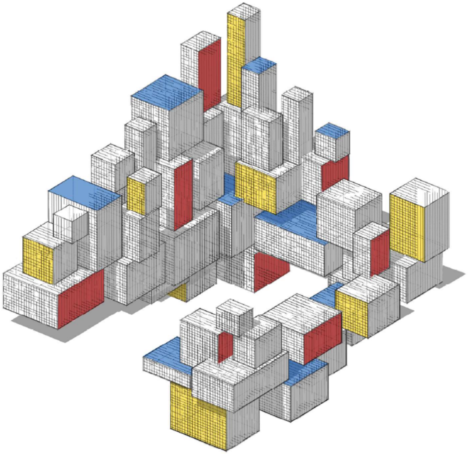
Decision Trees



Purpose

Statistical learning methods based on decision trees have gained tremendous popularity due to their simplicity, intuitive representation, and predictive accuracy. In this lecture we discuss:

- Decision trees for classification
- Construction of decision trees
- Implementation of decision trees

Tree-based Methods

Tree-based methods provide a simple, intuitive, and powerful mechanism for both regression and classification.

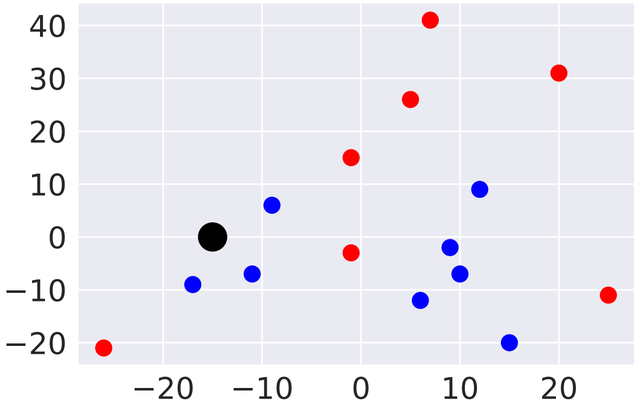
The main idea is to divide a (potentially complicated) feature space $\mathcal{X}$ into smaller regions and fit a simple prediction function to each region.

For example, in a regression setting, one could take the mean of the training responses associated with the training features that fall in that specific region.

In the classification setting, a commonly used prediction function takes the majority vote among the corresponding response variables.

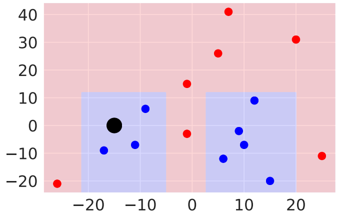
Decision Tree for Classification

The left panel shows a training set of 15 two-dimensional points (features) falling into two classes (red and blue). How should the new feature vector (black point) be classified?



Decision Tree for Classification

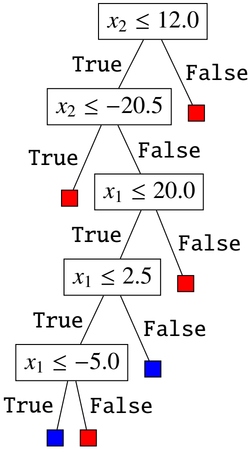
We can partition the feature space $\mathcal{X} = \mathbb{R}^2$ into rectangular regions and assign a class (color) to each region.



The partition thus defines a classifier (prediction function) g that assigns to each feature vector $\boldsymbol{x}$ a class “red” or “blue”. For example, for $\boldsymbol{x} = [-15, 0]^\top$ (solid black point), $g(\boldsymbol{x}) =$ “blue”, since it belongs to a blue region of the feature space.

Decision Tree

Both the classification procedure and the partitioning of the feature space can be represented by a binary decision tree, where each node v corresponds to a region (subset) $\mathcal{R}_v$ of the feature space $\mathcal{X}$.

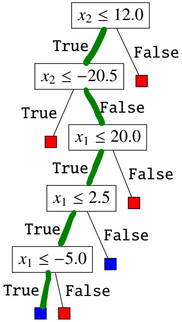


Decision Tree

Each internal node v contains a logical condition that divides $\mathcal{R}_v$ into two disjoint subregions.

The regions of the leaf nodes form a partition of $\mathcal{X}$; associated with each leaf node w is a regional prediction function g^w on $\mathcal{R}_w$.

For input $\boldsymbol{x} = [x_1, x_2]^\top = [-15, 0]^\top$, the classification goes as follows:



Decision Tree

A binary tree $\mathbb{T}$ will partition the feature space $\mathcal{X}$ into as many regions as there are leaf nodes. Denote the set of leaf nodes by $\mathcal{W}$.

The overall prediction function g that corresponds to the tree can then be written as

$$g(\boldsymbol{x}) = \sum_{w \in \mathcal{W}} g^{w}(\boldsymbol{x})\, \mathbb{I}\{\boldsymbol{x} \in \mathcal{R}_w\}, \tag{1}$$

where $\mathbb{I}$ denotes the indicator function.

This representation depends on:

1. How the regions $\{\mathcal{R}_w\}$ are constructed via the logical conditions in the decision tree; e.g., conditions of the form $x_j \leqslant \xi$ split a Euclidean feature space into rectangles aligned with the axes.
2. (2) How the regional prediction functions of the leaf nodes are defined; e.g., a constant function.

Training Loss

Constructing a tree with a training set $\tau = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{n}$ amounts to minimizing the training loss

$$\ell_\tau(g) = \frac{1}{n}\sum_{i=1}^{n} \mathrm{Loss}(y_i, g(\boldsymbol{x}_i)) \tag{2}$$

for some loss function. With g of the form (1), we can write

$$\ell_\tau(g) = \sum_{w \in \mathcal{W}} \underbrace{\frac{1}{n}\sum_{i=1}^{n} \mathbb{I}\{\boldsymbol{x}_i \in \mathcal{R}_w\}\, \mathrm{Loss}(y_i, g^w(\boldsymbol{x}_i))}_{(*)},$$

where $(*)$ is the contribution by the regional prediction function g^w to the overall training loss.

Top-Down Construction of Decision Trees

Finding a decision tree $\mathbb{T}$ that gives a zero squared-error or zero–one training loss is easy, but such an "overfitted" tree will have poor predictive behavior.

Instead we consider a restricted class of decision trees and aim to minimize the training loss within that class. It is common to use the following top-down greedy approach.

Let $\tau = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$ be the training set.

The key to constructing a binary decision tree $\mathbb{T}$ is to specify a splitting rule for each node v, which can be defined as a logical function $s : \mathcal{X} \to \{\texttt{False}, \texttt{True}\}$ or, equivalently, a binary function $s : \mathcal{X} \to \{0, 1\}$.

For example, in our decision tree example, the root node has splitting rule $\boldsymbol{x} \mapsto \mathbb{I}\{x_1 \leqslant 12.0\}$, in correspondence with the logical condition $\{x_1 \leqslant 12.0\}$.

Splitting Rule

During the construction of the tree, each node v is associated with a specific region $\mathcal{R}_v \subseteq \mathcal{X}$ and therefore also the training subset $\{(\boldsymbol{x}, y) \in \tau : \boldsymbol{x} \in \mathcal{R}_v\} \subseteq \tau$.

Using a splitting rule s, we can divide any subset σ of the training set τ into two sets:

$$\sigma_{\texttt{T}} := \{(\boldsymbol{x}, y) \in \sigma \ : \ s(\boldsymbol{x}) = \texttt{True}\} \text{ and } \sigma_{\texttt{F}} := \{(\boldsymbol{x}, y) \in \sigma \ : \ s(\boldsymbol{x}) = \texttt{False}\}$$

Starting from an empty tree and the initial data set τ, a generic decision tree construction takes the form of the recursive Algorithm 1.

Here we use the notation $\mathbb{T}_v$ for a subtree of $\mathbb{T}$ starting from node v.

The final tree $\mathbb{T}$ is thus obtained via $\mathbb{T} = \texttt{Construct\_Subtree}(v_0, \tau)$, where v_0 is the root of the tree.

Algorithm 1: `Construct_Subtree`

Input: A node v and a subset of the training data: $\sigma \subseteq \tau$.

Output: A (sub) decision tree $\mathbb{T}_v$.

```
1  if termination criterion is met then          // $v$ is a leaf node
2      Train a regional prediction function $g^v$ using training data $\sigma$.
3  else                                          // split the node
4      Find the best splitting rule $s_v$ for node $v$.
5      Create successors $v_{\mathtt{T}}$ and $v_{\mathtt{F}}$ of $v$.
6      $\sigma_{\mathtt{T}} \leftarrow \{(\boldsymbol{x}, y) \in \sigma \ : \ s_v(\boldsymbol{x}) = \mathtt{True}\}$
7      $\sigma_{\mathtt{F}} \leftarrow \{(\boldsymbol{x}, y) \in \sigma \ : \ s_v(\boldsymbol{x}) = \mathtt{False}\}$
8      $\mathbb{T}_{v_{\mathtt{T}}} \leftarrow$ Construct_Subtree $(v_{\mathtt{T}}, \sigma_{\mathtt{T}})$     // left branch
9      $\mathbb{T}_{v_{\mathtt{F}}} \leftarrow$ Construct_Subtree $(v_{\mathtt{F}}, \sigma_{\mathtt{F}})$     // right branch
10 return $\mathbb{T}_v$
```

Further Specification

The splitting rule s_v divides the region $\mathcal{R}_v$ into two disjoint parts, say $\mathcal{R}_{v_\mathtt{T}}$ and $\mathcal{R}_{v_\mathtt{F}}$. The corresponding prediction functions, $g^{v_\mathtt{T}}$ and $g^{v_\mathtt{F}}$, satisfy

$$g^v(\boldsymbol{x}) = g^\mathtt{T}(\boldsymbol{x})\,\mathbb{I}\{\boldsymbol{x} \in \mathcal{R}_{v_\mathtt{T}}\} + g^\mathtt{F}(\boldsymbol{x})\,\mathbb{I}\{\boldsymbol{x} \in \mathcal{R}_{v_\mathtt{F}}\}, \quad \boldsymbol{x} \in \mathbb{R}_v.$$

In order to implement the procedure described in Algorithm 1, we need to address:

1. The construction of the regional prediction functions g^v at the leaves (Line 2). In practice very simple prediction functions g^v are used. in Line 2 of Algorithm 1.
2. The specification of the splitting rule (Line 4).
3. The termination criterion (Line 1).

Regional Prediction Functions for Classification

In the *classification* setting with class labels $0, \ldots, c-1$, the regional prediction function g^w for leaf node w is usually chosen to be *constant* and equal to the most common class label of the training data in the associated region $\mathcal{R}_w$ (ties can be broken randomly).

More precisely, let n_w be the number of feature vectors in region $\mathcal{R}_w$ and let

$$p_z^w = \frac{1}{n_w} \sum_{\{(\boldsymbol{x}, y) \in \boldsymbol{\tau} \,:\, \boldsymbol{x} \in \mathcal{R}_w\}} \mathbb{I}_{\{y=z\}},$$

be the proportion of feature vectors in $\mathcal{R}_w$ that have class label $z = 0, \ldots, c-1$. The regional prediction function for node w is chosen to be the constant

$$g^w(\boldsymbol{x}) = \underset{z \in \{0,\ldots,c-1\}}{\operatorname{argmax}} \; p_z^w. \qquad (3)$$

Regional Prediction Functions for Regression

In the *regression* setting, g^w is usually chosen as the mean response in the region; that is,

$$g^w(\boldsymbol{x}) = \overline{y}_{\mathcal{R}_w} := \frac{1}{n_w} \sum_{\{(\boldsymbol{x},y)\in\tau \,:\, \boldsymbol{x}\in\mathcal{R}_w\}} y, \qquad (4)$$

where n_w is again the number of feature vectors in $\mathcal{R}_w$.

It is not difficult to show that $g^v(\boldsymbol{x}) = \overline{y}_{\mathcal{R}_w}$ minimizes the squared-error loss with respect to all constant functions, in the region $\mathcal{R}_w$.

Splitting Rules

In Line 4 in Algorithm 1, we divide region $\mathcal{R}_v$ into two sets, using a splitting rule (function) s_v. Consequently, the data set σ associated with node v (that is, the subset of the original data set τ whose feature vectors lie in $\mathcal{R}_v$), is also split — into $\sigma_{\mathtt{T}}$ and $\sigma_{\mathtt{F}}$. What is the benefit of such a split in terms of a reduction in the training loss? If v were set to a leaf node, its contribution to the training loss would be:

$$\frac{1}{n}\sum_{i=1}^{n} \mathbb{I}_{\{(\boldsymbol{x},y)\in\sigma\}} \operatorname{Loss}(y_i, g^{v}(\boldsymbol{x}_i)). \tag{5}$$

If v were to be split instead, its contribution would be:

$$\frac{1}{n}\sum_{i=1}^{n} \mathbb{I}_{\{(\boldsymbol{x},y)\in\sigma_{\mathtt{T}}\}} \operatorname{Loss}(y_i, g^{\mathtt{T}}(\boldsymbol{x}_i)) + \frac{1}{n}\sum_{i=1}^{n} \mathbb{I}_{\{(\boldsymbol{x},y)\in\sigma_{\mathtt{F}}\}} \operatorname{Loss}(y_i, g^{\mathtt{F}}(\boldsymbol{x}_i)), \tag{6}$$

where $g^{\mathtt{T}}$ and $g^{\mathtt{F}}$ are the prediction functions belonging to the child nodes $v_{\mathtt{T}}$ and $v_{\mathtt{F}}$.

Greedy Rule

A *greedy* heuristic is to pretend that the tree construction algorithm immediately terminates after the split, in which case $v_{\mathtt{T}}$ and $v_{\mathtt{F}}$ are leaf nodes, and $g^{\mathtt{T}}$ and $g^{\mathtt{F}}$ are readily evaluated.

Note that for any splitting rule the contribution (5) is always greater than or equal to (6).

It therefore makes sense to choose the splitting rule such that (6) is minimized.

Moreover, the termination criterion may involve comparing (6) with (5). If their difference is too small it may not be worth further splitting the feature space.

Optimal Splitting Rule

Suppose the feature space is $\mathcal{X} = \mathbb{R}^p$ and we consider "standard" splitting rules of the form

$$s(\boldsymbol{x}) = \mathbb{I}\{x_j \leqslant \xi\}, \tag{7}$$

for some $1 \leqslant j \leqslant p$ and $\xi \in \mathbb{R}$, where we identify 0 with `False` and 1 with `True`.

How should j and ξ be chosen so as to minimize

$$\frac{1}{n}\sum_{i=1}^{n} \mathbb{I}_{\{(\boldsymbol{x},y)\in\sigma_{\mathtt{T}}\}} \operatorname{Loss}(y_i, g^{\mathtt{T}}(\boldsymbol{x}_i)) + \frac{1}{n}\sum_{i=1}^{n} \mathbb{I}_{\{(\boldsymbol{x},y)\in\sigma_{\mathtt{F}}\}} \operatorname{Loss}(y_i, g^{\mathtt{F}}(\boldsymbol{x}_i))?$$

Recall that $g^{\mathtt{T}}$ and $g^{\mathtt{F}}$ are the prediction functions belonging to the child nodes $v_{\mathtt{T}}$ and $v_{\mathtt{F}}$.

Optimal Splitting Rule for Regression

For a regression problem, using a squared-error loss and a constant regional prediction function, the aim is to minimize:

$$\frac{1}{n} \sum_{(\boldsymbol{x},y)\in\tau : x_j \leqslant \xi} (y - \overline{y}_{\mathrm{T}})^2 + \frac{1}{n} \sum_{(\boldsymbol{x},y)\in\tau : x_j > \xi} (y - \overline{y}_{\mathrm{F}})^2 \,, \qquad (8)$$

where $\overline{y}_{\mathrm{T}}$ and $\overline{y}_{\mathrm{F}}$ are the average responses for the σ_{T} and σ_{F} data, respectively.

Let $\{x_{j,k}\}_{k=1}^{m}$ denote the possible values of x_j, $j = 1, \ldots, p$ within the training subset σ (with $m \leqslant n$ elements).

To minimize (8) over all j and ξ, it suffices to evaluate (8) for each of the $m \times p$ values $x_{j,k}$ and then take the minimizing pair $(j, x_{j,k})$.

Optimal Splitting Rule for Classification

For a classification problem, using the indicator loss and a constant regional prediction function, the aim is to choose a splitting rule that minimizes

$$\frac{1}{n} \sum_{(\boldsymbol{x},y)\in\sigma_{\mathrm{T}}} \mathbb{I}\{y \neq y^*_{\mathrm{T}}\} + \frac{1}{n} \sum_{(\boldsymbol{x},y)\in\sigma_{\mathrm{F}}} \mathbb{I}\{y \neq y^*_{\mathrm{F}}\}, \tag{9}$$

where $y^*_{\mathrm{T}} = g^{\mathrm{T}}(\boldsymbol{x})$ is the most prevalent class (majority vote) in the data set σ_{T} and y^*_{F} is the most prevalent class in σ_{F}.

If the feature space is $\mathcal{X} = \mathbb{R}^p$, the optimal standard splitting rule can be obtained in the same way as for the regression case; the only difference is that (8) is replaced with (9).

Impurities

We can view the minimization of (9) as minimizing a weighted average of "impurities" of nodes $\sigma_{\mathtt{T}}$ and $\sigma_{\mathtt{F}}$.

Namely, for an arbitrary training subset $\sigma \subseteq \tau$, if y^* is the most prevalent label, then

$$\frac{1}{|\sigma|} \sum_{(\boldsymbol{x},y)\in\sigma} \mathbb{I}\{y \neq y^*\} = 1 - \frac{1}{|\sigma|} \sum_{(\boldsymbol{x},y)\in\sigma} \mathbb{I}\{y = y^*\} = 1 - p_{y^*} = 1 - \max_{z\in\{0,\dots,c-1\}} p_z,$$

where p_z is the proportion of data points in σ that have class label z, $z = 0, \dots, c-1$.

The quantity $1 - \max_{z\in\{0,\dots,c-1\}} p_z$ measures the diversity of the labels in σ and is called the misclassification impurity.

Consequently, (9) is the weighted sum of the misclassification impurities of $\sigma_{\mathtt{T}}$ and $\sigma_{\mathtt{F}}$, with weights by $|\sigma_{\mathtt{T}}|/n$ and $|\sigma_{\mathtt{F}}|/n$, respectively.

Other Impurity Measures

Note that the misclassification impurity only depends on the label proportions p_z rather than on the individual responses.

Other impurity measures that only depend on the label proportions are entropy impurity:

$$-\sum_{z=0}^{c-1} p_z \log_2(p_z)$$

and the Gini impurity:

$$\frac{1}{2}\left(1-\sum_{z=0}^{c-1} p_z^2\right).$$

All of these impurities are maximal when the label proportions are equal to $1/c$.

Comparison of Impurity Measures

The figure shows similar shapes for various impurity measures for the two-label case, with class probabilities p and $1 - p$.

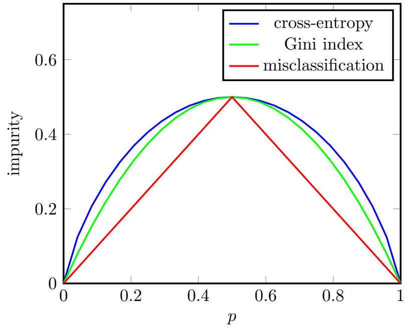


Figure: Entropy, Gini, and misclassification impurities for binary classification, with class frequencies $p_1 = p$ and $p_2 = 1 - p$. The entropy impurity was normalized (divided by 2), to ensure that all impurity measures attain the same maximum value of 1/2 at $p = 1/2$.

Termination Criterion

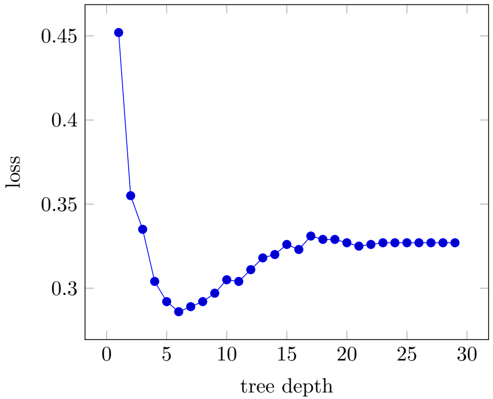
When building a tree, one can define various types of termination conditions. For example:

- Stop when the number of data points in the tree node (the size of the input σ set in Algorithm 1) is less than or equal to some predefined number.
- Choose the maximal depth of the tree in advance.
- Stop when there is no significant advantage, in terms of training loss, to split regions.

Ultimately, the quality of a tree is determined by its predictive performance (generalization risk) and the termination condition should aim to strike a balance between minimizing the approximation error and minimizing the statistical error.

Example: Fixed Tree Depth

The figure illustrates how the tree depth impacts on the generalization risk (estimated by the ten-fold cross-validation loss). The optimal maximal tree depth is here 6.



We used the Python method **make_blobs** from the **sklearn** module to produce a training set of size $n = 5000$ with ten-dimensional feature vectors (thus, $p = 10$ and $\mathcal{X} = \mathbb{R}^{10}$), each of which is classified into one of $c = 3$ classes. The full code is given below.

TreeDepthCV.py

```python
import numpy as np
from sklearn.datasets import make_blobs
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import zero_one_loss
import matplotlib.pyplot as plt

def ZeroOneScore(clf, X, y):
    y_pred = clf.predict(X)
    return zero_one_loss(y, y_pred)

# Construct the training set
X, y =  make_blobs(n_samples=5000, n_features=10, centers=3,
                            random_state=10, cluster_std=10)
```

```
# Construct a decision tree classifier
clf = DecisionTreeClassifier(random_state=0)

# Cross-validation loss as a function of tree depth (1 to 30)
xdepthlist = []
cvlist = []
tree_depth = range(1,30)
for d in tree_depth:
    xdepthlist.append(d)
    clf.max_depth=d
    cv = np.mean(cross_val_score(clf,X,y,cv=10,scoring=ZeroOneScore))
    cvlist.append(cv)

plt.xlabel('tree depth', fontsize=18, color='black')
plt.ylabel('loss', fontsize=18, color='black')
plt.plot(xdepthlist, cvlist,'-*' , linewidth=0.5)
```