# Decision Tree

*a non-probabilistic discriminative classifier*

# Contents

# Non-probabilistic Discriminative Classifiers

- Goal: Definition of a function $f(\mathbf{x})$ that predicts the class label $C$ from the data $\mathbf{x}$, i.e. $C = f(\mathbf{x})$
- Probabilities are not considered directly in this context

  → No assumptions about the distribution of the data!
- Focus on decision boundaries → Good results with a relatively low amount of training data
- Posterior probablities can usually be derived in post-processing

  →Required for further processing in a probabilistic context

# Non-probabilistic Discriminative Classifiers: Overview

- Different principles:
  - Decision Trees: Hierarchical classification of feature space
  - Random Forests: Combination of decision trees
  - Boosting: Combination of weak classifiers
  - Support Vector Machines: Find decision boundary having a maximum distance from the training samples
  - Neural Networks: Motivated by a model of information flow in neuron (cells of the nervous system)
  - etc.

# Decision Trees

- Many problems in everyday life are analysed by going through and answering a series of questions
- Example: Assume we have set of red and blue marbles, and we want to build a machine that sorts those marbles according to their colors

→ Question 1: “Is the color of marble red“?

If yes, marble goes to red class

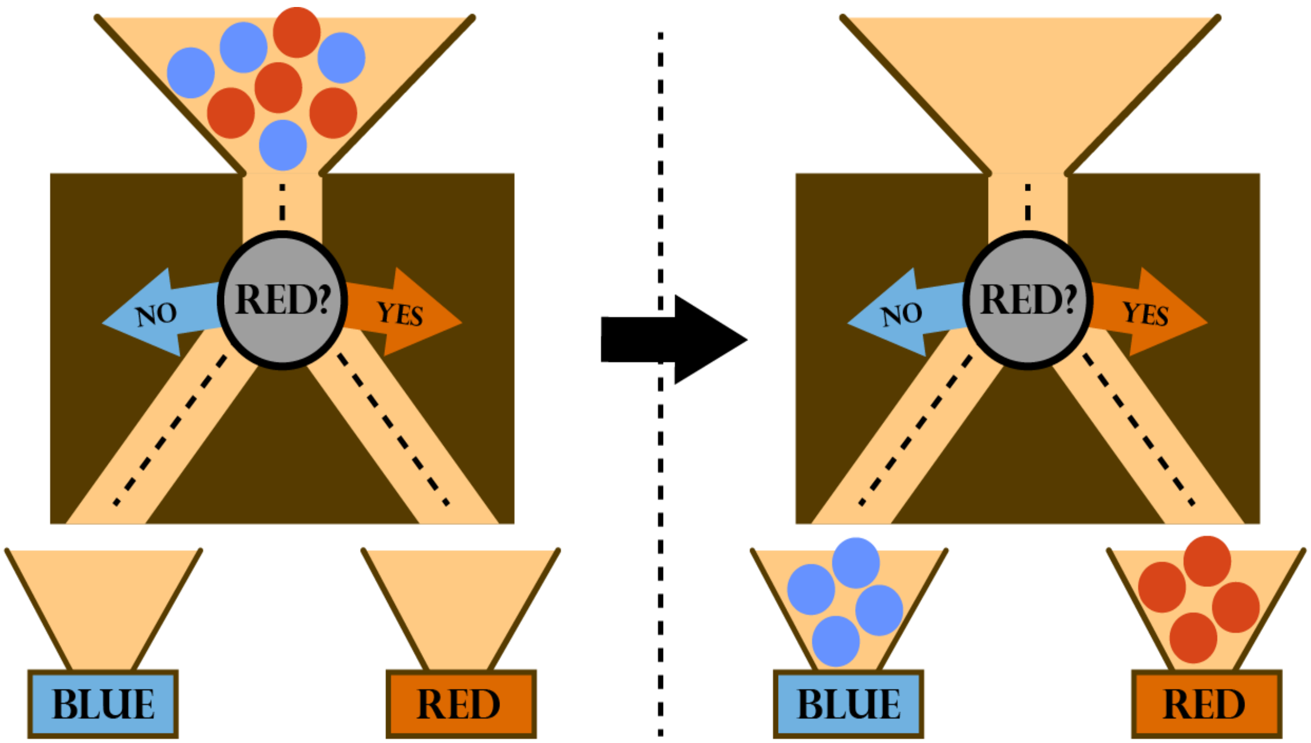
If not:

→ Question 2: “Is the color of marble blue?“

If yes, marble goes to blue class

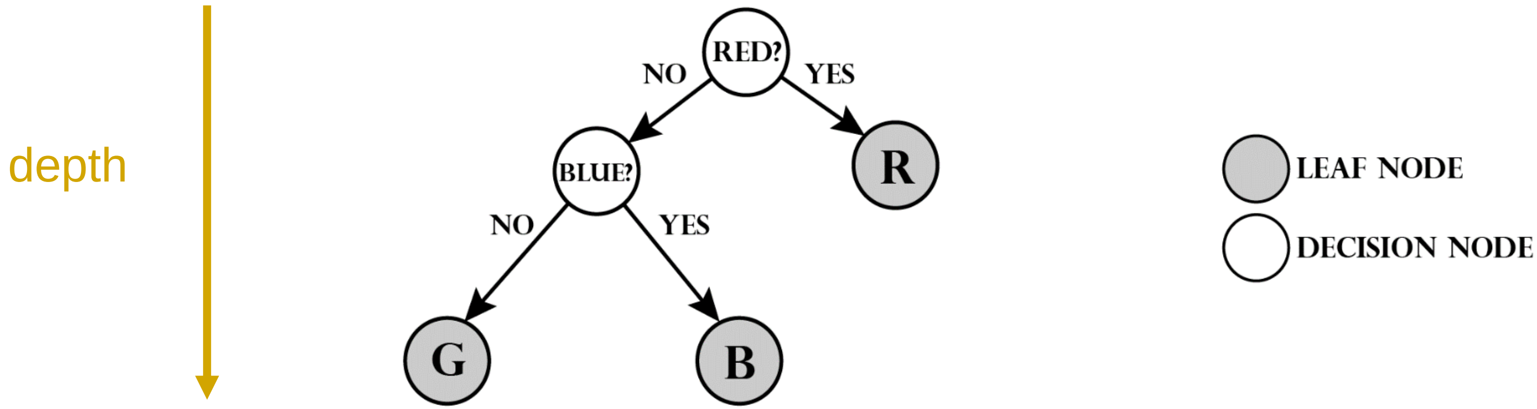
# Decision Trees

- The machine has one marble input and two outputs (one for each class / colour)



# Decision Trees

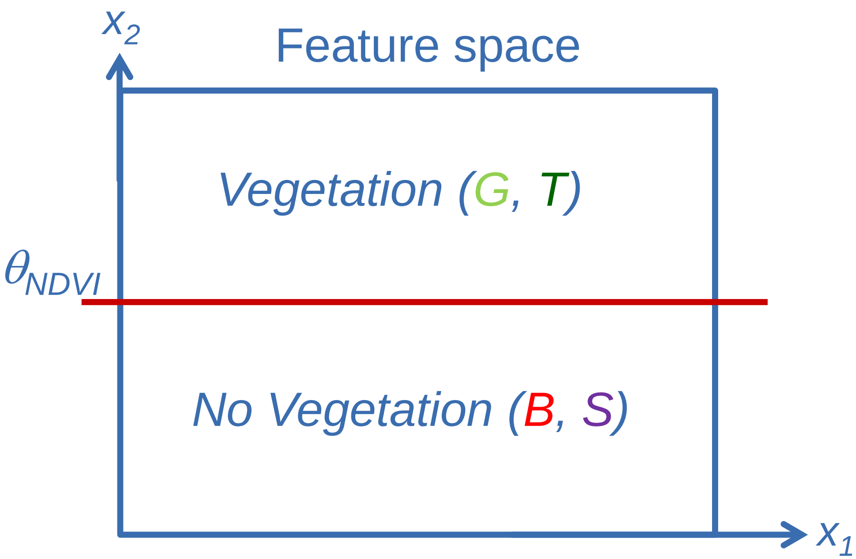
- Continue with same example, by adding a third class: green marbles
- This sequence of queries can be represented by a binary decision tree
- Decision Node: Queries / Decisions
- Leaf Node: Either a result or a probability
- Binary Tree: Every node that is no leaf has two child nodes

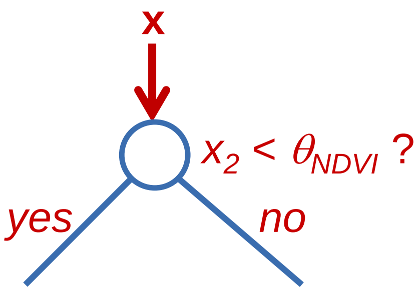


# Decision Trees: Example from Remote Sensing

- Each decision splits the feature space up into sub-regions
- Feature vector $\mathbf{x} = (x_1, x_2)^T$ is presented to the root node

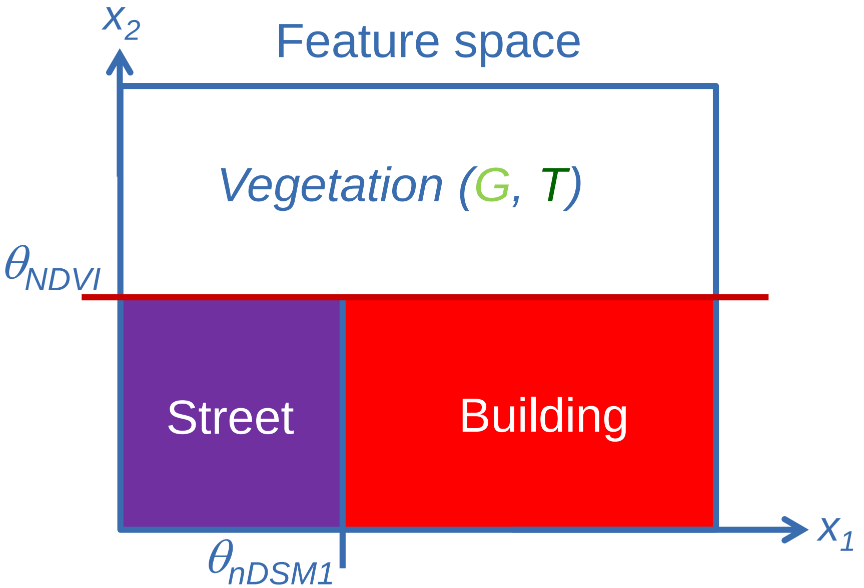
  - Decision 1: is $x_2$ (NDVI) smaller than a threshold value $\theta_{NDVI}$?



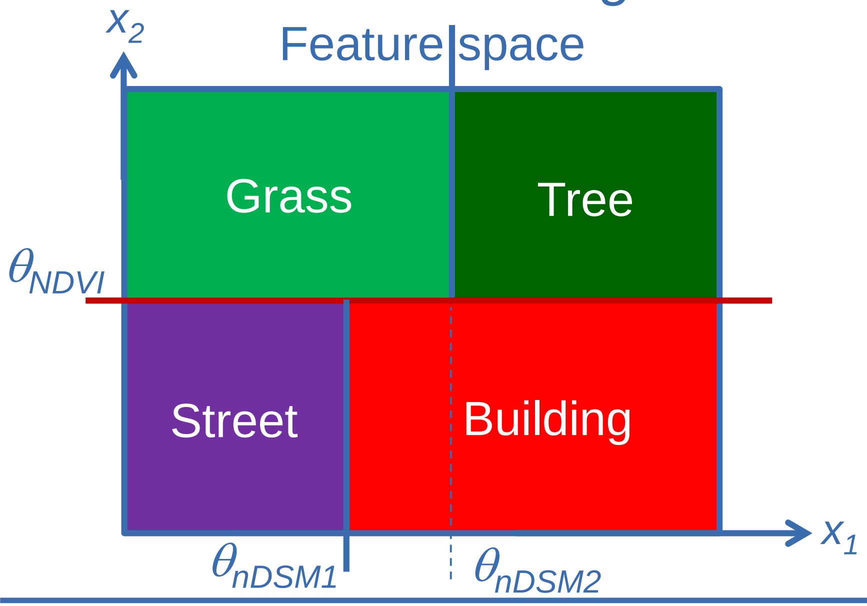
# Decision Trees: Example from Remote Sensing

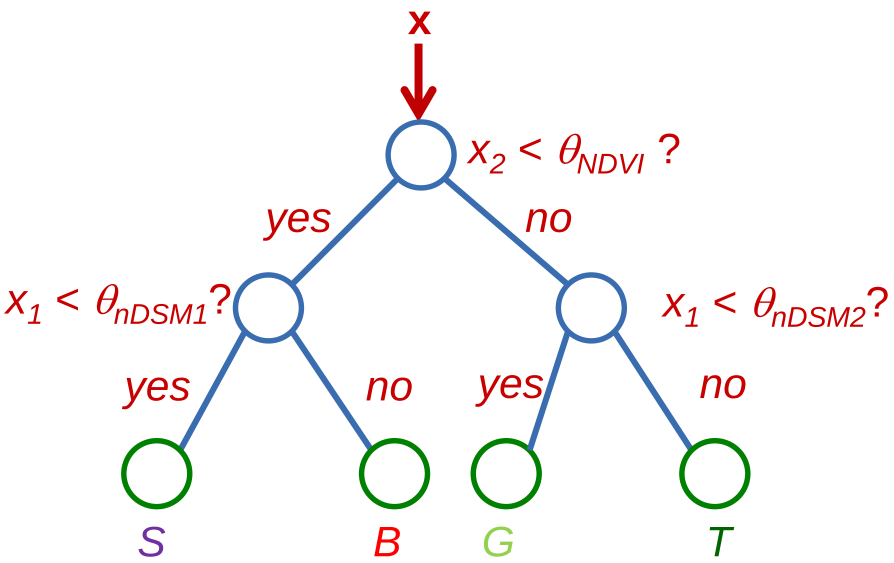
- Each decision splits the feature space up into sub-regions
- Feature vector **x** = $(x_1, x_2)^T$ is presented to the root node
  - Decision 1: is $x_2$ (NDVI) smaller than a threshold value $\theta_{NDVI}$?
  - Decision 2 for *no vegetation*: is $x_1$ smaller than $\theta_{nDSM1}$?



# Decision Trees: Example from Remote Sensing

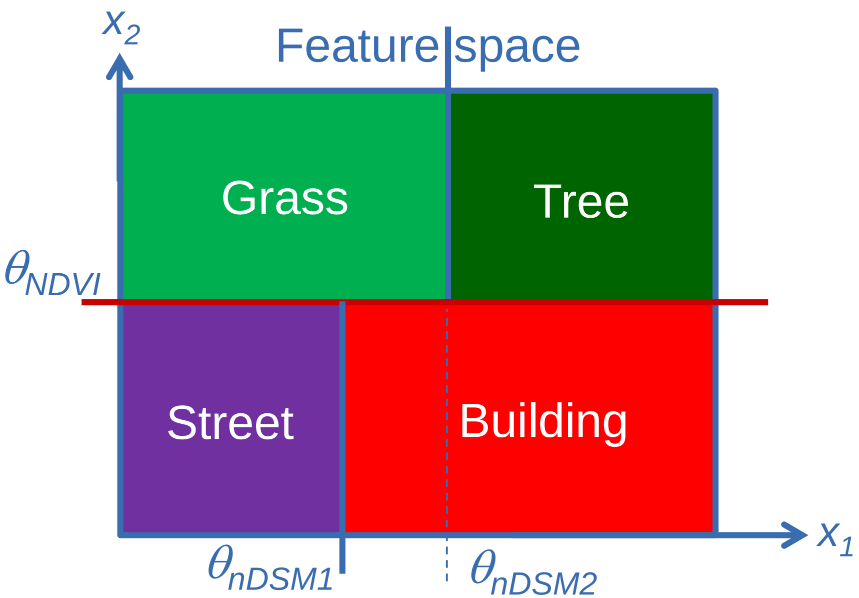
- Each decision splits the feature space up into sub-regions
- Feature vector $\mathbf{x} = (x_1, x_2)^T$ is presented to the root node
  - Decision 1: is $x_2$ (NDVI) smaller than a threshold value $\theta_{NDVI}$?
  - Decision 2 for *no vegetation*: is $x_1$ smaller than $\theta_{nDSM1}$?
  - Decision 2 for *vegetation*: is $x_1$ smaller than $\theta_{nDSM2}$?

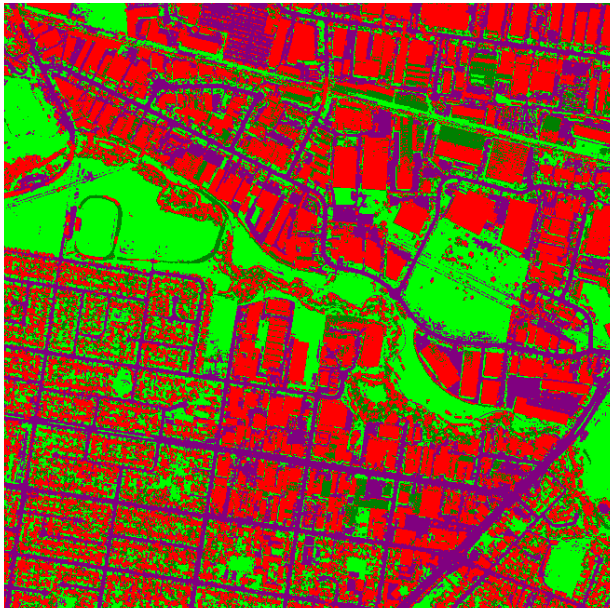


# Decision Trees: Example from Remote Sensing

- The feature space is hierarchically split into disjunct regions
- Different values for the three parameters ($\theta_{NDVI}$, $\theta_{nDSM1}$, $\theta_{nDSM2}$) lead to different results



# Decision Trees: Discussion

- Very simple and "clear" design → very popular
- Can be adapted by the user easily (choice of thresholds)
- This partitioning of the feature space does not adapt very well to the shapes of the clusters in feature space
- Result depends on the choice of the threshold values
- Different possibilities for the construction of the tree

→ Can these trees be learned for the training data?

→Is there a better way to adapt the decision boundaries than interactive trial-and-error?

# CART (Classification and Regression Trees)

- General method for learning of binary trees
- Applicable for classification, regression and clustering
- There are different versions of CART
- What is to be determined during training?
  1) How to split the data in each node?
  2) How to decide whether a node corresponds to a leaf or not?
  3) How to determine which class corresponds to a leaf?

# CART: Splitting of Data

- A test is carried out in each node
- Up to now: Each test is based on the comparison of a feature with a threshold value
- More general type of test: Split the feature space with a linear decision boundary (a hyperplane)
  - → Simultaneous consideration of several features
  - → Allows for decision boundaries in feature space that are not parallel to the coordinate axes
- The type of the tests (threshold vs. hyperplane) must be defined in advance
- Learning the tests only requires a part (e.g. 1/3) of the training data

# CART: Learning of the Tests

- In each node of the tree:
  - Randomly select $n$ features
  - Randomly generate $r$ different separating hyperplanes operating on the selected features
  - Each hyperplane is examined according to how well it can separate the data
    → information gain criterion
  - The best hyperplane is retained for the node
- The number of features for the test has to be specified by the user
  good value for D-dimensional feature vectors: $n = \sqrt{D}$

# CART: Selection of the Separating Hyperplane

- The parameters of the tests (threshold vs. hyperplane) can be learned
- Separating hyperplane: $\mathbf{w}^T \cdot \mathbf{x} + w_0 = 0$

  → $\mathbf{w}$: random numbers numbers in [-1, 1] for the *n* features selected randomly;
  for the other features, the components of $\mathbf{w}$ are set to 0

  → $w_0$: random number between [min ($\mathbf{w}^T \cdot \mathbf{x}$), max ($\mathbf{w}^T \cdot \mathbf{x}$)]
- The hyperplane splits the training data in two parts $M_1$, $M_2$:
  1) $M_1$: $\mathbf{w}^T \cdot \mathbf{x} + w_0 \leq 0$
  2) $M_2$: $\mathbf{w}^T \cdot \mathbf{x} + w_0 > 0$
- $M_1$ and $M_2$ correspond to the branches leaving the node

# CART: Information Gain

- For each of the subsets ($M_1$, $M_2$) generated by the test, a histogram of the class labels can be determined
- The histogram entries for $M_j$ are interpreted as $P_j(C=L^k)$
- Criterion for the quality of the separation: information gain $\Delta E$:

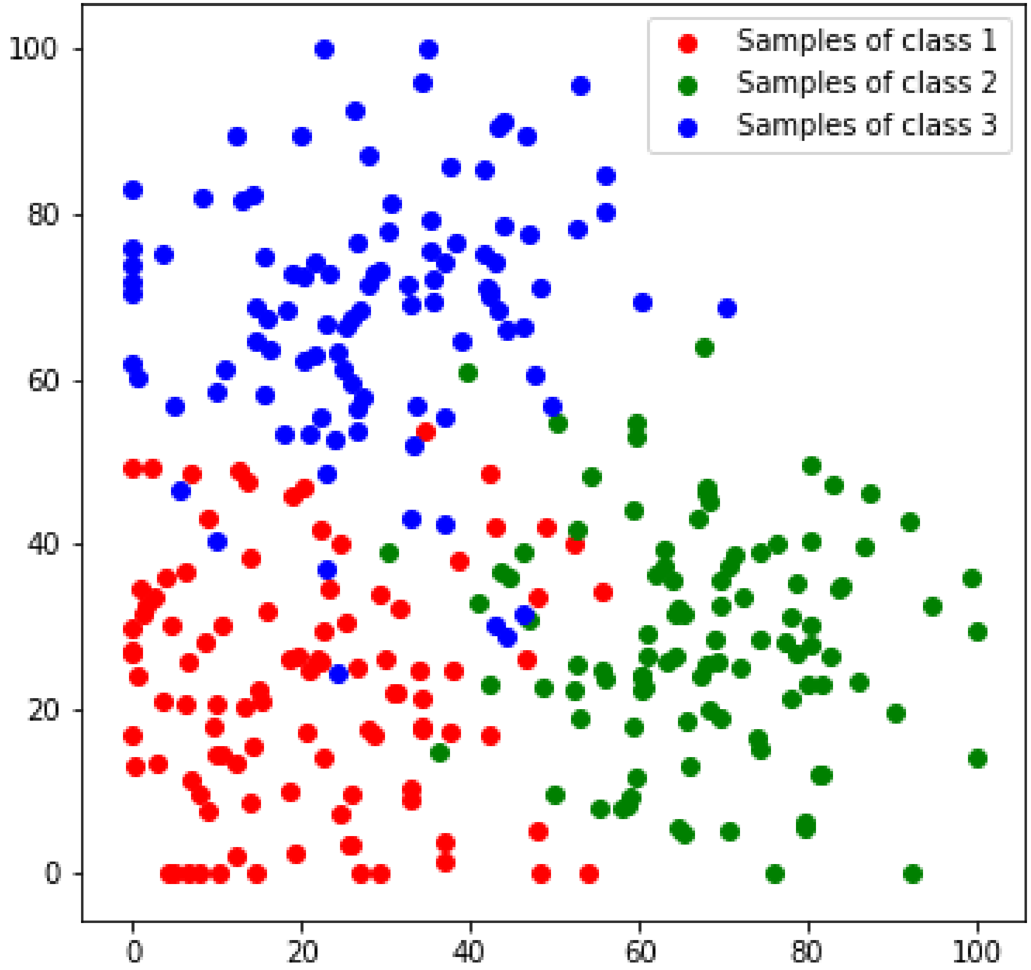
$$\Delta E = \frac{N_1}{N_1 + N_2} \cdot \sum_k P_1\left(L^k\right) \cdot \log_2\left[P_1\left(L^k\right)\right] + \frac{N_2}{N_1 + N_2} \cdot \sum_k P_2\left(L^k\right) \cdot \log_2\left[P_2\left(L^k\right)\right]$$

(only relevant terms are shown)

- $N_1$, $N_2$ are the number of training samples in $M_1$ and $M_2$, respectively
- Each of the sums is the entropy E of the histogram
- The bigger $\Delta E$, the better a hyperplane separates the data

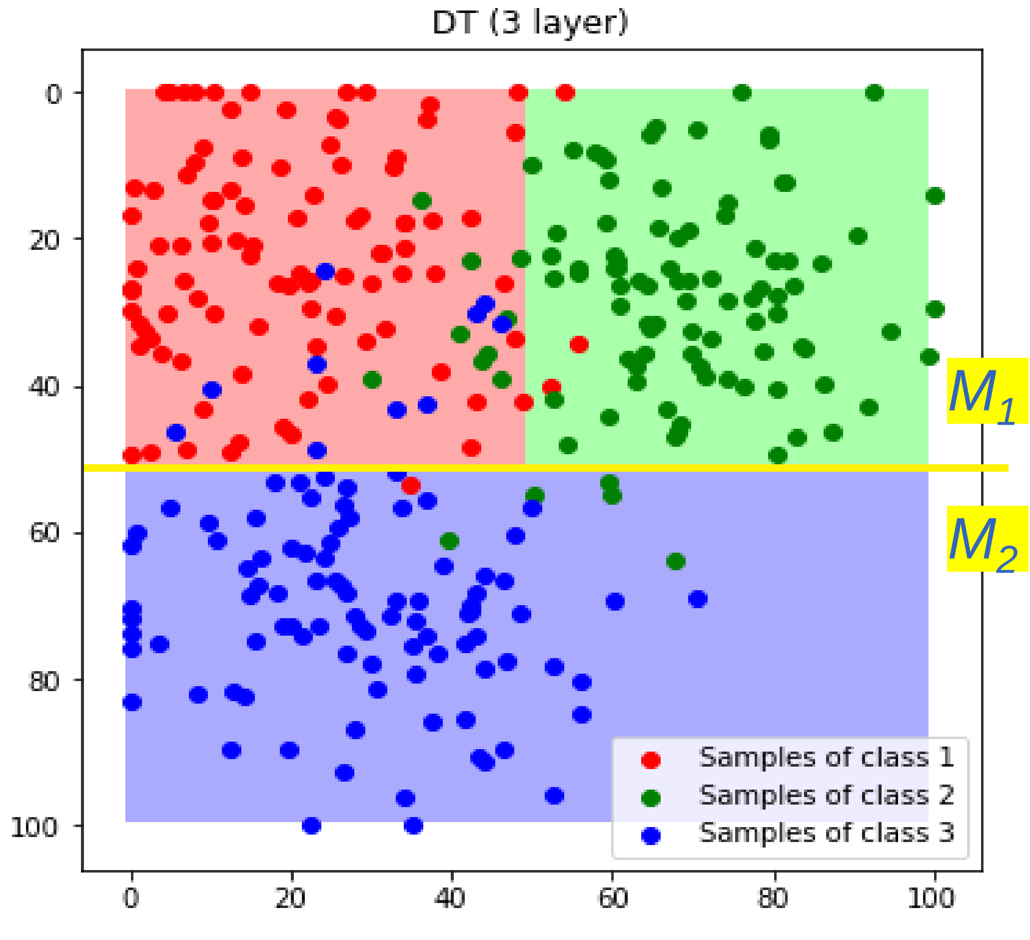
# CART: Example for Learning

- Example with three classes, two features $x_1$, $x_2$



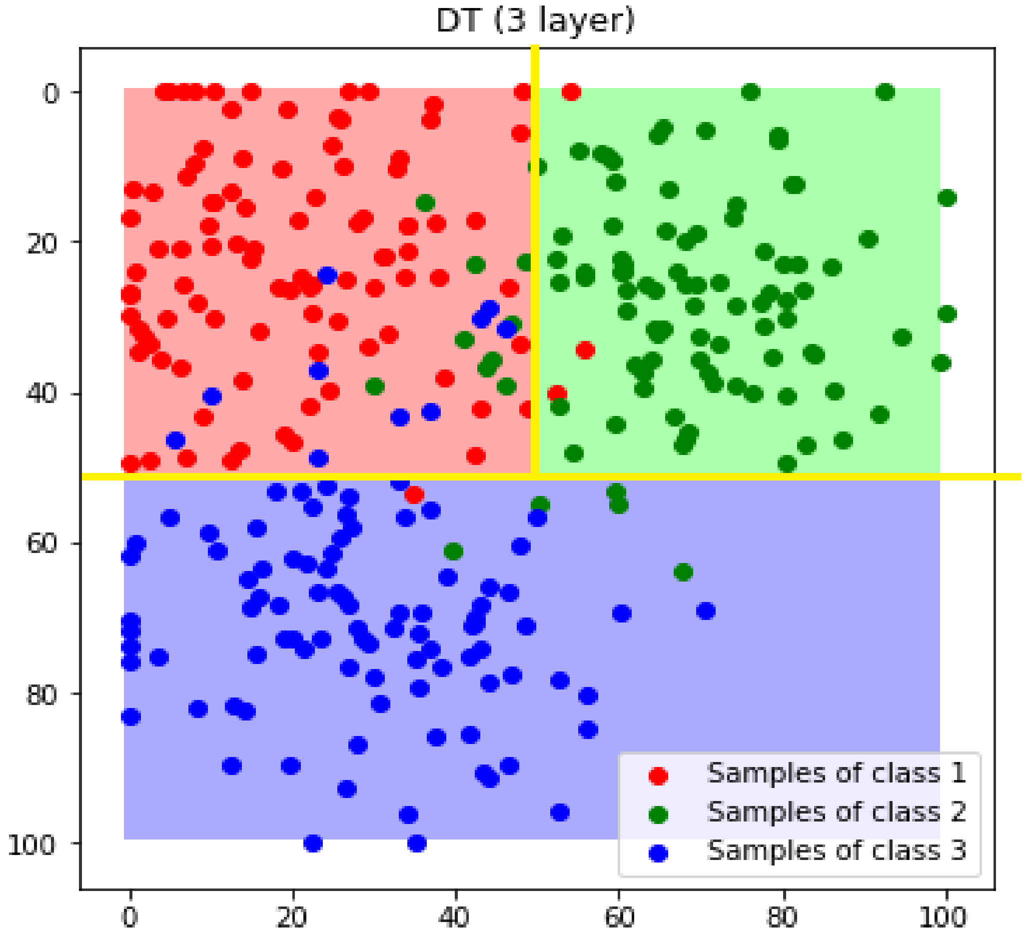
# CART: Example for Learning

- The decision boundaries are generated after a few step(tree with a depth of 3):
  - ➢ Random selection of a separatiing hyperplane
  - ➢ Determination of the histogram
  - ➢ Computation of information gain and selection of the hyperplane with maximum $\Delta E$
  - ➢ Repeat recursively for $M_1$



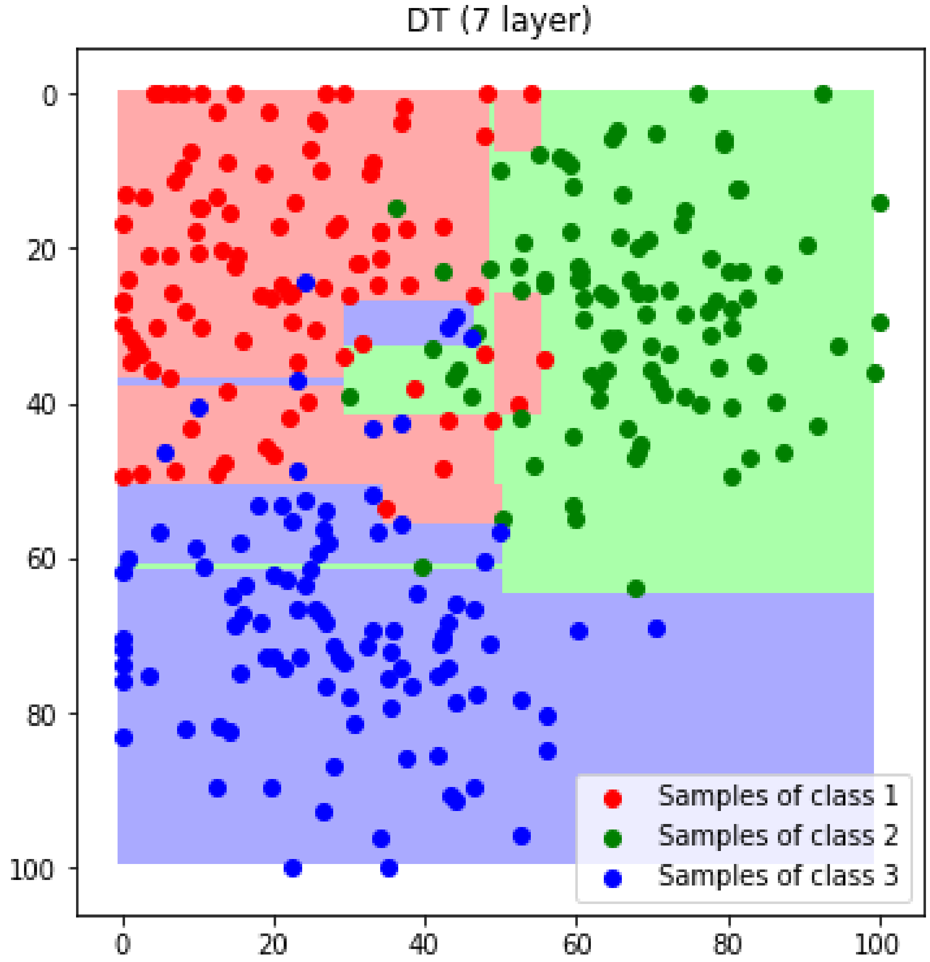
# CART: Example for Learning

- After same process repeated here, we can think about the influence of the depth



# CART: Example for Learning

- A tree with a depth of 7
- Increasing the number of layers, the tree starts to add thin areas that correspond to outliers
- The model is overfitting to the training data
- When to stop the recursion?



# CART: Stopping Criteria for Training

- For a unique assignment of a leaf to class: recursion is finished if only training samples of a single class are available in the leaf
- This may lead to overfitting and very deep trees
  → finish the recursion if
  - very few training samples fall into one node
  - the information gain is very small
  - a specified maximum depth is reached
- If one of the termination criteria is met, a node is declared to a leaf
- As soon as each path through the tree ends in a leaf, the training of the test is finished

# CART: Assignment of Leafs to Classes

- Remember: The learning of the tests only requires a part (e.g. 1/3) of the training data
- The remaining training samples are presented to the tree and passed through the tree until they end up in a leaf
- In every leaf *b* the normalised histogram of class labels $P_b(C=L^k)$ is determined on the basis of the training samples arriving at the leaf
- Interpretation of histogram as posterior : $P(C=L^k \mid \mathbf{x}) = P_b(C=L^k)$
- The leaf is assigned to the class for which $P(C=L^k \mid \mathbf{x}) \rightarrow \max$
- The posterior can be stored in the leaf if a probabilistic output is required

# CART: Pruning

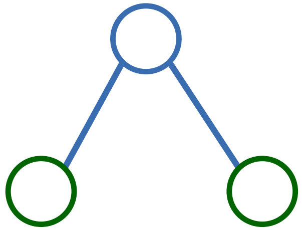
- Problems of the CART-algorithm:
  - Overfitting
  - Generation of trees that are too deep
  - Generation of trees that have too many leaves

    → Pruning: check whether the training error or a different criterion will change significantly if a node that is not a leaf is declared a leaf; if not → branches emanating from that node are deleted
- Variants of decision trees
  - ID3: Multipath splits, termination if all leaves are "pure“
  - C4.5 [Quinlan, 1993]: based on ID3, includes pruning

# CART: Discussion

- How many features or tests should one try?
  - Only one → „Extremely randomized tree“
  - Few → Fast training, may lead to underfitting
  - Many → slower training, may lead to overfitting

- Decision Stump: The simplest conceivable tree consisting of the root and two leafs only
  - Used in combination with other methods



# Discussion

- CART are still quite popular
- Requires good choice of the parameters for learning
  - Type of the tests to be caried out in each node
  - Number of features per test
  - Number *r* of attempts to find the optimal boundary in a node
  - Minimum number of training points per node
  - Maximum depth
- Very fast both in training as well as in classification
- CART have a tendency to overfit
- Small changes in the training data can lead to major changes in the decision boundaries