MADS-ML – Machine Learning

Decision Trees

Prof. Dr. Stephan Doerfel

FACHHOCHSCHULE KIEL
University of Applied Sciences

Moodle (WiSe 2024/25)

"if-then-else"

Outline

Basics

Construction

Overfitting

Decision Trees in Python

Basic Idea 1/2

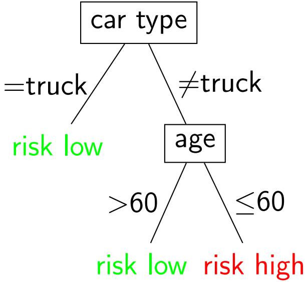
- think of the game 20 questions

Basic Idea 1/2

- think of the game 20 questions
- drawback of kNN: No explanation on how which features yield which decision

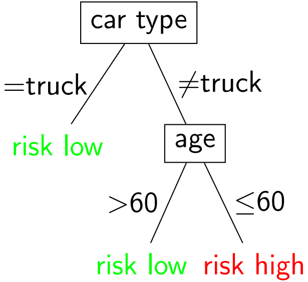
Motivation

ID	age	car type	risk
1	23	family	high
2	17	sports	high
3	43	sports	high
4	68	family	low
5	32	truck	low



Motivation

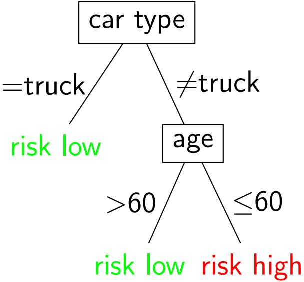
ID	age	car type	risk
1	23	family	high
2	17	sports	high
3	43	sports	high
4	68	family	low
5	32	truck	low



- decision trees find **explicit** knowledge

Motivation

ID	age	car type	risk
1	23	family	high
2	17	sports	high
3	43	sports	high
4	68	family	low
5	32	truck	low



- decision trees find **explicit** knowledge
- decision trees provide **explanations** for their decisions (white box)

Rooted Trees

Definition 1 (Tree)

Rooted Trees

Definition 1 (Tree)

- A **graph** is a data structure that consists of a non-empty set of nodes and as set of edges, each connecting two nodes.

Rooted Trees

Definition 1 (Tree)

- A **graph** is a data structure that consists of a non-empty set of nodes and as set of edges, each connecting two nodes.
- A **tree** is a graph that is connected and acyclic, i.e.,
  - there is a path between any two nodes and
  - there is no path of edges starting and ending at the same node.

Rooted Trees

Definition 1 (Tree)

- A **graph** is a data structure that consists of a non-empty set of nodes and as set of edges, each connecting two nodes.
- A **tree** is a graph that is connected and acyclic, i.e.,
  - there is a path between any two nodes and
  - there is no path of edges starting and ending at the same node.

Definition 2 (Rooted Tree)

A rooted tree is a graph in which one node is designated the **root**. The root induces a direction on the edges, i.e. all edges point away from the root.

Rooted Trees – Remarks

- If an edge points from B to A, then A is B's child and B is A's **parent**.

Rooted Trees – Remarks

- If an edge points from B to A, then A is B's child and B is A's **parent**.
- Each node has exactly one parent with the exception of the root which has none.

Rooted Trees – Remarks

- If an edge points from B to A, then A is B's child and B is A's **parent**.
- Each node has exactly one parent with the exception of the root which has none.
- Nodes without children are called **leafs**.

Rooted Trees – Remarks

- If an edge points from B to A, then A is B's child and B is A's **parent**.
- Each node has exactly one parent with the exception of the root which has none.
- Nodes without children are called **leafs**.
- Nodes that are not leafs are called **inner nodes**.

Rooted Trees – Remarks

- If an edge points from B to A, then A is B's child and B is A's **parent**.
- Each node has exactly one parent with the exception of the root which has none.
- Nodes without children are called **leafs**.
- Nodes that are not leafs are called **inner nodes**.
- children, children of children, ... are called descendants

Rooted Trees – Remarks

- If an edge points from B to A, then A is B's child and B is A's **parent**.
- Each node has exactly one parent with the exception of the root which has none.
- Nodes without children are called **leafs**.
- Nodes that are not leafs are called **inner nodes**.
- children, children of children, ... are called descendants
- one node A with all its descendants forms another binary rooted tree, called the **branch** or **subtree** of A

Rooted Trees – Further Notions

level of a node: Number of steps from the root

Rooted Trees – Further Notions

level of a node: Number of steps from the root

width of a level: Number of nodes on that level

Rooted Trees – Further Notions

level of a node: Number of steps from the root

width of a level: Number of nodes on that level

depth of a tree: Highest level

Rooted Trees – Further Notions

level of a node: Number of steps from the root

width of a level: Number of nodes on that level

depth of a tree: Highest level

breadth of a tree: Number of leafs

Decision Trees: Basics

- A decision tree is a binary rooted tree with the following properties:

Decision Trees: Basics

- A decision tree is a binary rooted tree with the following properties:
  - each inner node represents a test / question on one feature

Decision Trees: Basics

- A decision tree is a binary rooted tree with the following properties:
  - each inner node represents a test / question on one feature
  - each edge denotes on possible outcome of a parent-node's test

Decision Trees: Basics

- A decision tree is a binary rooted tree with the following properties:
  - each inner node represents a test / question on one feature
  - each edge denotes on possible outcome of a parent-node's test
  - a leaf represents the decision for one particular class

Decision Trees: Basics

- A decision tree is a binary rooted tree with the following properties:
  - each inner node represents a test / question on one feature
  - each edge denotes on possible outcome of a parent-node's test
  - a leaf represents the decision for one particular class
- Construction

Decision Trees: Basics

- A decision tree is a binary rooted tree with the following properties:
  - each inner node represents a test / question on one feature
  - each edge denotes on possible outcome of a parent-node's test
  - a leaf represents the decision for one particular class
- Construction
  - based on the training dataset

Decision Trees: Basics

- A decision tree is a binary rooted tree with the following properties:
  - each inner node represents a test / question on one feature
  - each edge denotes on possible outcome of a parent-node's test
  - a leaf represents the decision for one particular class
- Construction
  - based on the training dataset
  - Top-Down

Decision Trees: Basics

- A decision tree is a binary rooted tree with the following properties:
  - each inner node represents a test / question on one feature
  - each edge denotes on possible outcome of a parent-node's test
  - a leaf represents the decision for one particular class
- Construction
  - based on the training dataset
  - Top-Down
- Classification:

Decision Trees: Basics

- A decision tree is a binary rooted tree with the following properties:
  - each inner node represents a test / question on one feature
  - each edge denotes on possible outcome of a parent-node's test
  - a leaf represents the decision for one particular class
- Construction
  - based on the training dataset
  - Top-Down
- Classification:
  - take a data instance w and conduct the first test (root node)

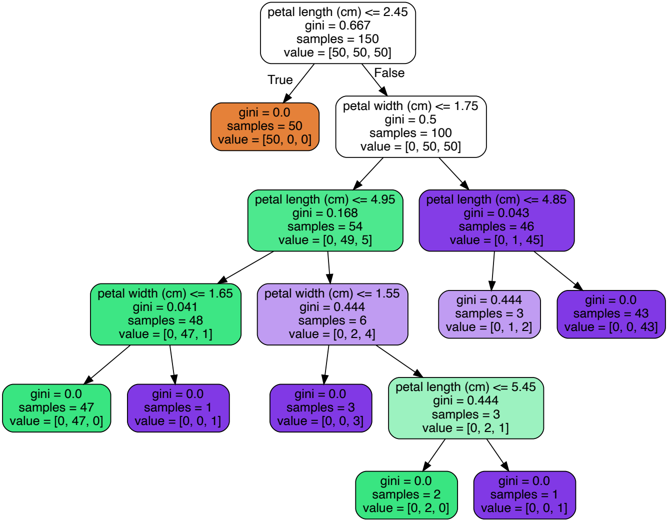
Decision Trees: Basics

- A decision tree is a binary rooted tree with the following properties:
  - each inner node represents a test / question on one feature
  - each edge denotes on possible outcome of a parent-node's test
  - a leaf represents the decision for one particular class
- Construction
  - based on the training dataset
  - Top-Down
- Classification:
  - take a data instance w and conduct the first test (root node)
  - follow the decision tree from the root until a leaf is reached (unique path!)

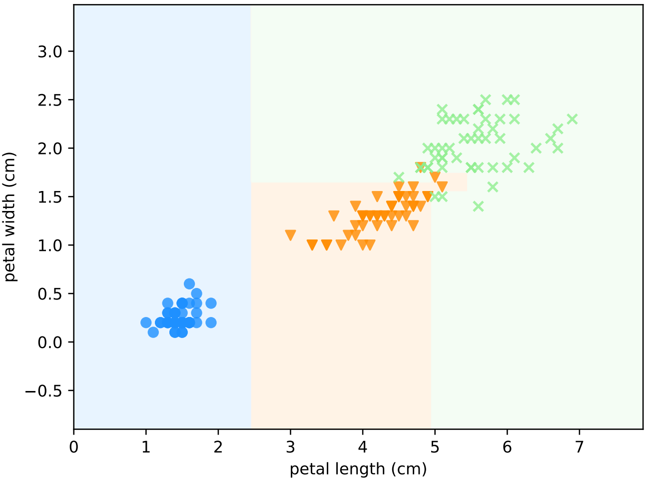
Decision Trees: Basics

- A decision tree is a binary rooted tree with the following properties:
  - each inner node represents a test / question on one feature
  - each edge denotes on possible outcome of a parent-node's test
  - a leaf represents the decision for one particular class
- Construction
  - based on the training dataset
  - Top-Down
- Classification:
  - take a data instance w and conduct the first test (root node)
  - follow the decision tree from the root until a leaf is reached (unique path!)
  - predict the class corresponding to that leaf.

An Example – The Iris Dataset (Features 2 and 3)



An Example – The Iris Dataset (Features 2 and 3)



Remark: Notice the combination of rectangular regions!

Outline

Basics

Construction

Overfitting

Decision Trees in Python

Construction of a Decision Tree

algorithm:

Construction of a Decision Tree

algorithm:

1. Create a root node and assign to it the full training dataset.

Construction of a Decision Tree

algorithm:

1. Create a root node and assign to it the full training dataset.
2. Consider various potential splits (**split candidates**) and select one split using a **split strategy**

Construction of a Decision Tree

algorithm:

1. Create a root node and assign to it the full training dataset.
2. Consider various potential splits (**split candidates**) and select one split using a **split strategy**
   - categorical features: a feature and subsets of its values

Construction of a Decision Tree

algorithm:

1. Create a root node and assign to it the full training dataset.
2. Consider various potential splits (**split candidates**) and select one split using a **split strategy**
   - categorical features: a feature and subsets of its values
   - continuous features: a feature and split points (buckets) of its values

Construction of a Decision Tree

algorithm:

1. Create a root node and assign to it the full training dataset.
2. Consider various potential splits (**split candidates**) and select one split using a **split strategy**
   - categorical features: a feature and subsets of its values
   - continuous features: a feature and split points (buckets) of its values
3. Partition the data of the node using the split and create one child for each partition.

Construction of a Decision Tree

algorithm:

1. Create a root node and assign to it the full training dataset.
2. Consider various potential splits (**split candidates**) and select one split using a **split strategy**
   - categorical features: a feature and subsets of its values
   - continuous features: a feature and split points (buckets) of its values
3. Partition the data of the node using the split and create one child for each partition.
4. Repeat 2 and 3 recursively for each new node.
   → locally optimizing algorithm (greedy approach)

Construction of a Decision Tree

algorithm:

1. Create a root node and assign to it the full training dataset.
2. Consider various potential splits (**split candidates**) and select one split using a **split strategy**
   - categorical features: a feature and subsets of its values
   - continuous features: a feature and split points (buckets) of its values
3. Partition the data of the node using the split and create one child for each partition.
4. Repeat 2 and 3 recursively for each new node.
   → locally optimizing algorithm (greedy approach)

Abort criteria:

Construction of a Decision Tree

algorithm:

1. Create a root node and assign to it the full training dataset.
2. Consider various potential splits (**split candidates**) and select one split using a **split strategy**
   - categorical features: a feature and subsets of its values
   - continuous features: a feature and split points (buckets) of its values
3. Partition the data of the node using the split and create one child for each partition.
4. Repeat 2 and 3 recursively for each new node.
   → locally optimizing algorithm (greedy approach)

Abort criteria:

- all data instances in the current node belong to the same class

Construction of a Decision Tree

algorithm:

1. Create a root node and assign to it the full training dataset.
2. Consider various potential splits (**split candidates**) and select one split using a **split strategy**
   - categorical features: a feature and subsets of its values
   - continuous features: a feature and split points (buckets) of its values
3. Partition the data of the node using the split and create one child for each partition.
4. Repeat 2 and 3 recursively for each new node.
   ➔ locally optimizing algorithm (greedy approach)

Abort criteria:

- all data instances in the current node belong to the same class
- no more split attributes available

Decision Tree: Example

day	forecast	temperature	humidity	wind	tennis?
1	sunny	hot	high	weak	no
2	sunny	hot	high	strong	no
3	overcast	hot	high	weak	yes
4	rain	mild	high	weak	yes
5	rain	cool	normal	weak	yes
6	rain	cool	normal	strong	no

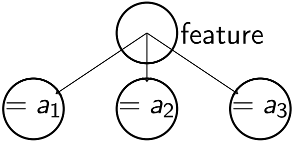
Is this a day to play tennis?

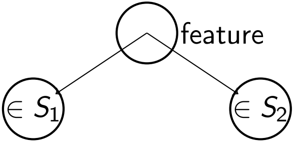
Let's try a split by forecast, . . .

Split Types

Categorical Features:

- split condition of the form "feature $= a$" or "feature in $\in$ set"
- many possible subsets

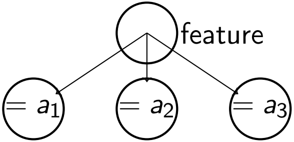


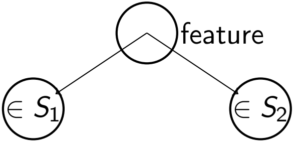


Split Types

Categorical Features:

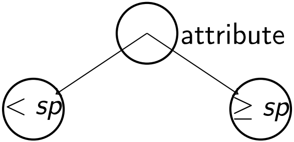
- split condition of the form "feature $= a$" or "feature in $\in$ set"
- many possible subsets





numerical features:

- split condition of the form "feature $< a$"
- many possible splits



Measuring the quality of a split

Setting:

Measuring the quality of a split

Setting:

- a set of training instances T,

Measuring the quality of a split

Setting:

- a set of training instances T,
- a given partition $T_1, T_2, \ldots, T_m$ von T

Measuring the quality of a split

Setting:

- a set of training instances T,
- a given partition $T_1, T_2, \dots, T_m$ von T
- p_i relative frequency of class c_i in T.

Measuring the quality of a split

Setting:

- a set of training instances T,
- a given partition $T_1, T_2, \ldots, T_m$ von T
- p_i relative frequency of class c_i in T.

Required:

Measuring the quality of a split

Setting:

- a set of training instances T,
- a given partition $T_1, T_2, \ldots, T_m$ von T
- p_i relative frequency of class c_i in T.

Required:

- a measure of impurity of a set S of training instances with regard to class membership

Measuring the quality of a split

Setting:

- a set of training instances T,
- a given partition $T_1, T_2, \ldots, T_m$ von T
- p_i relative frequency of class c_i in T.

Required:

- a measure of impurity of a set S of training instances with regard to class membership
- a measure of **information gain** that assesses the quality of a split

Measuring the quality of a split

Setting:

- a set of training instances T,
- a given partition $T_1, T_2, \ldots, T_m$ von T
- p_i relative frequency of class c_i in T.

Required:

- a measure of impurity of a set S of training instances with regard to class membership
- a measure of **information gain** that assesses the quality of a split
- a split of T into $T_1, T_2, \ldots, T_m$, that minimizes this measure

Split strategy: Misclassification Error?

- **Misclassification error** of a node is the error probability when the node votes for its most frequent class (counted among its training instances).

Split strategy: Misclassification Error?

- **Misclassification error** of a node is the error probability when the node votes for its most frequent class (counted among its training instances).
- The **misclassification error** of a set T of training instances w.r.t. a set of classes $c_1, c_2, \ldots, c_k$ is defined as

$$error(T) = 1 - \max_{i=1}^{k} p_i$$

Split strategy: Misclassification Error?

- **Misclassification error** of a node is the error probability when the node votes for its most frequent class (counted among its training instances).
- The **misclassification error** of a set T of training instances w.r.t. a set of classes $c_1, c_2, \ldots, c_k$ is defined as

$$error(T) = 1 - \max_{i=1}^{k} p_i$$

- Issues:

Split strategy: Misclassification Error?

- **Misclassification error** of a node is the error probability when the node votes for its most frequent class (counted among its training instances).
- The **misclassification error** of a set T of training instances w.r.t. a set of classes $c_1, c_2, \ldots, c_k$ is defined as

$$error(T) = 1 - \max_{i=1}^{k} p_i$$

- Issues:
  - Misclassification error treats node as leaf (even though we might split it again during the further construction).

Split strategy: Misclassification Error?

- **Misclassification error** of a node is the error probability when the node votes for its most frequent class (counted among its training instances).
- The **misclassification error** of a set T of training instances w.r.t. a set of classes $c_1, c_2, \ldots, c_k$ is defined as

$$error(T) = 1 - \max_{i=1}^{k} p_i$$

- Issues:
  - Misclassification error treats node as leaf (even though we might split it again during the further construction).
  - It depends only on the largest class among the training instances, the distribution on the other classes is ignored.

Split Strategy: Information Gain 1/2

- Entropy is a measure from information theory

Split Strategy: Information Gain 1/2

- Entropy is a measure from information theory
- Entropy measures impurity of a set. Lower means “more pure”.

Split Strategy: Information Gain 1/2

- Entropy is a measure from information theory
- Entropy measures impurity of a set. Lower means "more pure".
- **Entropy** of a set T of training instances w.r.t. a set of classes $c_1, c_2, \ldots, c_k$ is defined as

$$entropy(T) = -\sum_{i=1}^{k} p_i \cdot \log_2 p_i$$

Observations:

Split Strategy: Information Gain 1/2

- Entropy is a measure from information theory
- Entropy measures impurity of a set. Lower means "more pure".
- **Entropy** of a set T of training instances w.r.t. a set of classes $c_1, c_2, \ldots, c_k$ is defined as

$$entropy(T) = -\sum_{i=1}^{k} p_i \cdot \log_2 p_i$$

  Observations:
  - $entropy(T) = 0$ if $p_i = 1$ for one class c_i (with the convention that $0 \cdot \log_2 0 = 0$),

Split Strategy: Information Gain 1/2

- Entropy is a measure from information theory
- Entropy measures impurity of a set. Lower means “more pure”.
- **Entropy** of a set T of training instances w.r.t. a set of classes $c_1, c_2, \ldots, c_k$ is defined as

$$entropy(T) = -\sum_{i=1}^{k} p_i \cdot \log_2 p_i$$

Observations:

- $entropy(T) = 0$ if $p_i = 1$ for one class c_i (with the convention that $0 \cdot \log_2 0 = 0$),
- $entropy(T)$ is maximal for $p_1 = \cdots = p_k = \frac{1}{k}$.

Split Strategy: Information Gain 1/2

- Entropy is a measure from information theory
- Entropy measures impurity of a set. Lower means "more pure".
- **Entropy** of a set T of training instances w.r.t. a set of classes $c_1, c_2, \ldots, c_k$ is defined as

$$entropy(T) = -\sum_{i=1}^{k} p_i \cdot \log_2 p_i$$

Observations:

- $entropy(T) = 0$ if $p_i = 1$ for one class c_i (with the convention that $0 \cdot \log_2 0 = 0$),
- $entropy(T)$ is maximal for $p_1 = \cdots = p_k = \frac{1}{k}$.
- $entropy(T) = 1$ for $k = 2$ with $p_i = \frac{1}{2}$.

Split Strategy: Information Gain 2/2

- Let a split A create the partition $T_1, T_2, \ldots, T_m$ of T.

Split Strategy: Information Gain 2/2

- Let a split A create the partition $T_1, T_2, \ldots, T_m$ of T.
- The **Information Gain** of split A w.r.t. T is:

$$informationGain(T, A) = entropy(T) - \sum_{i=1}^{m} \frac{|T_i|}{|T|} \cdot entropy(T_i)$$

Split Strategy: Information Gain 2/2

- Let a split A create the partition $T_1, T_2, \ldots, T_m$ of T.
- The **Information Gain** of split A w.r.t. T is:

$$informationGain(T, A) = entropy(T) - \sum_{i=1}^{m} \frac{|T_i|}{|T|} \cdot entropy(T_i)$$

- maximizing information gain is equivalent to minimizing the subtrahend

$$\sum_{i=1}^{m} \frac{|T_i|}{|T|} \cdot entropy(T_i)$$

Split Strategy: Gini-Coefficient 1/2

- **Gini-Coefficient** for a set T of training instances w.r.t. a set of classes $c_1, c_2, \ldots, c_k$ is defined as:

$$gini(T) = \sum_{i=1}^{k} (p_i) \cdot (1 - p_i) = 1 - \sum_{i=1}^{k} p_i^2$$

- measures the expected risk of miss-classification (an element of class i occurs with probability p_i and is classified as not i with probability $1 - p_i$)
  - small gini-coefficient $\Leftrightarrow$ small risk
  - high gini-coefficient $\Leftrightarrow$ high risk

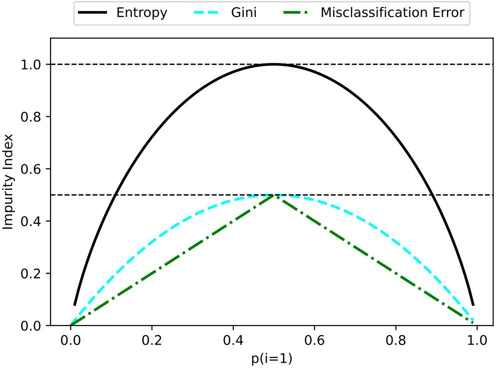
Split Strategy: Gini-Coefficient 2/2

- Let a split A create the partition $T_1, T_2, \ldots, T_m$ of T.
- The **Gini-Coefficient** of split A w.r.t. T is:

$$gini_A(T) = \sum_{i=1}^{m} \frac{|T_i|}{|T|} \cdot gini(T_i)$$

Gini vs. Entropy

- Entropy and Gini-Coefficient are similar
- Gini is computationally less expensive.
- Information gain is biased towards many-valued attributes



Split Strategies: Example – Tennis dataset

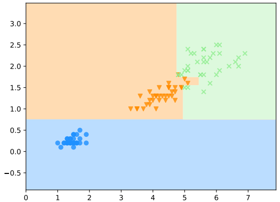
day	outlook	temp	humidity	wind	play
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

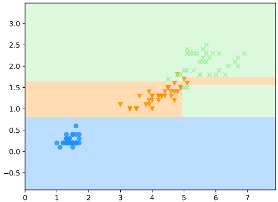
Compare a possible split by humidity to one by wind.

Notebook 04_1_decision_trees_information_gain_example

Stability of a tree

- Decision trees are very sensitive towards small changes in the training data
- Two different ways of sampling training data might create completely different trees.





Outline

Basics

Construction

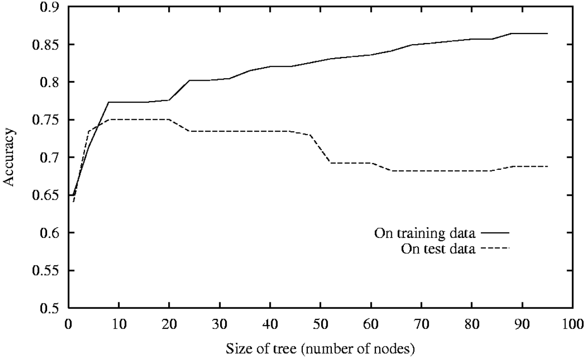
Overfitting

Decision Trees in Python

Problem: Overfitting[2]

Overfitting in the construction of a decision tree occurs, when a smaller tree (less depth) has

- a lower classification quality on the training data
- but a higher classification quality on the test data.



[1]

[1] source https://jmvidal.cse.sc.edu/talks/decisiontrees/allslides.html

[2] More detail on overfitting later

Overfitting in Trees

- By construction, splitting could continue until each leaf has exactly one training instance or only duplicates with the same features.

Overfitting in Trees

- By construction, splitting could continue until each leaf has exactly one training instance or only duplicates with the same features.
- The tree would have learned the training data “by heart”

Overfitting in Trees

- By construction, splitting could continue until each leaf has exactly one training instance or only duplicates with the same features.
- The tree would have learned the training data “by heart”
- Idea 1: Find criteria to stop the construction earlier

Overfitting in Trees

- By construction, splitting could continue until each leaf has exactly one training instance or only duplicates with the same features.
- The tree would have learned the training data “by heart”
- Idea 1: Find criteria to stop the construction earlier
- Idea 2: Prune the tree after it has been constructed

Avoid Overfitting during Construction

- Choice of a *minimum support*: Minimum number of training instances to create further splits ($>> 1$)

Avoid Overfitting during Construction

- Choice of a *minimum support*: Minimum number of training instances to create further splits ($>> 1$)
- Choice of a *minimum information gain*: minimum information gain to allow a split

Avoid Overfitting during Construction

- Choice of a *minimum support*: Minimum number of training instances to create further splits ($>> 1$)
- Choice of a *minimum information gain*: minimum information gain to allow a split
- Choice of a *minimum confidence*: threshold on the share of the majority class in a leaf to stop further splitting – leafs can thus absorb noise or erroneous data.

Avoid Overfitting during Construction

- Choice of a *minimum support*: Minimum number of training instances to create further splits ($>> 1$)
- Choice of a *minimum information gain*: minimum information gain to allow a split
- Choice of a *minimum confidence*: threshold on the share of the majority class in a leaf to stop further splitting – leafs can thus absorb noise or erroneous data.

➔ drawback: influence on overfitting not measured,
hyper-parameter has to be optimized on separate data subset (validation set).

Pruning of Decision Trees

Error-Reduction-Pruning [Mitchell 1997]

- Split training set again into (a smaller) training and a test set.

Pruning of Decision Trees

Error-Reduction-Pruning [Mitchell 1997]

- Split training set again into (a smaller) training and a test set.
- construct the tree on the new training data

Pruning of Decision Trees

Error-Reduction-Pruning [Mitchell 1997]

- Split training set again into (a smaller) training and a test set.
- construct the tree on the new training data
- Prune using the new test set:

Pruning of Decision Trees

Error-Reduction-Pruning [Mitchell 1997]

- Split training set again into (a smaller) training and a test set.
- construct the tree on the new training data
- Prune using the new test set:
  1. determine that subtree, that if cut off, reduces the classification error on the test set the most.

Pruning of Decision Trees

Error-Reduction-Pruning [Mitchell 1997]

- Split training set again into (a smaller) training and a test set.
- construct the tree on the new training data
- Prune using the new test set:
    1. determine that subtree, that if cut off, reduces the classification error on the test set the most.
    2. remove that subtree

Pruning of Decision Trees

Error-Reduction-Pruning [Mitchell 1997]

- Split training set again into (a smaller) training and a test set.
- construct the tree on the new training data
- Prune using the new test set:
  1. determine that subtree, that if cut off, reduces the classification error on the test set the most.
  2. remove that subtree
  3. repeat 1 and 2 until no further such subtree exists

Pruning of Decision Trees

Error-Reduction-Pruning [Mitchell 1997]

- Split training set again into (a smaller) training and a test set.
- construct the tree on the new training data
- Prune using the new test set:
    1. determine that subtree, that if cut off, reduces the classification error on the test set the most.
    2. remove that subtree
    3. repeat 1 and 2 until no further such subtree exists

→ advantage: directly adress overfitting, drawback: need spearate data subset (validation set).

Pruning of Decision Trees

Minimal cost-complexity pruning [Breiman, Friedman, Olshen & Stone 1984]

- use no separate test set, thus applicable for smaller datasets

Pruning of Decision Trees

Minimal cost-complexity pruning [Breiman, Friedman, Olshen & Stone 1984]

- use no separate test set, thus applicable for smaller datasets
- Pruning on the training data, thus classification quality is no sufficient quality measure

Pruning of Decision Trees

Minimal cost-complexity pruning [Breiman, Friedman, Olshen & Stone 1984]

- use no separate test set, thus applicable for smaller datasets
- Pruning on the training data, thus classification quality is no sufficient quality measure
- new quality measure: trade-off (weighted sum) of classification error and tree size – regulated by trade-off parameter α

Pruning of Decision Trees

Minimal cost-complexity pruning [Breiman, Friedman, Olshen & Stone 1984]

- use no separate test set, thus applicable for smaller datasets
- Pruning on the training data, thus classification quality is no sufficient quality measure
- new quality measure: trade-off (weighted sum) of classification error and tree size – regulated by trade-off parameter α

→ drawback: influence on overfitting not measured,
hyper-parameter has to be optimized on separate data subset (validation set).

Outline

Basics

Construction

Overfitting

Decision Trees in Python

Decision Trees in Python

sklearn.tree.DecisionTreeClassifier

- supports only numerical attributes! (encoding of categorical attributes later in this course)
- parameters for stopping criteria and cost complexity pruning
- graphical output via graphviz[3]

```
# Graphviz has to be installed on your os.
# The following module allow
# accessing Graphviz via Python:
pip3 install graphviz
```

Decision Trees in Python

sklearn.tree.DecisionTreeClassifier

- supports only numerical attributes! (encoding of categorical attributes later in this course)
- parameters for stopping criteria and cost complexity pruning
- graphical output via graphviz[3]

```
# Graphviz has to be installed on your os.
# The following module allow
# accessing Graphviz via Python:
pip3 install graphviz
```

Notebook 04_2_plot_tree

Notebook 04_3_decision_tree_digits

Exercises 1–3