Machine Learning I

Chapter 07 - Classification and Regression Trees (CART)

Prof. Dr. Sandra Eisenreich

WS 2024/25

Hochschule Landshut

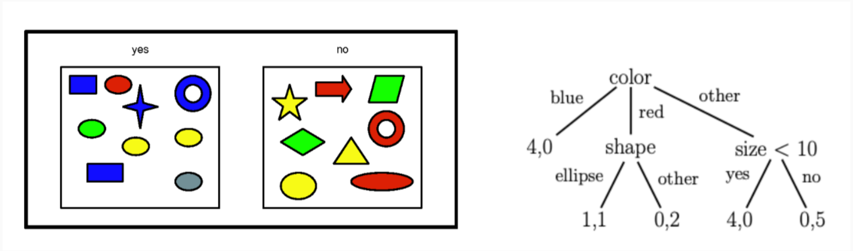
Overview

- tasks: **classification and regression**
- data: labeled, i.e. **supervised** learning; Note: not necessary to scale data!
- model: **non-parametric** in the sense that they have parameters, but not a fixed number, i.e. the resulting prediction function varies from model to model.
- complexity: easy and fast to train and make predictions
- focus: **explainable, very flexible, not robust, i.e. high variance, low bias.** Also used for feature selection. Attention: tends to overfit

When to use:

- CART can deal with all sorts of data without much preprocessing, it can deal with outliers or non-standardized data.
- for automatic feature selection
- simple, easy, interpretable. Only downside: high variance, overfitting. This is why decision trees are often used in the form of Random Forests or Boosting Models (see next chapter).

The idea: Splitting the data into a tree to capture the structure



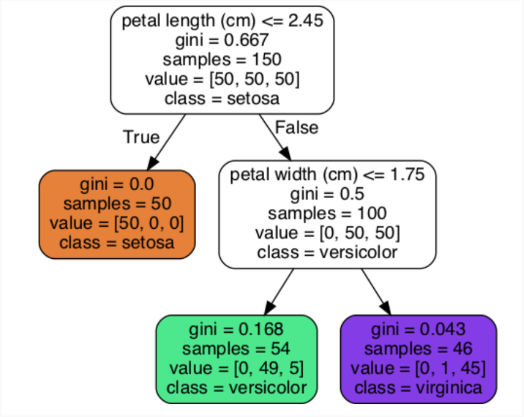
Basic idea is similar to ID3 algorithm: ?

However, in comparison to ID3, CART works for regression too, grows only binary trees and doesn't work with information gain/entropy, but usually with different metrics (Gini impurity/MSE).

Example of inference with decision trees

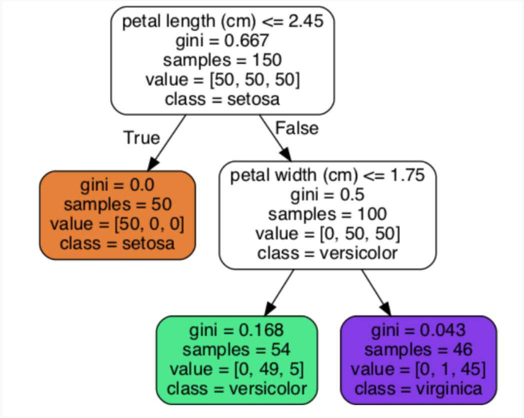
We want to classify an iris flower. Start at the root node (depth 0 at the top), which asks the question: is the petal length smaller than 2.45 cm?

- If true → go to left child node (depth 1, left) which is a leaf node (i.e. doesn't have any child nodes) with predicted class = Iris setosa.
- If not → go to the right child node, which asks another question: is the petal width smaller than 1.75 cm?
    - If so, it is an Iris versicolor,
    - If not it is an Iris virginica.



Growing a tree

- start at the root node with all instances from the training set.
- Optimization step: At each node choose *one feature i* (e.g. petal length) and a threshold τ, which makes the resulting groups of data as *homogeneous/pure* as possible
- Partition the data in two: put instances with $x_i \leq \tau$ in the left node and all instances with $x_i > \tau$ in the right.
- Contintute splitting until you reach a stopping criterion.



Impurity Loss Functions

How do we measure how homogeneous a group of instances D is?

Measures for impurity

- for classification: denote $p_c = \frac{\text{class } c \text{ instances in D}}{\text{number of instances in D}} \in (0,1)$
  - Gini impurity (default):
  $$G(D) = 1 - \sum_{c=1}^{n} p_c^2 \in (0,1)$$
  - Entropy:
  $$H(D) = -\sum_{\substack{c=1 \\ p_c \neq 0}}^{n} p_c \log_2(p_c)\,.$$

  Both are minimized if the node is pure, i.e. only has instances from one class.
- for regression: MSE of how far the labels of the instances in D are away from the mean $\bar{y}$ of all labels in D:
$$\text{MSE}(D) = \frac{1}{|D|} \sum_{k \in D} (y_k - \bar{y})^2.$$

How to pick the best threshold τ

If we have a measure of impurity c as before, we want to split a group of instances into two groups such that the impurity of left group + impurity of right group is minimal (weighted by the number of samples in the groups). I.e. the objective is:

CART (=Classification and Regression Tree) loss function:

$$L(k, t_k) = \frac{m_{\text{left}}}{m} c_{\text{left}} + \frac{m_{\text{right}}}{m} c_{\text{right}},$$

$$\text{where} \begin{cases} c_{\text{left/right}} & = \text{impurity measure of the left/right subset} \\ m_{\text{left/right}} & = \text{number of instances in the left/right subset} \end{cases}$$

We want to choose the threshold τ such that it minimizes the impurity cost of the partition, i.e. $\tau = \underset{\tau}{\operatorname{argmin}}\ L(i, \tau)$.

Greedily pick the feature where a split decreases our cost function the most. $\rightarrow$ good solution, but not optimal.

CART

A **CART (Classification and Regression Tree)** is the model defined by the tree.

- Prediction function is given by the tree in itself: Follow the tree to a final leaf. The output of a leaf is given as follows:
  - **classification**: majority vote, i.e. the class of the most instances in the leaf.
  - **regression**: the mean $\bar{y}$ of the instance labels at the leaf.
- Objective function at each splitting to grow the tree: at the current node we pick

  $$i, \tau = \operatorname*{argmin}_{i} \left(\operatorname*{argmin}_{\tau} L(i, \tau) \right),$$

  where $L(i, \tau)$ is the CART loss function

Note: for each node, i and τ are parameters, but the number of branches of the tree is variable.
$\Rightarrow$ non-parametric model.

Example

The following dataset D shows whether Aileen finds a dish in a Restaurant appealing based on the dish's teperature, taste and size.

Appealing	Temperature	Taste	Size
No	Hot	Sweet	Small
No	Cold	Sweet	Large
No	Cold	Sour	Small
No	Hot	Sweet	Large
Yes	Cold	Sour	Large
Yes	Hot	Sour	Small
Yes	Hot	Sour	Large
Yes	Cold	Sweet	Small

We build a CART to predict whether a dish is appealing to Aileen.

- What is the initial Gini impurity of the dataset D?
- Which feature is used for the split?

Feature Importance - Motivation

A speciality of trees: you can determine how important each feature is by looking at how each feature contributes to reduce impurity. For a feature i:

- look at all nodes k where feature i is used to split a dataset D_k into a left dataset D_k^L and a right dataset D_k^R.
- The impurity before the split is $c(D_k)$. The impurity after the split is $c(D_k^L) + c(D_k^R)$. Compute how much impurity decreases at that node:

$$G_k(c) = c(D_k) - c(D_k^L) - c(D_k^R)$$

.

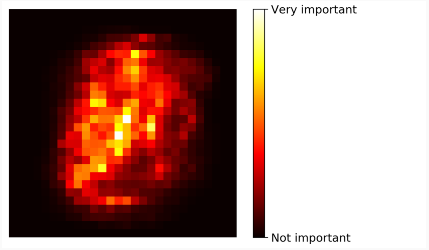
Feature Importance

The importance of feature i in a decision tree T is given by

$$\text{Importance}_i(T) = \sum_{\text{k}} G_k(c),$$

where the sum is over all nodes k that are not leaves and use feature i to partition, c is the impurity measure, and $G_k(c) = c(D_k) - c(D_k^L) - c(D_k^R)$ is the reduction in cost when going from node k to its left and right children nodes.

Here is the feature importance of each pixel in the MNIST dataset:



Complexity

as always: n = number of features and m = number of training instances;

The main driver of complexity is computing the loss

$$L(i,\tau) = \frac{m_{\text{left}}}{m} c_{\text{left}} + \frac{m_{\text{right}}}{m} c_{\text{right}},$$

$$\text{where} \begin{cases} c_{\text{left/right}} & = \text{ impurity measure of the left/right subset} \\ m_{\text{left/right}} & = \text{ number of instances in the left/right (of } \tau\text{) subset} \end{cases}$$

for each feature i and threshold τ at a node.

To compute it... ?

- The complexity of growing CART is $O(d \cdot n \cdot m \log_2 m)$, where d is the depth.
- The inference complexity is ?

Complexity of a CART

The complexity notion before has nothing to do with the following notion of complexity of the model itself:

The complexity of a decision tree is defined as the number of splittings.

Some splittings create more effect (i.e. reduction on loss) than others. In order to increase performance of the CART and as a means of regularization, you can use pruning after training the model.

Pruning

Pruning (or: Post-Pruning) is the process of removing nodes in a trained decision tree. There are two methods:

- Bottom-up pruning:
  - Simple method - reduced error pruning: for each leaf node, evaluate the effect of removing the split above using a hold-out test set. Leaf nodes are removed only if it results in a drop in the overall cost function on the entire test set. You stop removing nodes when no further improvements can be made.
  - Alternative - Minimal Cost-Complexity Pruning: generate a series of trees and corresponding parameters α (one value for each tree) as follows: at step i, the next tree is created by removing a subtree and replacing it with a leaf node. The subtree is chosen to minimize the difference in classification errors divided by the difference in number of leaves.
- Top-down pruning - Pessimistic Error Pruning (PEP) (not covered here)

Decision Trees with Scikit-Learn

Training a Decision Tree:

```
from sklearn.tree import DecisionTreeClassifier
tree_clf = DesicionTreeClassifier(max_depth=2)
tree_clf.fit(X,y)
```

- Inference on new data: `.predict()`.
- A classification tree can also estimate the probability of an instance belonging to class *k* as the fraction of training samples of the class in the leaf: You can output probabilities with `.predict_proba()`
- feature importance, use `.feature_importances_`
- Prune a tree as in this documentation

Regularization Hyperparameters

If you leave it to itself, the CART (=Classification and Regression Tree) algorithm will keep splitting until all leaves are pure, which would be clearly overfitting the data. To avoid overfitting, you need to stop the tree from growing to long.

Scikit Learn's `DecisionTreeClassifier` class has the follwoing hyperparameters to stop it from growing:

- `min_samples_split`= minimum number of samples a node must have before it can be split
- `min_samples_leaf`= minimum number of samples a leaf node must have
- `min_weight_fraction_leaf`= same as above but expressed as a fraction of the total number of weighted instances
- `max_leaf_nodes`= maximum of leaf nodes
- `max_features`= maximum number of features that are evaluated for splitting at each node

Plotting a decision tree with Scikit-Learn

```python
from sklearn.tree import export_graphviz

export_graphviz(
                tree_clf,
                out_file = image_path("iris_tree.dot")
                feature_names = iris.feature_names[2:],
                class_names = iris.target_names,
                rounded = True,
                filled=True
        )
```

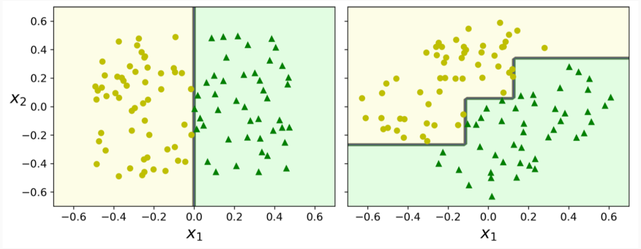
Then use the dot command-line tool from the graphviz package to convert this .dot file to a variety of formats such as PDF or PNG like

```
$ dot —Tpng iris_tree.dot —o iris_tree.png
```

Instability

Decision trees are highly depending on slight variation of the training data, e.g.:

Here we simply rotate the data points by 90 degrees and the prediction of the decision trees will be different.



Even the order of the data in the training set is important. You may get very different models on slightly different datasets → not robust, very high variance

Chapter 07 Summary

- ML Tasks: supervised classification and regression. The model is non-parametric, because it doesn't have a fixed number of parameters. Explainable, flexible, not robust, high variance, low bias.
- Growing the tree: split up the data into smaller groups of similar instances step by step, by
    - start at the root node with all instances from the training set.
    - At each node choose *one feature i* (e.g. petal length) and a threshold τ.
    - Partition the data in two: put instances with $x_i \leq \tau$ in the left node and all instances with $x_i > \tau$ in the right.
    - Continute splitting until you reach a stopping criterion.
    - Remember the splitting rules!
- inference: predict by applying these splitting rules to new instances $\mathbf{x}$ until it ends up in a leaf, and define the prediction $h(\mathbf{x})$ as:
    - for classification, the majority vote, i.e. the class of the most instances in the data of the leaf.
    - for regression, the the mean $\bar{y}$ of the data at the leaf.

- Aim: Splits should decrease the impurity of the instance groups. Measures for impurity:
  - for classification: denote $p_c = \frac{\text{class } c \text{ instances in D}}{\text{number of instances in D}} \in (0,1)$
    - Gini impurity (default):
$$G(D) = 1 - \sum_{c=1}^{n} p_c^2 \in (0,1)$$
    - Entropy:
$$H(D) = - \sum_{\substack{c=1 \\ p_c \neq 0}}^{n} p_c \log_2 (p_c) \, .$$
  - for regression:
$$\text{MSE}(D) = \frac{1}{|D|} \sum_{k \in D} (y_k - \bar{y})^2 .$$
- A CART/decision tree is the ML model with:
  - Prediction function: h as in “inferenceäbove
  - Cost function = CART (=Classification and Regression Tree) cost function
$$L(k, t_k) = \frac{m_{\text{left}}}{m} c_{\text{left}} + \frac{m_{\text{right}}}{m} c_{\text{right}} ,$$
$$\text{where} \begin{cases} c_{\text{left/right}} & = \text{cost function of the left/right subset} \\ m_{\text{left/right}} & = \text{number of instances in the left/right subset} \end{cases}$$

  and c is either Gini impurity or entropy for classification, or MSE for regression.

- The importance of feature i in a decision tree T is given by

$$\text{Importance}_i(T) = \sum_{\text{k}} G_k(c),$$

  where the sum is over all nodes k that are not leaves and use feature i to partition, c is the impurity measure, and $G_k(c) = c(D_k) - c(D_k^L) - c(D_k^R)$ is the reduction in cost when going from node k to its left and right children nodes.
- The tree will grow until it overfits, unless you stop it using regularization hyperparameters:
  - `min_samples_split`= minimum number of samples a node must have before it can be split
  - `min_samples_leaf`= minimum number of samples a leaf node must have
  - `min_weight_fraction_leaf`= same as above but expressed as a fraction of the total number of weighted instances
  - `max_leaf_nodes`= maximum of leaf nodes
  - `max_features`= maximum number of features that are evaluated for splitting at each node

- The complexity of a decision tree is defined as the number of splittings.
- (Post-)Pruning as a method of regularization: process of removing nodes in a trained decision tree. There are two methods:
    - Bottom-up pruning:
        - Simple method - reduced error pruning: for each leaf node, evaluate the effect of removing the split above using a hold-out test set. Leaf nodes are removed only if it results in a drop in the overall cost function on the entire test set. You stop removing nodes when no further improvements can be made.
        - Alternative - Minimal Cost-Complexity Pruning: generate a series of trees and corresponding parameters α (one value for each tree) as follows: at step i, the next tree is created by removing a subtree and replacing it with a leaf node. The subtree is chosen to minimize the difference in classification errors divided by the difference in number of leaves.
    - Top-down pruning - Pessimistic Error Pruning (PEP) (not covered here)
- Complexity:
    - The complexity of growing CART is $O(d \cdot n \cdot m \log_2 m)$, where d is the depth.
    - The inference complexity is independet of the number of features and approximately $O(\log_2 m)$, since at each node (ca. $\log_2 m$ many) only one value needs to be checked.