Machine Learning

Lecture 3: Decision Trees

Prof. Dr. Aleksandar Bojchevski

17.04.24

Outline

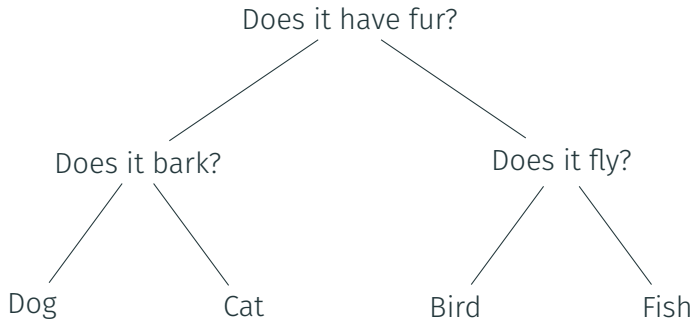
Decision trees blueprint

Building the tree

Impurity measures

Stopping and pruning

The 20-Questions Game

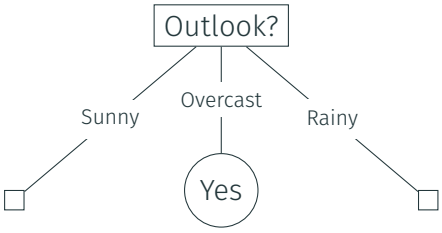


Tennis dataset

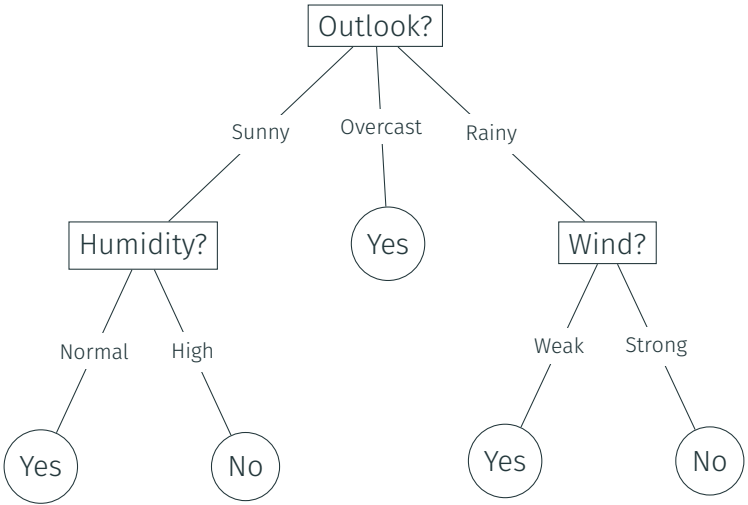
Outlook	Temperature	Humidity	Windy	PlayTennis
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Overcast	Hot	High	False	Yes
Rainy	Mild	High	False	Yes
Rainy	Cool	Normal	False	Yes
Rainy	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Sunny	Mild	High	False	No
Sunny	Cool	Normal	False	Yes
Rainy	Mild	Normal	False	Yes
Sunny	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Rainy	Mild	High	True	No

The root of the decision tree for the tennis dataset

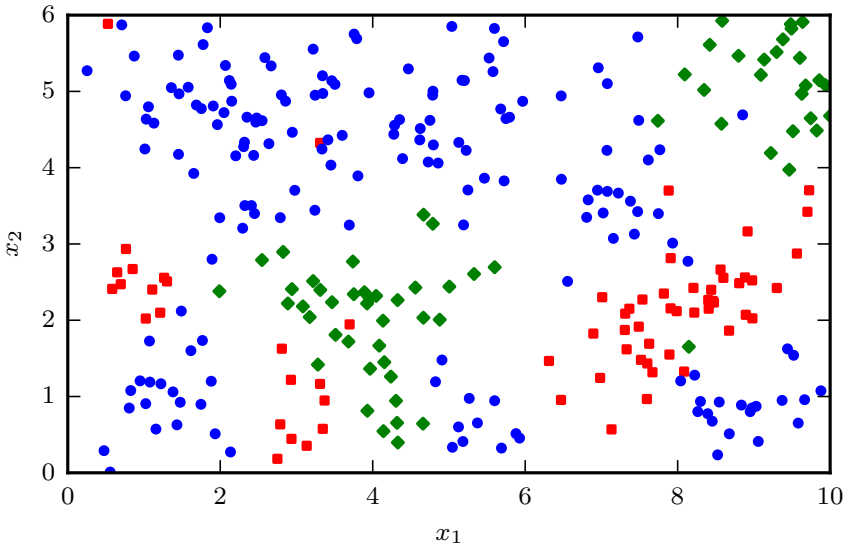
Outlook	Temperature	Humidity	Windy	PlayTennis
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Overcast	Hot	High	False	Yes
Rainy	Mild	High	False	Yes
Rainy	Cool	Normal	False	Yes
Rainy	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Sunny	Mild	High	False	No
Sunny	Cool	Normal	False	Yes
Rainy	Mild	Normal	False	Yes
Sunny	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Rainy	Mild	High	True	No



The final decision tree for the tennis dataset

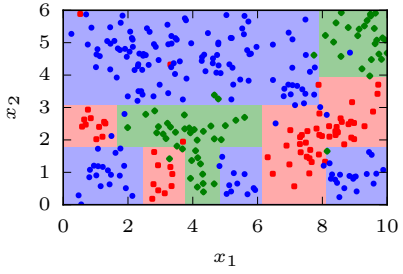


How can we handle numerical features?



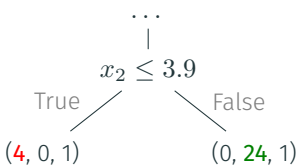
Binary split

Simplest decision: binary split on a single feature, $x_i \leq a$



Interpretation of a decision tree

Decision trees partition the input space into cuboid regions.



Node $\widehat{=}$ *feature* test $\rightarrow$ leads to decision boundaries.

Branch $\widehat{=}$ different outcome of the preceding feature test.

Leaf $\widehat{=}$ region in the input space and the distribution of *samples* in that region.

Inference

To classify a new sample $\boldsymbol{x}$:

1. Test the attributes of $\boldsymbol{x}$ to find the region R that contains it and get the counts of each class $\boldsymbol{n}_R = (n_{c_1,R}, n_{c_2,R}, \ldots, n_{c_k,R})$ for $C = \{c_1, \ldots, c_k\}$.
2. The probability that a data point $\boldsymbol{x} \in R$ is classified as class c is:

$$p(y = c \mid R) = \frac{n_{c,R}}{\sum_{c_i \in C} n_{c_i,R}}$$

3. A new unseen sample $\boldsymbol{x}$ is simply given the label which is most common[1] in its region: $\hat{y} = \arg\max_c \; p(y = c \mid \boldsymbol{x}) = \arg\max_c \; p(y = c \mid R) = \arg\max_c \; n_{c,R}$.

[1] Majority label, similar to k-NN.

Inference

To classify a new sample $\boldsymbol{x}$:

1. Test the attributes of $\boldsymbol{x}$ to find the region R that contains it and get the counts of each class $\boldsymbol{n}_R = (n_{c_1,R}, n_{c_2,R}, \ldots, n_{c_k,R})$ for $C = \{c_1, \ldots, c_k\}$.
2. The probability that a data point $\boldsymbol{x} \in R$ is classified as class c is:

$$p(y = c \mid R) = \frac{n_{c,R}}{\sum_{c_i \in C} n_{c_i,R}}$$

3. A new unseen sample $\boldsymbol{x}$ is simply given the label which is most common[1] in its region: $\hat{y} = \arg\max_c \; p(y = c \mid \boldsymbol{x}) = \arg\max_c \; p(y = c \mid R) = \arg\max_c \; n_{c,R}$.

[1] Majority label, similar to k-NN.

Inference

To classify a new sample $\boldsymbol{x}$:

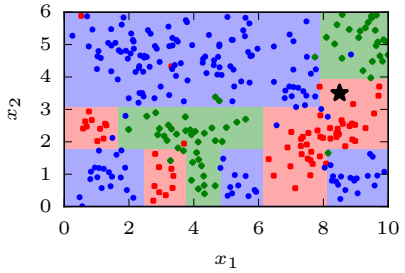
1. Test the attributes of $\boldsymbol{x}$ to find the region R that contains it and get the counts of each class $\boldsymbol{n}_R = (n_{c_1,R}, n_{c_2,R}, \ldots, n_{c_k,R})$ for $C = \{c_1, \ldots, c_k\}$.
2. The probability that a data point $\boldsymbol{x} \in R$ is classified as class c is:

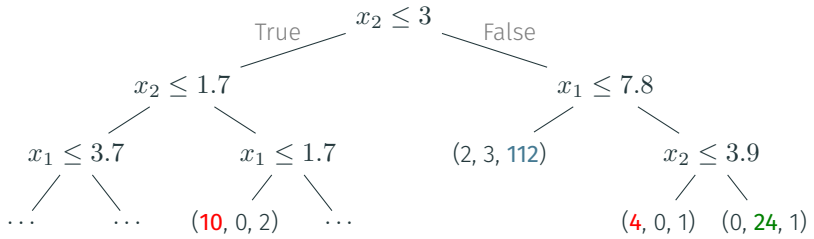
$$p(y = c \mid R) = \frac{n_{c,R}}{\sum_{c_i \in C} n_{c_i,R}}$$

3. A new unseen sample $\boldsymbol{x}$ is simply given the label which is most common[1] in its region: $\hat{y} = \arg\max_c \; p(y = c \mid \boldsymbol{x}) = \arg\max_c \; p(y = c \mid R) = \arg\max_c \; n_{c,R}$.

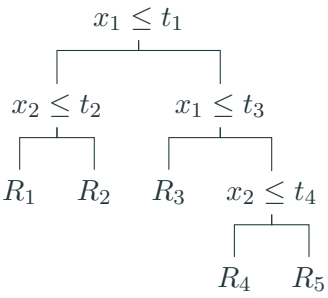
[1] Majority label, similar to k-NN.

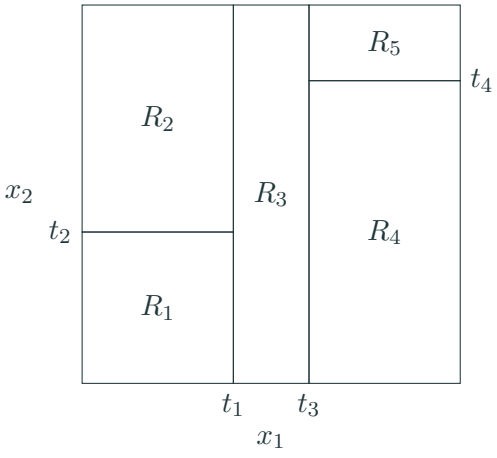
Example prediction: classifcation of $x = (8.5, 3.5)^T$





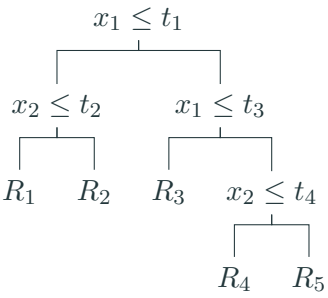
From trees to regions and back

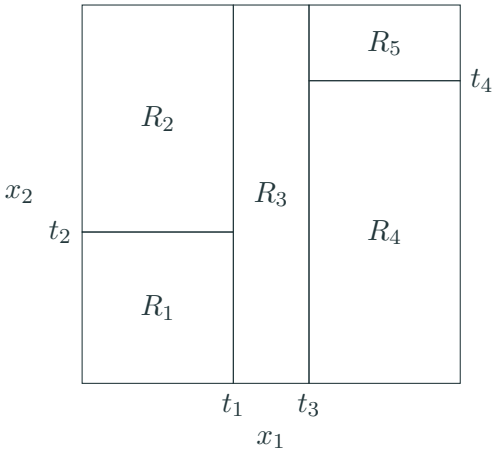




Is there a different equivalent tree that leads to the same regions and back?

From trees to regions and back





Is there a different equivalent tree that leads to the same regions and back?

Outline

Decision trees blueprint

Building the tree

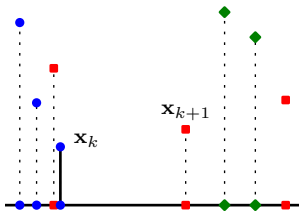
Impurity measures

Stopping and pruning

Naïve idea

Idea: Build all possible trees and evaluate how they perform on new data. All combinations of features and values can serve as tests in the tree.

feature	tests
x_1	≤ 0.36457631
	≤ 0.50120369
	≤ 0.54139549
	$\leq \ldots$
x_2	≤ 0.09652214
	≤ 0.20923062
	$\leq \ldots$



2 features $\times$ 300 unique values per feature, 2 features $\times$ 299 possible thresholds per feature: 598 possible tests at the root node, slightly fewer at each descendant.

Building the optimal decision tree is intractable

Iterating over all possible trees is possible only for very small examples because the number of trees quickly explodes.

Finding the optimal tree is *NP-complete*[2].

Instead: Grow the tree top-down and choose the best split node-by-node using a greedy heuristic on the *training data*.

[2] Optimal in the sense of minimizing the expected number of tests required to classify an unknown sample. Even the problem of identifying the root node in an optimal strategy is NP-hard. And several other aspects of optimal tree construction are known to be intractable.

Example heuristic: misclassification rate

Split the node if it improves the misclassification rate (error) i_E at node t

$$i_E(t) = 1 - \max_c \; p(y = c \mid t)$$

The improvement when performing a split s of t into t_R and t_L for $i(t) = i_E(t)$ is

$$\Delta i(s, t) = i(t) - p_L \cdot i(t_L) - p_R \cdot i(t_R)$$

where p_L and p_R are the fraction of instances in each subtree.

Example: $\Delta i(x_1 \leq 5, t) = 0.5 - 0.5 \cdot 0 - 0.5 \cdot 0.33$

Example heuristic: misclassification rate

Split the node if it improves the misclassification rate (error) i_E at node t

$$i_E(t) = 1 - \max_c \; p(y = c \mid t)$$

The improvement when performing a split s of t into t_R and t_L for $i(t) = i_E(t)$ is

$$\Delta i(s, t) = i(t) - p_L \cdot i(t_L) - p_R \cdot i(t_R)$$

where p_L and p_R are the fraction of instances in each subtree.

Example: $\Delta i(x_1 \leq 5, t) = 0.5 - 0.5 \cdot 0 - 0.5 \cdot 0.33$

Example heuristic: misclassification rate

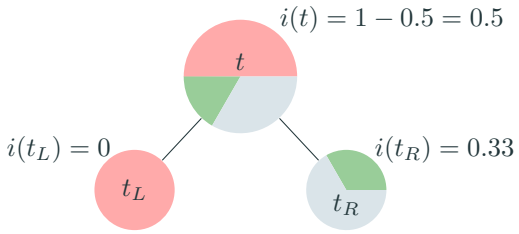
Split the node if it improves the misclassification rate (error) i_E at node t

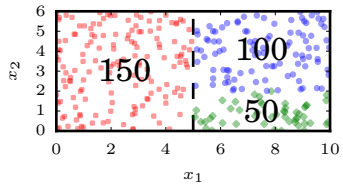
$$i_E(t) = 1 - \max_c \; p(y = c \mid t)$$

The improvement when performing a split s of t into t_R and t_L for $i(t) = i_E(t)$ is

$$\Delta i(s, t) = i(t) - p_L \cdot i(t_L) - p_R \cdot i(t_R)$$

where p_L and p_R are the fraction of instances in each subtree.

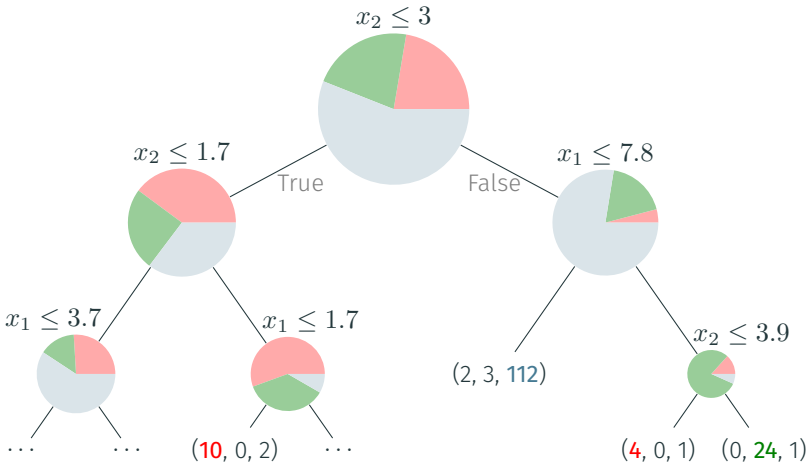




Example: $\Delta i(x_1 \leq 5, t) = 0.5 - 0.5 \cdot 0 - 0.5 \cdot 0.33$

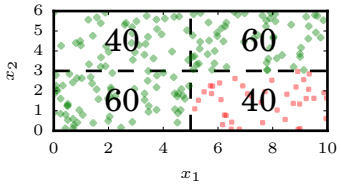
By repeatedly applying the heuristic

The distribution of labels becomes progressively more pure: meaning we mostly have instances of the same class.



Problems with misclassification rate

Problem 1: No split performed even though combining the two tests would result in perfect classification.



Before split: $i_E(t) = \frac{40}{200}$

$x_1 \leq 5$: $p_L \cdot i_E(t_L) + p_R \cdot i_E(t_R) = \frac{40}{200}$

$x_2 \leq 3$: $p_L \cdot i_E(t_L) + p_R \cdot i_E(t_R) = \frac{40}{200}$

Problem 2: No sensitivity to changes in class probability.

- Before split: $(400, 400) \rightarrow i_E(t) = 0.5$
  Split a: $\{(100, 300)\ ,\ (300, 100)\} \rightarrow \Delta i_E(a, t) = 0.25$
  Split b: $\{(200, 400)\ ,\ (200, 0)\} \rightarrow \Delta i_E(b, t) = 0.25$

Outline

Decision trees blueprint

Building the tree

Impurity measures

Stopping and pruning

What is a suitable criterion or impurity measure

Use a criterion $i(t)$ that measures how *pure* the class distribution at a node t is.

It should be

- **maximum** if classes are equally distributed in the node
- **minimum**, usually 0, if the node is pure
- **symmetric**

Impurity measures

Define $\pi_c = p(y = c \mid t)$ for convenience.

Misclassification rate:

$$i_E(t) = 1 - \max_c \ \pi_c$$

Entropy:

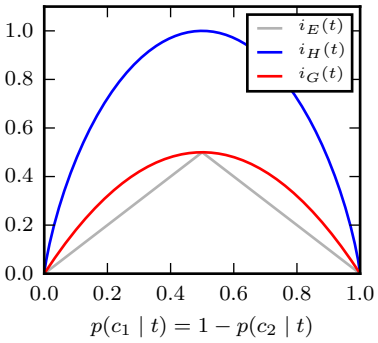
$$i_H(t) = -\sum_{c_i \in C} \pi_{c_i} \log \pi_{c_i}$$

(Note that $\lim_{x \to 0+} x \log x = 0$.)

Gini index:

$$i_G(t) = \sum_{c_i \in C} \pi_{c_i}(1 - \pi_{c_i}) = 1 - \sum_{c_i \in C} \pi_{c_i}^2$$

For $C = \{c_1, c_2\}$:



Shannon Entropy

Expected number of bits needed to encode a randomly drawn value from a distribution (under most efficient code).

For a discrete random variable X with possible values $\{x_1, \ldots, x_n\}$

$$\mathbb{H}(X) = -\sum_{i}^{n} p(X = x_i) \log_2 p(X = x_i)$$

Higher entropy → flatter histogram → sampled values less predictable

Lower entropy → peakier histogram → sampled values more predictable

Detour: Information Theory

Information theory is about encoding and transmitting information.

We would like to encode four messages:

- $m_1 =$ “There is free beer.” $p(m_1) = 0.01$ $\rightarrow$ Code 111
- $m_2 =$ “You have an exam.” $p(m_2) = 0.02$ $\rightarrow$ Code 110
- $m_3 =$ “You have a lecture.” $p(m_3) = 0.30$ $\rightarrow$ Code 10
- $m_4 =$ “Nothing happening.” $p(m_4) = 0.67$ $\rightarrow$ Code 0

The code above is called a *Huffman Code*.

On average:

$$0.01 \times 3\,\text{bits} + 0.02 \times 3\,\text{bits} + 0.3 \times 2\,\text{bits} + 0.67 \times 1\text{bit} = 1.36\,\text{bits}$$

Detour: Information Theory

Information theory is about encoding and transmitting information.

We would like to encode four messages:

- $m_1 =$ “There is free beer.” $p(m_1) = 0.01$ $\rightarrow$ Code 111
- $m_2 =$ “You have an exam.” $p(m_2) = 0.02$ $\rightarrow$ Code 110
- $m_3 =$ “You have a lecture.” $p(m_3) = 0.30$ $\rightarrow$ Code 10
- $m_4 =$ “Nothing happening.” $p(m_4) = 0.67$ $\rightarrow$ Code 0

The code above is called a *Huffman Code*.

On average:

$$0.01 \times 3\,\text{bits} + 0.02 \times 3\,\text{bits} + 0.3 \times 2\,\text{bits} + 0.67 \times 1\text{bit} = 1.36\,\text{bits}$$

Gini index

Measures how often a randomly chosen instance would be misclassified if it was randomly classified according to the class distribution.

$$i_G(t) = \sum_{c_i \in C} \underbrace{\pi_{c_i}}_{\substack{\text{probability of} \\ \text{picking element}}} \cdot \underbrace{(1 - \pi_{c_i})}_{\substack{\text{probability is} \\ \text{misclassified}}}$$

Entropy vs Gini Index:

- It only matters in 2% of the cases which one you use.[3]
- Gini Index small advantage: no need to compute $\log$ which can be a bit faster.

[3] See Raileanu LE, Stoffel K. Theoretical comparison between the gini index and information gain criteria.

Gini index

Measures how often a randomly chosen instance would be misclassified if it was randomly classified according to the class distribution.

$$i_G(t) = \sum_{c_i \in C} \underbrace{\pi_{c_i}}_{\substack{\text{probability of} \\ \text{picking element}}} \cdot \underbrace{(1 - \pi_{c_i})}_{\substack{\text{probability is} \\ \text{misclassified}}}$$

Entropy vs Gini Index:

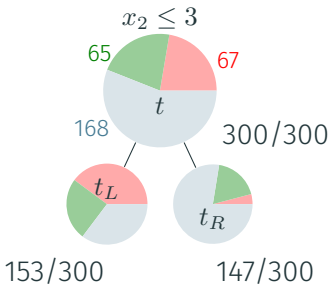
- It only matters in 2% of the cases which one you use.[3]
- Gini Index small advantage: no need to compute $\log$ which can be a bit faster.

[3] See Raileanu LE, Stoffel K. Theoretical comparison between the gini index and information gain criteria.

Building a decision tree

Compare all possible tests and choose the one where the improvement $\Delta i(s,t)$ for some splitting criterion $i(t)$ is largest.

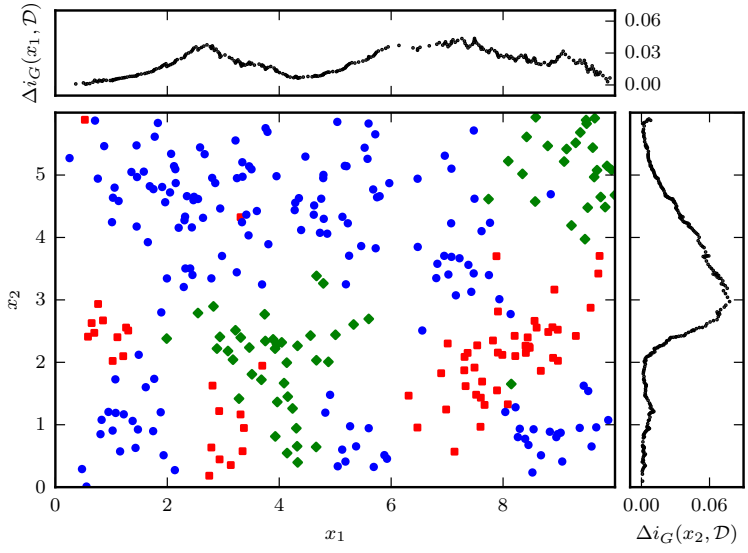
$$i_G(t) = 1 - \left(\tfrac{67}{300}\right)^2 - \left(\tfrac{65}{300}\right)^2 - \left(\tfrac{168}{300}\right)^2 \approx 0.5896$$



After testing $x_2 \leq 3$: $i_G(t_L) \approx 0.6548$ and $i_G(t_R) \approx 0.3632$

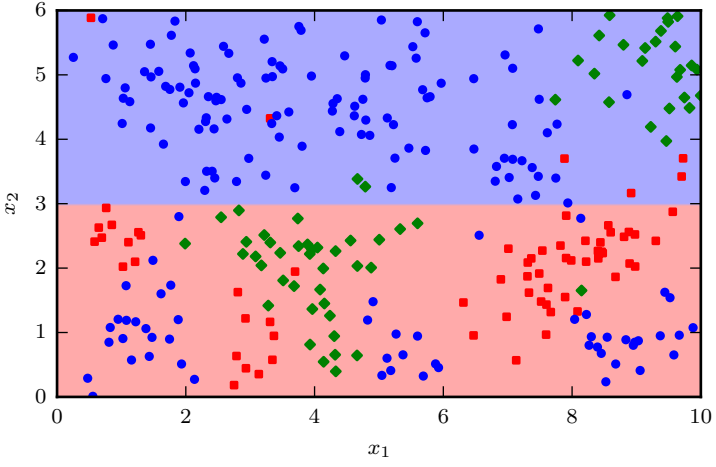
So $\Delta i_G(x_2 \leq 3, t) = i_G(t) - \frac{153}{300} \cdot i_G(t_L) - \frac{147}{300} \cdot i_G(t_R) \approx 0.07768$

Example: Improvement using Gini index



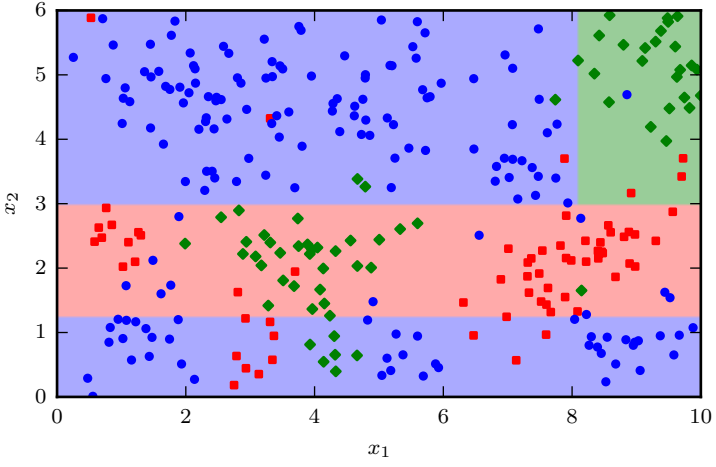
Decision boundary at depth 1

Accuracy on the whole data set: 58.3%



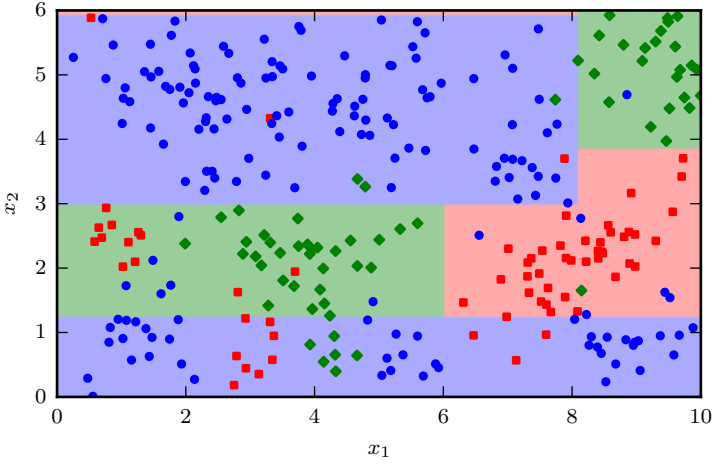
Decision boundaries at depth 2

Accuracy on the whole data set: 77%



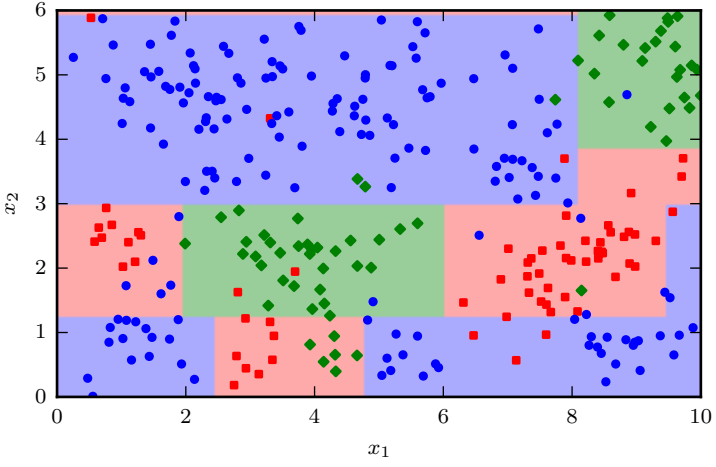
Decision boundaries at depth 3

Accuracy on the whole data set: 84.3%



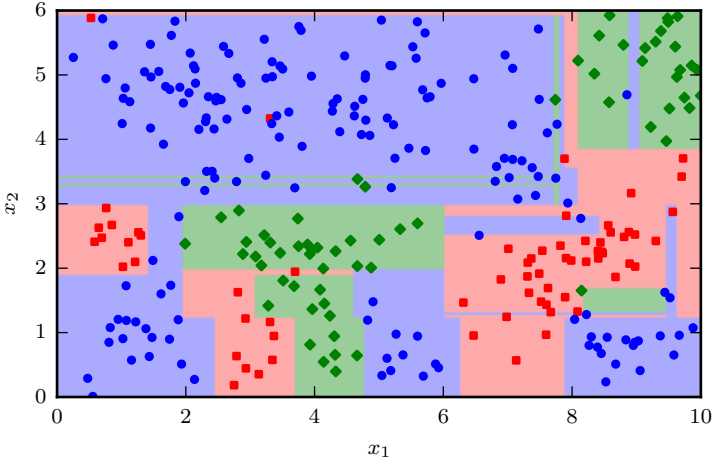
Decision boundaries at depth 4

Accuracy on the whole data set: 90.3%



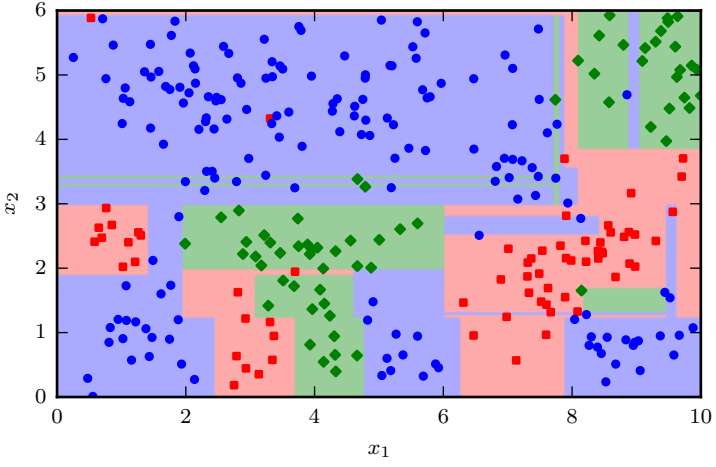
Decision boundaries of a maximally pure tree

Accuracy on the whole data set: 100% → *Good generalization?*



Decision boundaries of a maximally pure tree

Accuracy on the whole data set: 100% $\rightarrow$ *Good generalization*?



Outline

Decision trees blueprint

Building the tree

Impurity measures

Stopping and pruning

Stopping criterion

We are recursively splitting the data, thereby growing the DT. When to stop?

Possible stopping (or *pre-pruning*) criteria:

- distribution in branch is *pure*, i.e $i(t) = 0$
- maximum depth reached (we prefer shallow trees based on *Occam's razor*)
- number of samples in each branch below certain threshold
- benefit of splitting is below certain threshold $\Delta i(s, t) < t_\Delta$
- accuracy on the validation set

Disadvantage: We could miss a good split if we stop too early.

Alternatively we can grow a tree maximally and then (post-)prune it.

Stopping criterion

We are recursively splitting the data, thereby growing the DT. When to stop?

Possible stopping (or *pre-pruning*) criteria:

- distribution in branch is *pure*, i.e $i(t) = 0$
- maximum depth reached (we prefer shallow trees based on *Occam's razor*)
- number of samples in each branch below certain threshold
- benefit of splitting is below certain threshold $\Delta i(s,t) < t_\Delta$
- accuracy on the validation set

Disadvantage: We could miss a good split if we stop too early.

Alternatively we can grow a tree maximally and then (post-)prune it.

Stopping criterion

We are recursively splitting the data, thereby growing the DT. When to stop?

Possible stopping (or *pre-pruning*) criteria:

- distribution in branch is *pure*, i.e $i(t) = 0$
- maximum depth reached (we prefer shallow trees based on *Occam's razor*)
- number of samples in each branch below certain threshold
- benefit of splitting is below certain threshold $\Delta i(s,t) < t_\Delta$
- accuracy on the validation set

Disadvantage: We could miss a good split if we stop too early.

Alternatively we can grow a tree maximally and then (post-)prune it.

Stopping criterion

We are recursively splitting the data, thereby growing the DT. When to stop?

Possible stopping (or *pre-pruning*) criteria:

- distribution in branch is *pure*, i.e $i(t) = 0$
- maximum depth reached (we prefer shallow trees based on *Occam's razor*)
- number of samples in each branch below certain threshold
- benefit of splitting is below certain threshold $\Delta i(s,t) < t_\Delta$
- accuracy on the validation set

Disadvantage: We could miss a good split if we stop too early.

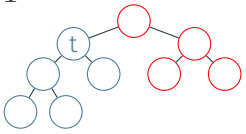
Alternatively we can grow a tree maximally and then (post-)prune it.

Reduced error pruning

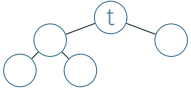
Let T be our decision tree and t one of its inner nodes.

Pruning T w.r.t. t means deleting all descendant nodes of t (but not t itself). We denote the pruned tree $T \backslash T_t$.

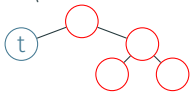
T



T_t



$T \backslash T_t$



1. Use validation set to get an error estimate: $\mathrm{err}_{\mathcal{D}_V}(T)$.
2. For each node t calculate $\mathrm{err}_{\mathcal{D}_V}(T \backslash T_t)$
3. Prune the tree at the node that yields the highest error reduction.
4. Repeat until for all nodes t: $\mathrm{err}_{\mathcal{D}_V}(T) > \mathrm{err}_{\mathcal{D}_V}(T \backslash T_t)$.

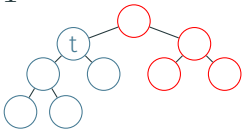
After pruning you may use both training and validation data to update the leaves.

Reduced error pruning

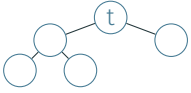
Let T be our decision tree and t one of its inner nodes.

Pruning T w.r.t. t means deleting all descendant nodes of t (but not t itself). We denote the pruned tree $T \backslash T_t$.

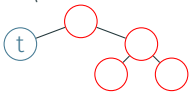
T



T_t



$T \backslash T_t$



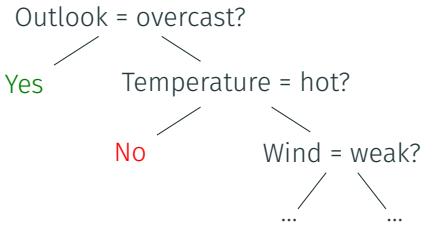
1. Use validation set to get an error estimate: $\mathrm{err}_{\mathcal{D}_V}(T)$.
2. For each node t calculate $\mathrm{err}_{\mathcal{D}_V}(T \backslash T_t)$
3. Prune the tree at the node that yields the highest error reduction.
4. Repeat until for all nodes t: $\mathrm{err}_{\mathcal{D}_V}(T) > \mathrm{err}_{\mathcal{D}_V}(T \backslash T_t)$.

After pruning you may use both training and validation data to update the leaves.

Decision trees with categorical features

Different algorithm variants (ID3, C4.5, CART) handle these things differently.

Day	Outlook	Temperature	Humidity	Wind	Play Tennis?
D1	sunny	hot	high	weak	No
D2	sunny	hot	high	strong	No
...					

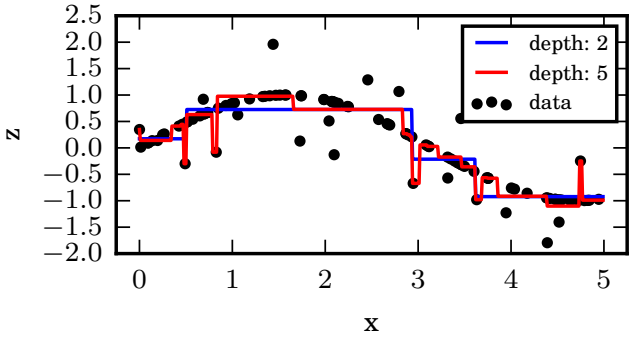


Decision trees for regression

For regression when y_i is a real value rather than a class:

At the leaves compute the mean (instead of the mode) over the instances.

Use the mean-squared error as a splitting heuristic.



Handling missing input features

We could fill missing values (e.g. with the mean), but for trees we can do better.

Categorical predictors: make a new category for “missing”.

General approach is to introduce **surrogate** variables:

- Define primary features (and split) as usual
- Form a list of surrogate features
- First surrogate is the feature (and split) that best mimics the primary
- Second surrogate is second-best, etc.
- If the feature is missing, use the surrogates

Another approach is to average across all missing branches.

Considerations

Decision trees are human interpretable.

Can handle any **combination** of numerical and categorical features and targets.

Extensions (e.g. random forests, boosted trees – that will be discussed in the future) have very competitive performance (e.g. Kaggle competitions).

Insensitive to monotone transformations of the inputs.

Fast to fit and scale well to large datasets.

Compared to k-NN:

- Much better complexity w.r.t. memory/storage and inference
- More flexible decision function

Summary

Decision trees make predictions by performing a series of tests which partition the input space into cuboid regions.

Finding the optimal tree in NP-complete so we build trees top-down with a greedy heuristic.

Stopping criteria or pruning to improve the generalizability of the model.

Reading material

Main reading

- ”Probabilistic Machine Learning: An Introduction” by Murphy [ch. 18.1]

Some slides adapted from a previous version by S. Günnemann.