodes (recursion!)
        return prediction

    def split(self, X, Y):
        if self.stopcriterion(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 splitcriterion(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 stopcriterion(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 makerule(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`.

`loadbreastcancer` and `sklearn.datasets.loadiris`. Moreover, use `sklearn.modelselection.traintestsplit` to create train/test splits with the settings `testsize=0.3` and `randomstate=2019`.