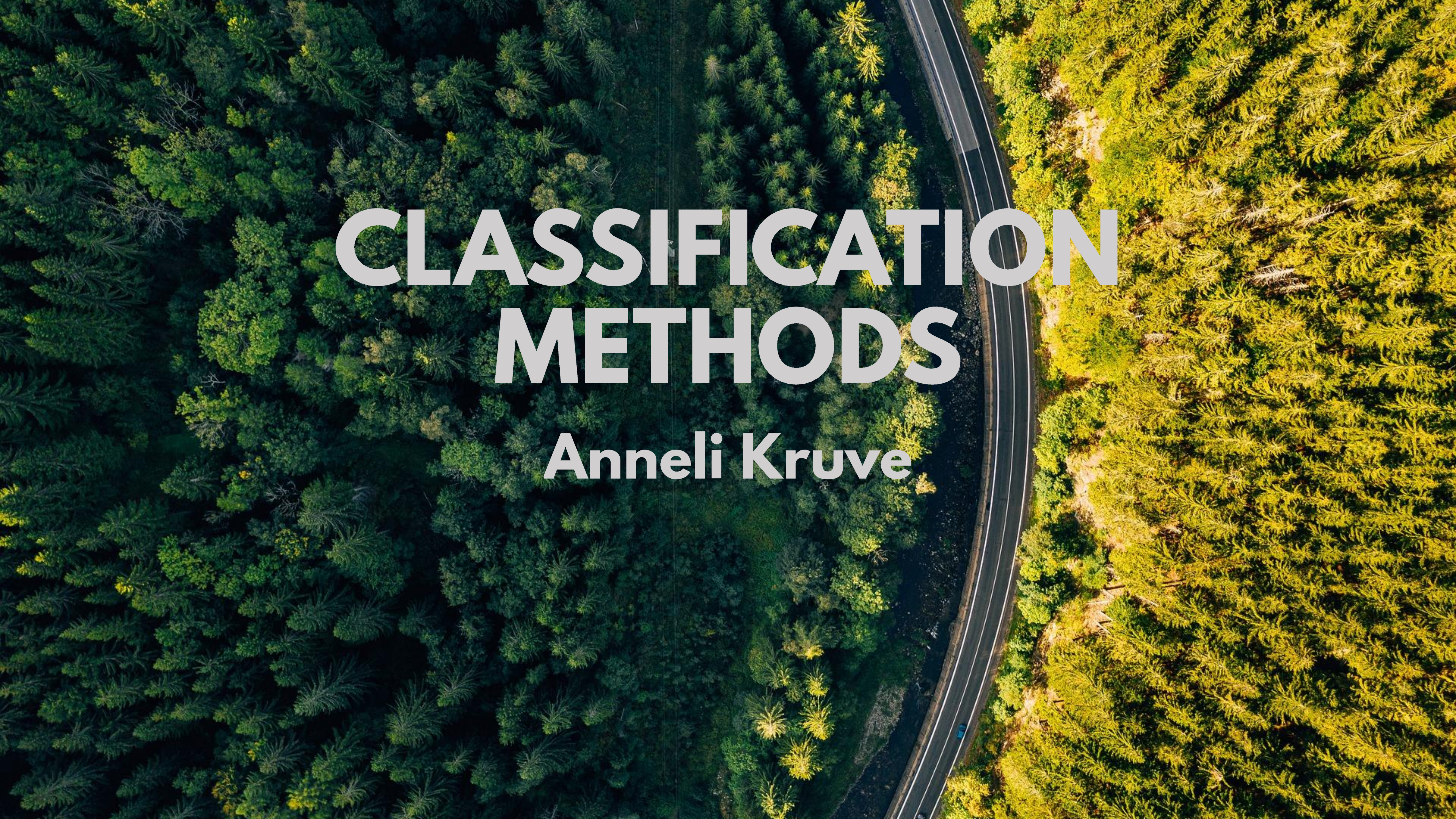
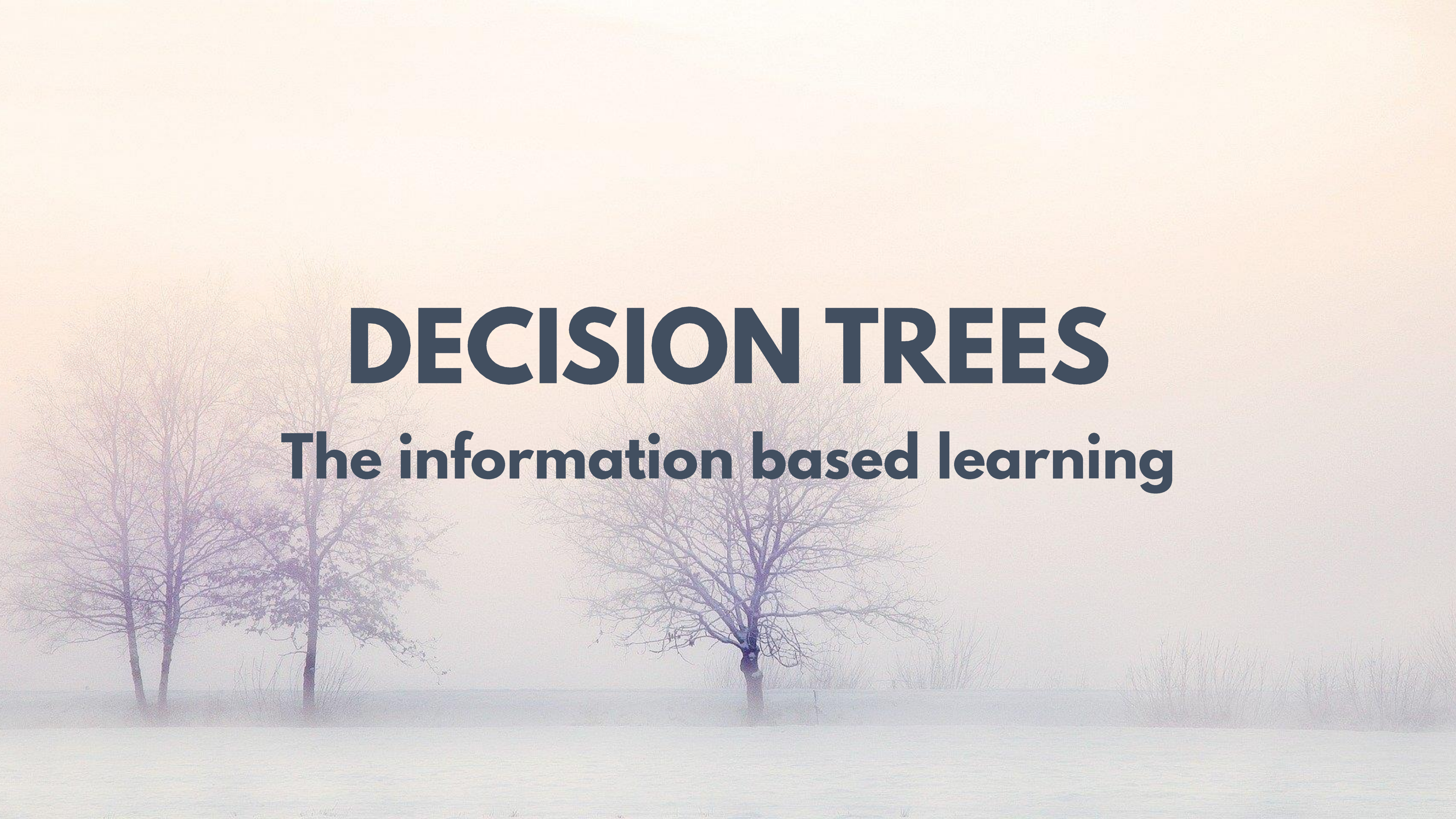


CLASSIFICATION METHODS

Anneli Kruve

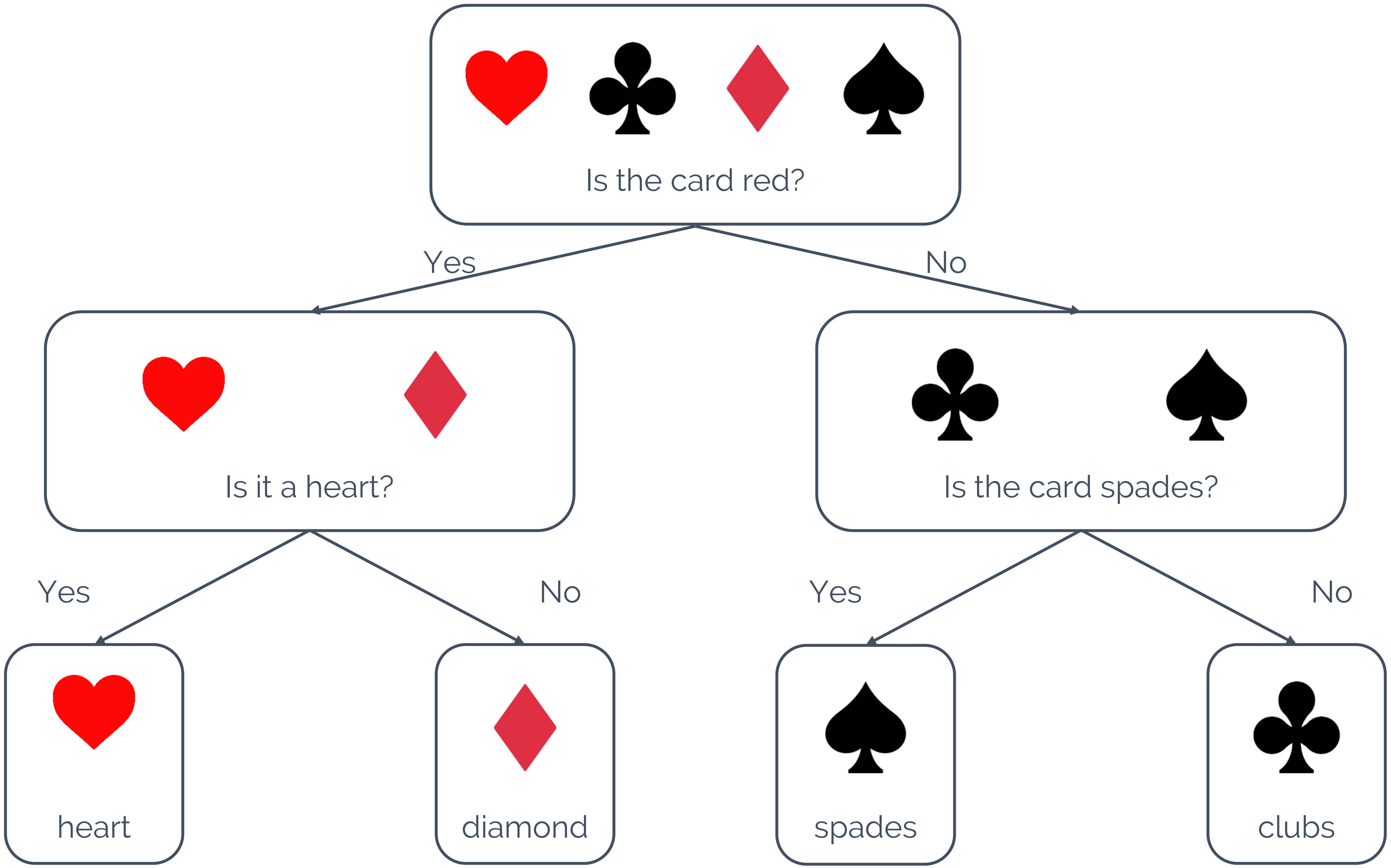


DECISION TREES

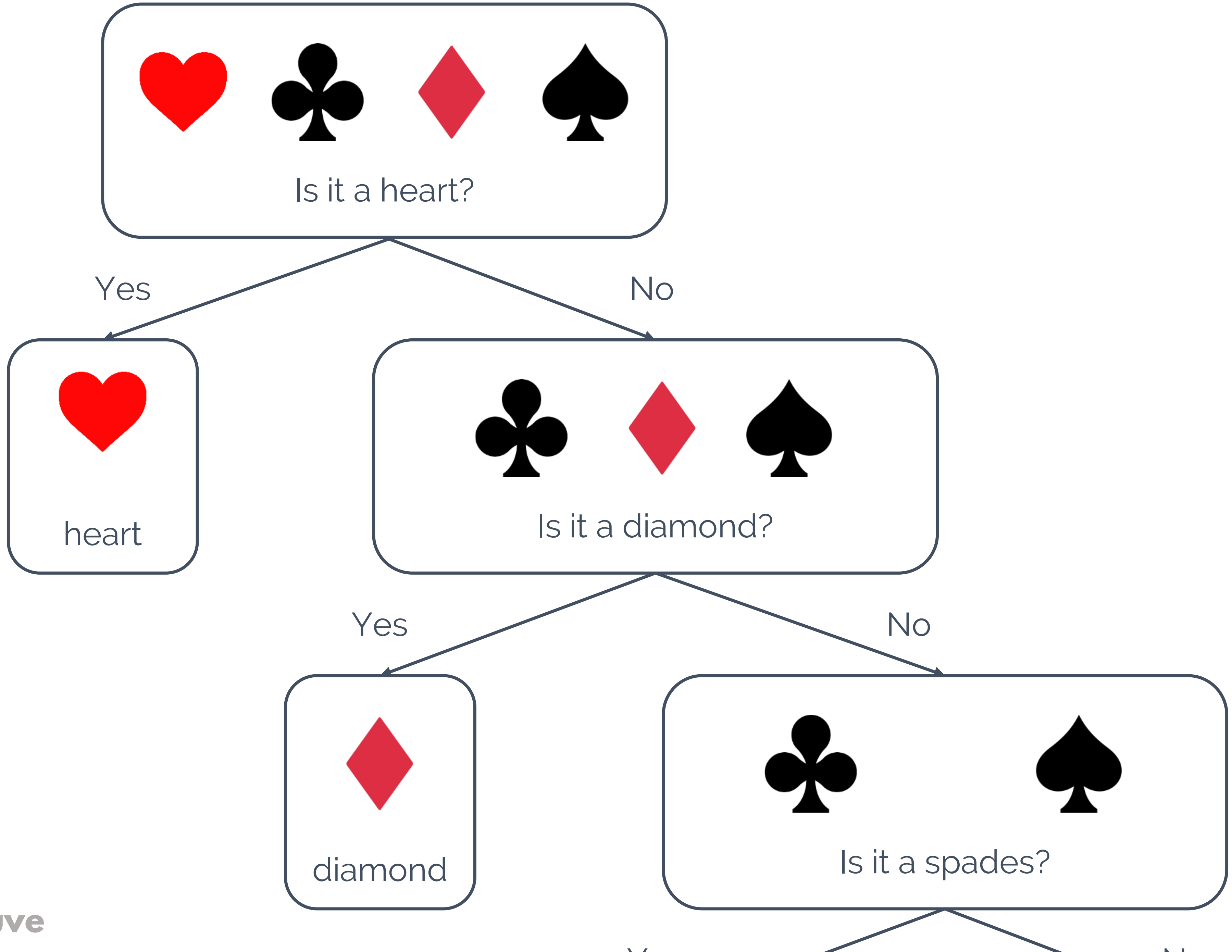
The information based learning

LETS PLAY A GUESS THE CARD GAME

Anneli Kruve



Anneli Kruve



Anneli Kruve

BUT HOW TO ASK THE QUESTIONS?

DECISION TREE

AIMS AT

Finding the most suitable questions to ask to classify the objects fastest.
Also called: shallow decision trees

For this we need to find out as much information as possible with each question.

SHANNON'S ENTROPY MODEL

Entropy is defined as

$$H(t) = -\sum_{i=1}^{l} \left(P(t = i) \times \log_2\left(P(t = i)\right)\right)$$

where
P(*t* = *i*) is the probability that randomly selecting an element *t* is the type *I*
l is the number of different types (classes) of objects in the set

The entropy is directly linked to the heterogeneity of the set

INFORMATION GAIN

STEP 1

Compute the entropy of the original dataset with respect to target feature

$$H(t, D) = -\sum_{i=1}^{l} \left(P(t = i) \times \log_2\left(P(t = i)\right)\right)$$

levels(t) is the set of levels in the domain of the target feature *t*

INFORMATION GAIN

STEP 2

1. Create the sets that result by partitioning the instances in the dataset based on descriptive feature *d*.
2. Calculate the entropy of each of the sets.
3. Sum all of the entropy values.
4. Repeat steps 2 – 3 with the next feature.

$$rem(t, D) = \sum_{l \ in \ levels(d)} \frac{D_{d=l}}{|D|} \times H(t, D_{d=l})$$

INFORMATION GAIN

STEP 3

Subtract the remaining entropy value from the original entropy value for each feature *d*.

$$InformationGain = H(t, D) - rem(d, D)$$

The feature *d* allowing largest information gain is the one that should be used for splitting the dataset.

WORKFLOW OF DECISION TREES

CALLED ID3 algorithm

```
Require: set of descriptive features d
Require: set of training instances D
if all the instances in D have the same target level C then
        return a decision tree consisting of leaf node with label C
else if d is empty then
        return a decision tree consisting of a leaf node with the label of
        the majority target level in D
else if D is empty then
        return a decision tree consisting of a leaf node with the label of
        the majority target level of the dataset of the immediate parent
        node
else ....
```

WORKFLOW OF DECISION TREES

CALLED ID3 algorithm

```
...
else
    d [best] <- arg max InformationGain(d,D)
    make a new node, $Node_{d[best]}$ and label it with d [best]
    partition D using d[best]
    remove d[best] from d
    fore each partition $D_i$ of D do
        grow a branch from $Node_{d[best]}$ to the decision tree created
        by rerunning ID3 with $D = D_i$
```

ADVANTAGES

Minimal data preprocessing is needed
Tree generation includes feature selection
Usually performs very well

DISADVANATAGES

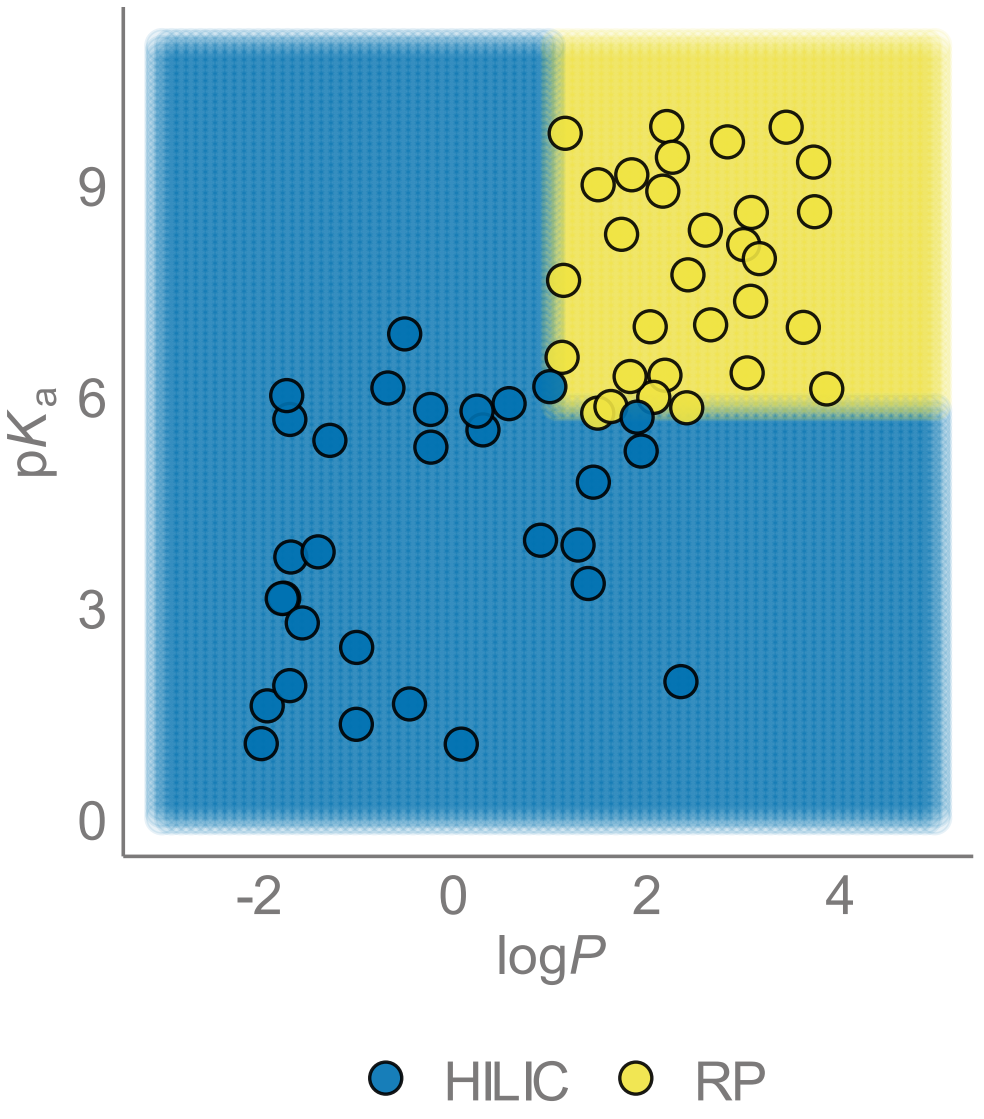
Low explainability

  not good for understanding underlying processes

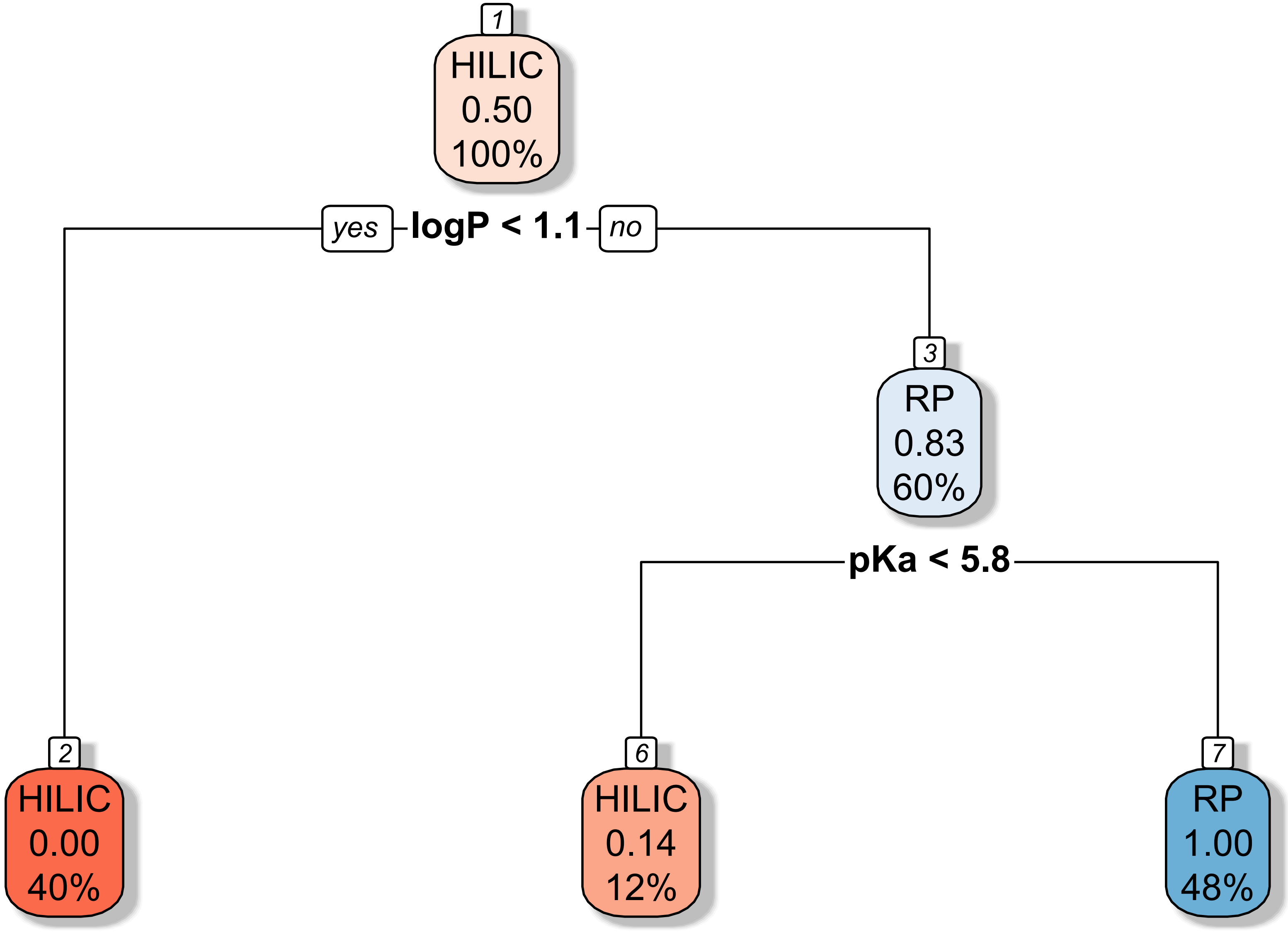
Prone to overfitting

  rigorous training-testing scheme needed

VISUALIZING RESULTS



DECISIONS



Anneli Kruve

RANDOM FOREST

BACKGROUND

Decision trees have a problem of overfitting

    low bias but high variance

A method to overcome this is to train several decision trees and average

BOOTSTRAP AGGREGATING, THE BAGGING

Given a training set $X = x_1, \ldots, x_n$ with classification $Y = y_1, \ldots, y_n$, bagging repeatedly, B times, selects a random sample with replacement of the training set and fit trees to these samples:

For $b = 1, \ldots, B$:

1. Sample, with replacement, *n* training examples from X, Y; call these X_b, Y_b.
2. Train a classification tree on X_b, Y_b.

The data points not used for training of a particular tree can be used to evaluate the performance

FEATURE BAGGING

Often correlation of the trees in an ordinary bootstrap sample:
....if one or a few features are very strong predictors for the response variable (target output), these features will be selected in many of the B trees, causing them to become correlated

Feature bagging helps to overcome this
...at each candidate split in the learning process, a random subset of the features is taken for testing

WORKFLOW OF RANDOM FOREST

```
given training data set
select number of trees to build (ntrees)
for i = 1 to ntrees do
    Generate a bootstrap sample of the original data
    Grow a regression tree to the bootstrapped data
    for each split do
        select m variables at random from all p variables
        pick the best variable/split-point among the m
        split the node into two child nodes
    end
    use typical tree model stopping criteria
end
```

OPTIMIZATION

`ntree`: number of trees. We want enough trees to stabilize the error but using too many trees is unnecessarily inefficient, especially when using large data sets.

`mtry`: the number of variables to randomly sample as candidates **at each split**.

mtry = p the model equates to bagging.

mtry = 1 the split variable is completely random, so all variables get a chance but can lead to overly biased results. A common suggestion is to start with 5 values evenly spaced across the range from 2 to p.

OPTIMIZATION

`sampsize`: the number of samples to train on. The default value is 63.25% of the training set since this is the expected value of unique observations in the bootstrap sample.
Lower sample sizes can reduce the training time but may introduce more bias than necessary.
Increasing the sample size can increase performance but at the risk of overfitting because it introduces more variance.
Typically, when tuning this parameter we stay near the 60-80% range.

OPTIMIZATION

`nodesize`: minimum number of samples within the terminal nodes.
Controls the complexity of the trees. Smaller node size allows for deeper, more complex trees and smaller node results in shallower trees.
Bias-variance tradeoff:
…deeper trees introduce more variance (risk of overfitting)
…shallower trees introduce more bias (risk of not fully capturing unique patters and relationships in the data).
`maxnodes`: maximum number of terminal nodes.
…increase in nodes results in deeper and more complex trees
…less nodes result in shallower trees

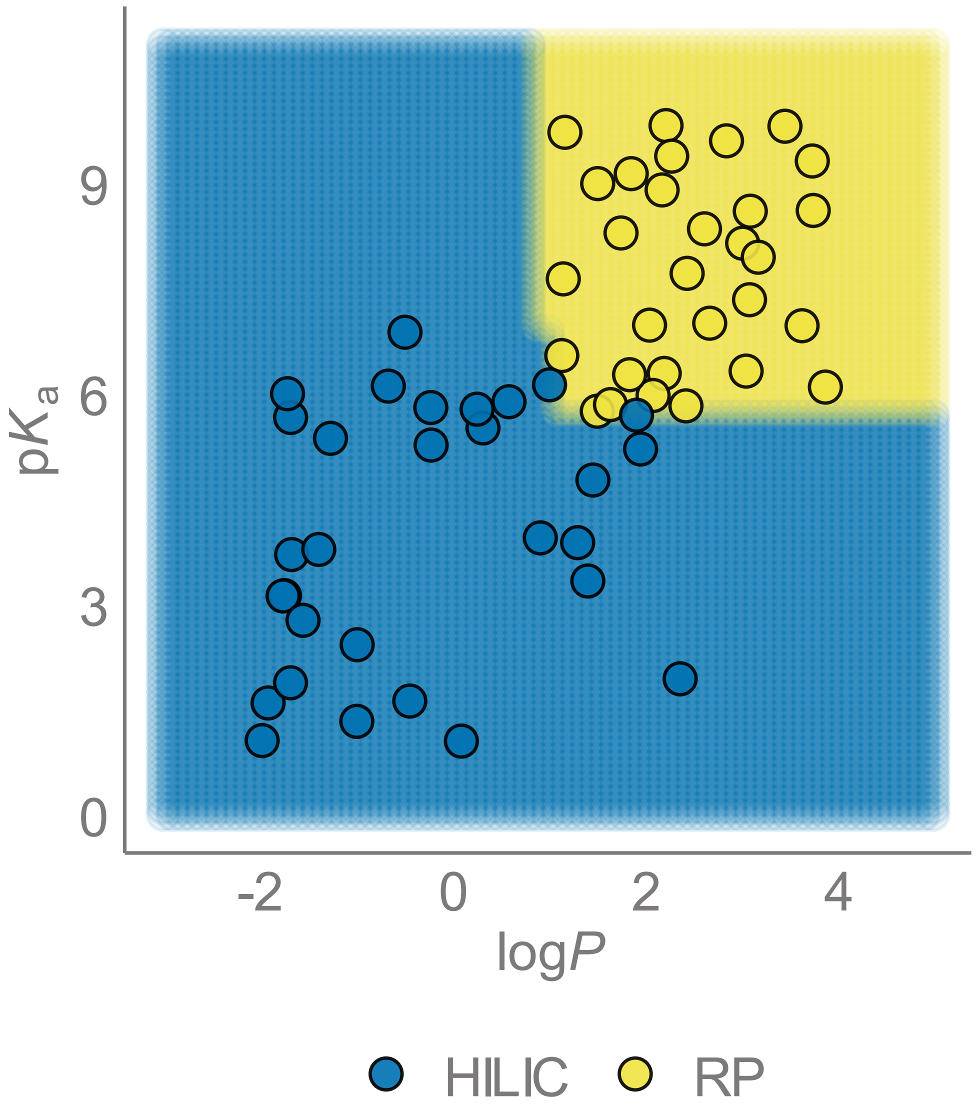
ADVANTAGES

Typically have very good performance
Remarkably good “out-of-the box” solution - very little tuning required
Built-in validation set - don’t need to sacrifice data for extra testing
….validation is still needed!
No pre-processing required
Robust to outliers

DISADVANTAGES

Can become slow on large data sets
Although accurate, often cannot compete with advanced boosting algorithms
Less interpretable

VISUALIZING RESULTS



Anneli Kruve