Machine Learning

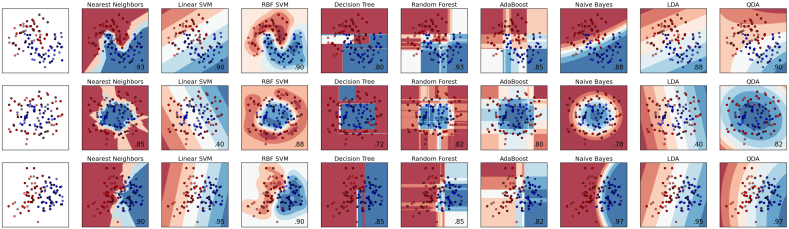
– winter term 2016/17 –

Chapter 02: Decision Trees

Prof. Adrian Ulges
Masters "Computer Science"
DCSM Department
University of Applied Sciences RheinMain

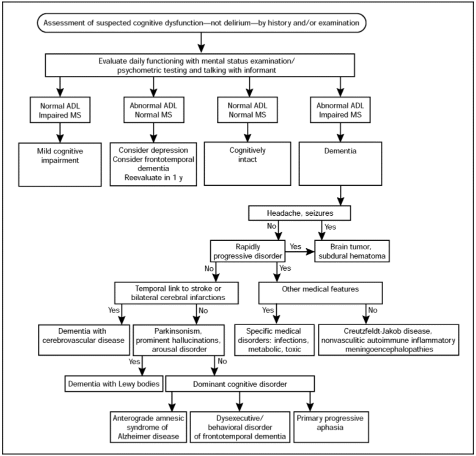
Outline

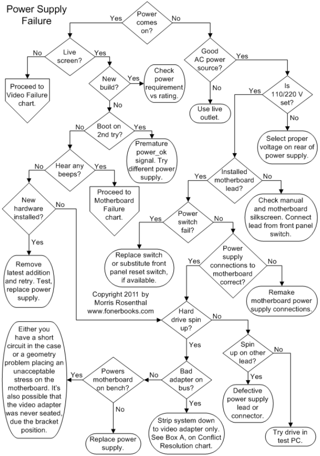
Classifiers image from [3]



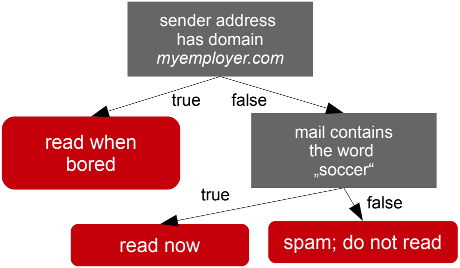
- Many different **models** exist for solving **classification problems**
- We will discuss some of the most common
  - **Decision Trees**
  - Naive Bayes
  - Nearest Neighbor
  - Support Vector Machines
  - Neural Networks

Decision Trees in Expert Systems image from [7]





Decision Trees: Introduction



Decision trees are claimed to be **the most** popular classifier world-wide [8]

Benefits

- **flexibility** (dealing with non-numeric features and regression problems)
- **simplicity** and **speed**
- **transparency** of the classifier's decisions

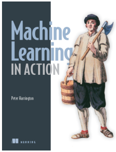
Approach

- Choose a class based on simple **recursive decisions** (or *rules*)

Key Question

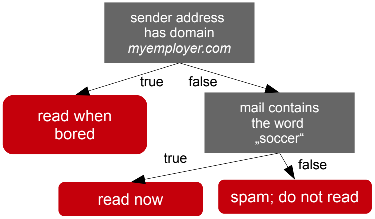
- **Learning**: How do we construct a tree structure / rule set based on a (labeled) training set?

Hello World: Classifying Water Animals [6]



sample	must come to surface?	color	has flippers?	class
1	yes	gray	yes	fish
2	yes	blue	yes	fish
3	yes	blue	no	non-fish
4	no	green	yes	non-fish
5	no	gray	yes	non-fish

Decision Tree Learning: Basics



- **General Approach**: recursive construction of tree using a **greedy strategy**
- Each node in the tree is associated with a **subset** of the training data: the root with the *whole* training set, nodes further down in the tree with increasingly smaller subsets
- For each node N ...
  - pick the **'best' feature** F
  - use F to **split** N's set of samples into **subsets**, each associated with one of N's children
  - continue **recursively**
- Stop once a node contains only samples from one class.

The ID3 Decision Tree

There are different **types of decision trees**, all following the above greedy approach towards learning:

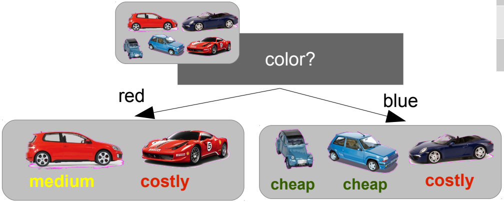
- **ID3**
- **C4.5**
- **CART**

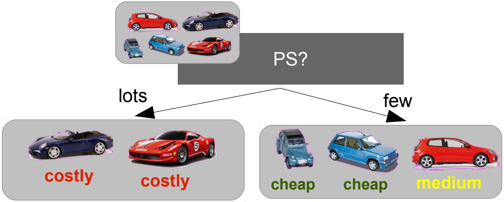
ID3 Decision Trees

- Assumption: Each feature has only a **finite number** of realizations (*example 'color': red, silver, blue*)
- When splitting, we split into *all* possible realizations of a feature (*in the example: three-fold split*)
- As the 'best' feature, we choose the most **informative** one
- **Analogy**: The game *20 Questions* $\rightarrow$ reach an unambiguous answer with as few questions as possible

Which Split is Better?

object	color	PS	class
	blue	few	cheap
	blue	lots	costly
	red	few	medium
	red	lots	costly
	blue	few	cheap





→ **Strategy**: Pick the split that leads to the **purest subnodes**

Outline

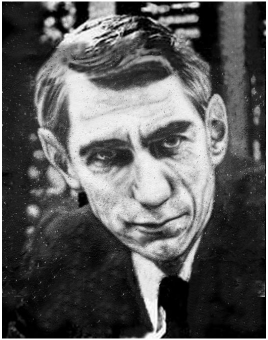
Excursion: Information Theory image from [4]

Are these questions related...?

- How do we measure "uncertainty" / **"randomness"**?
- How dense can **zip** compress English text?
- How do we measure the **similarity** of two histograms?
- How do we measure whether two categorial variables *(e.g., clothing and wheather)* are **related**?

Information Theory

- Claude E. Shannon: "A Mathematical Theory of Communication" (1948)
- Various applications
  - data compression
  - natural language processing
  - statistical inference
  - pattern recognition / machine learning
  - cryptography



Excursion: Information Theory[1]

Binary Codes

- Imagine a language with four letters a,b,c,d.
  A **message** in this language might be: "abaadbaabcabacda"
- Imagine transmitting this message in **bits**. We **encode** each character separately:

character x	a	b	c	d
code $c(x)$	00	01	10	11

- This turns the message into:
  00.01.00.00.11.01.00.00.01.10.00.01.00.10.11.00
- The message is 32 bits long. On average,
  each character requires **2 bits** of coding.

[1] **Very nice read:** Christopher Olah: "Visual Information Theory".
https://colah.github.io/posts/2015-09-Visual-Information/

Prefix Codes

- **Idea: Frequent items should get shorter codes!**

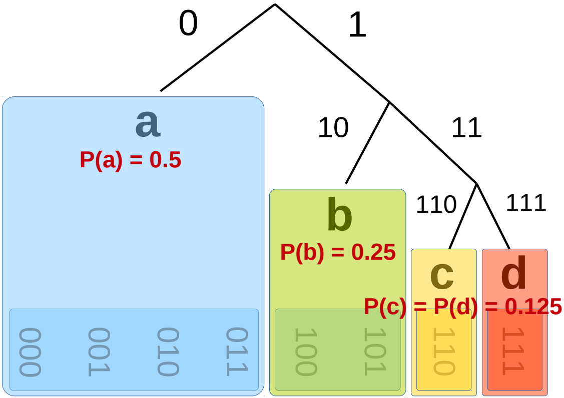
character x	a	b	c	d
probability $P(x)$	1/2	1/4	1/8	1/8
code $c(x)$	0	10	110	111

- This turns the message into: 0.10.0.0.111.10.0.0.10.110.0.10.0.110.111.0
- The message is 28 Bits long. On average, **each character** requires **1.75 bits** of coding.
- This is better! But what's the **best compression** we could achieve this way?

Remark

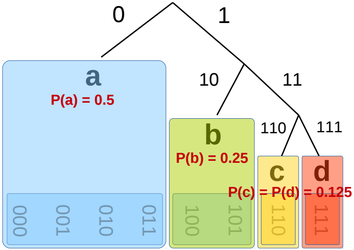
- The **separation** between the single characters is implicit.
- Why is that? Because **no code** is the **prefix** of another code! This is why we call such codes **prefix codes**.

Prefix Codes: Illustration



- We can visualize prefix codes as **trees**!
- Shorter codewords cause **higher "costs"**, because they block larger parts of the **space** of codewords

Codelength vs. Probability



Our **current strategy** for choosing short codes for high-probability characters is based on the following relation between **probability** $P(x)$ and **code length** $L(x)$:

$$P(x) = \frac{1}{2^{L(x)}}$$

$$\Leftrightarrow L(x) = \frac{1}{\log_2 P(x)}$$

$$\Leftrightarrow L(x) = -\log_2 P(x)$$

Remark

- If $P(x)$ is not a power of two, we need to **round up** *(we cannot spend fractions of bits)*

$$L(x) = \lceil -\log_2 P(x) \rceil$$

Optimal Prefix Codes...?

- This means that – using our strategy – we spend the following amount of bits on average per character x:

$$\bar{L} = \sum_x P(x) \cdot L(x) = \sum_x P(x) \cdot \lceil - \log_2 P(x) \rceil$$

- Could we do better with a different strategy? Maybe this one?

character x	a	b	c	d
probability $P(x)$	1/2	1/4	1/8	1/8
code $c(x)$	0	110	10	111

- This would lead to a (slightly worse) average codelength:

$$\frac{1}{2} \cdot 1 + \frac{1}{4} \cdot 3 + \frac{1}{8} \cdot 2 + \frac{1}{8} \cdot 3 = 1.875$$

- It turns out: We cannot do better than our strategy from the last slide *(check* **Huffman Coding** *for more details)*.
- This leads to the central definition in information, **entropy**!

The Entropy

Definition (Entropy)

Let $x_1, ..., x_m$ *be the realizations (characters, events, classes, ...) of a discrete random variable* X *with distribution* $P = (p_1, ..., p_m)$*. Then we call*

$$H(X)\Big(= H(P)\Big) = -\sum_i p_i \cdot log_2(p_i)$$

the **entropy** *of* X *(or* P*).*

Remarks

- The entropy is a **lower bound** on the **average character code length** achieveable by **any prefix code c** (proof: [1])

$$H(X) \leq \sum_i p_i \cdot length(c(x_i)) \qquad \text{for all prefix codes } c$$

- In the above definition, $0 \cdot log_2(0) = 0$ *(i.e., a never-occurring character does not contribute to the overall codelength).*

Entropy and Uncertainty

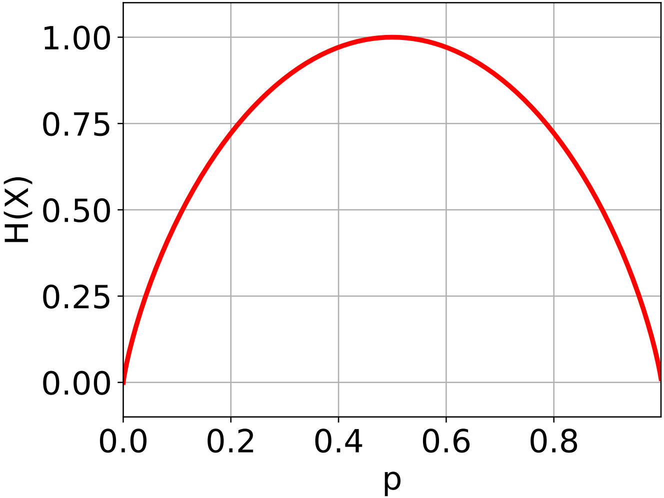
The entropy is a measure of the **randomness** (or **uncertainty**) of a probability distribution.

Example

- Compute the entropy of these distributions!
  - $P = (\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4})$
  - $P = (\frac{1}{4}, \frac{1}{4}, \frac{1}{8}, \frac{3}{8})$
  - $P = (\frac{1}{2}, \frac{1}{4}, \frac{1}{4}, \frac{1}{8}) \rightarrow H(P) = 1.75$
  - $P = (1, 0, 0, 0)$

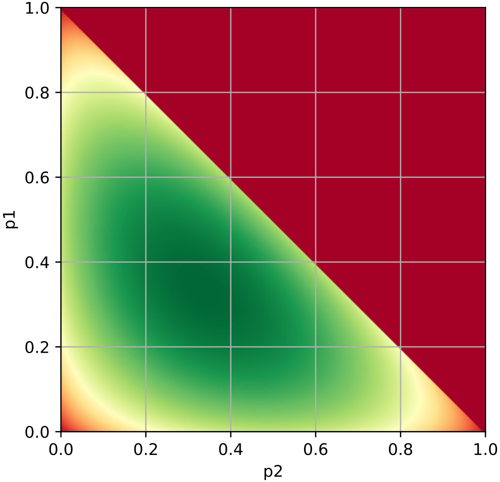
Entropy and Uncertainty

The entropy of a Bernoulli distribution: $P = (p, 1 - p)$



Entropy and Uncertainty

The entropy with 3 realizations: $P = (p_1, p_2, 1 - p_1 - p_2)$



Cross Entropy

- We can also use entropy to measure the **difference** between two distributions!
- Say, we have two languages X_1 and X_2:

character x	a	b	c	d
$P(x)$ / Language X_1	1/2	1/4	1/8	1/8
$P(x)$ / Language X_2	1/8	1/4	1/4	3/8
$-log_2(P(x))$ / Language X_2	3	2	2	1.42

- When encoding messages from Language X_1 using a **code learned from Language $\mathbf{X_2}$**, the average code length per character is (at least):

$$1/2 \cdot 3 + 1/4 \cdot 2 + 1/8 \cdot 2 + 1/8 \cdot 1.42 \approx 2.43$$

- Using the code from Language 2 requires (a lot) more bits compared to Language 1's original code (1.75 bits).
- This is because the **probabilities** are very different!

Cross Entropy

Definition (Cross Entropy)

Let X_1, X_2 be random variables with distributions $P = (p_1, ..., p_m)$ and $Q = (q_1, ..., q_m)$. Then we call

$$H_Q(P) = -\sum_{i=1}^{m} p_i \cdot log_2(q_i)$$

the **cross entropy** *of P and Q.*

Remarks

- The cross entropy is <u>not symmetric</u>: $H_Q(P) \neq H_P(Q)$.
- The cross entropy is always larger than the original entropy: $H_Q(P) \geq H_P(P) = H(P)$ *(i.e., the code from a different language is never better than the original code).*

The Kullback-Leibler Divergence

The cross entropy leads to a **distance measure** between distributions:

Definition (Kullback-Leibler Divergence)

Let X_1, X_2 be random variables with distributions $P = (p_1, ..., p_m)$ and $Q = (q_1, ..., q_m)$. Then we call

$$D_{KL}(P||Q) = H_Q(P) - H(P) = \sum_i p_i \cdot log_2 \frac{p_i}{q_i}$$

the **Kullback-Leibler divergence** *(short:* **KL divergence***) between X_1 and X_2.*

Remarks

- The KL divergence is the **difference in bits** required when encoding characters from P using the code from Q (instead of P).
- The KL Divergence is not symmetric: $H_{X_2}(X_1) \neq H_{X_1}(X_2)$.
- There is a symmetric version, the Jensen-Shannon-Divergence:

$$D_{JS}(P||Q) = \frac{1}{2}\Big(D_{KL}(P||Q) + D_{KL}(Q||P)\Big)$$

Joint Entropy

Finally, we look at the **joint distribution** of random variables:

Definition (Joint Entropy)

Let X and Y be random variables with realizations $x_1, \ldots, x_m$ and $y_1, \ldots, y_n$. Then we call

$$H(X, Y) = -\sum_{x,y} P(x, y) \cdot \log_2(P(x, y))$$

the **joint entropy** *of X and Y.*

Remarks

- This is straightforward: We compute the entropy to the joint probability table of X and Y
- Example: $H(W, C) = -(56\% \cdot log_2(56\%) + \ldots) = 1.59$

weather W / clothing C	t-shirt	coat
sunny	56%	13%
rainy	6%	25%

Mutual Information

Definition (Mutual Information)

Let X and Y be random variables with realizations $x_1, \ldots, x_m$ and $y_1, \ldots, y_n$. Then we call

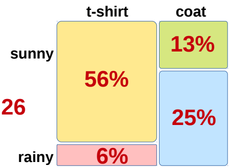
$$I(X,Y) = H(X) + H(Y) - H(X,Y)$$

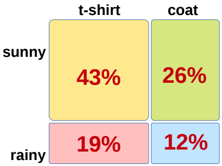
the **mutual information** *between X and Y.*

Remarks

- The mutual information is a measure for the **relatedness** of two variables. Think of it like *correlation* (only, it works for non-numerical variables too)!

H(W,C) = 0.26



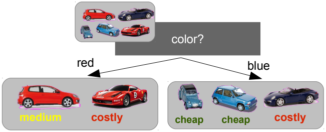


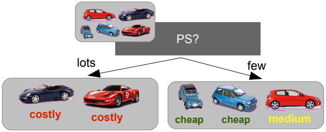
H(W,C) = 0

Outline

Information Theory for Decision Tree Learning

Back to **decision trees** – we compare splits by their **purity**!





- We check the **class distribution** in top node and child nodes
- Which split gives the **strongest reduction in entropy**?
- We measure this reduction by the **information gain**.

Example: Split by Color

- top: $H^{top} = H(p_{cheap}, p_{medium}, p_{costly}) = H(\frac{2}{5}, \frac{1}{5}, \frac{2}{5}) = 1.52$
- left: $H^{left} = H(p_{cheap}, p_{medium}, p_{costly}) = H(0, \frac{1}{2}, \frac{1}{2}) = 1$
- right: $H^{right} = H(p_{cheap}, p_{medium}, p_{costly}) = H(\frac{2}{3}, 0, \frac{1}{3}) = 0.92$
- **information gain**: $H^{top} - \left(\frac{2}{5} \cdot H^{left} + \frac{3}{5} \cdot H^{right}\right) = 0.57$

Definition: Information Gain

Definition (Information Gain)

Let $X = \{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_n, y_n)\}$ be a node's samples (y_i denotes $\mathbf{x}_i$'s class). We split X into subsets $X_1, ..., X_k$ by a feature F. Given a set X', we define its class distribution's entropy as:

$$H(X') = H(p_1, ..., p_C) \text{ with } p_c := \frac{\#\{(\mathbf{x}, y) \in X' \mid y = c\}}{\#X'}$$

Then the **Information Gain** *of feature F is:*

$$Gain(X, F) := H(X) - \sum_{k=1}^{K} \frac{\#X_k}{\#X} \cdot H(X_k)$$

Remarks

- The information gain is always ≥ 0.

Summary: ID3 Learning procedure

- Given: a set of **samples** X, each sample $(\mathbf{x}, y) \in X$ consisting of **features x** and **class label** y
- Given: A set of **features** F, each feature $f \in F$ **mapping** objects to a finite set of values $(f : X \rightarrow \{v_f^1, ..., v_f^{n_f}\})$, for example: $f_{color} : X \rightarrow \{red, silver, blue\}$

```
function build_tree_id3(X,F):
    if all samples in X have the same label y':
        return (y', -, {})            // leaf node: label y', no feature, no children

    if F == {}:                       // no features left to split
        y' := most frequent label in X
        return (y', -, {})

    f' := argmax_{f∈F} Gain(X, f)
    use f' to split X into subsets X_1, ..., X_k
    return ( -, f', { build_tree_id3(X_1, F\{f'}) ,
                      build_tree_id3(X_2, F\{f'}) ,
                      . . .
                      build_tree_id3(X_k, F\{f'}) })
```

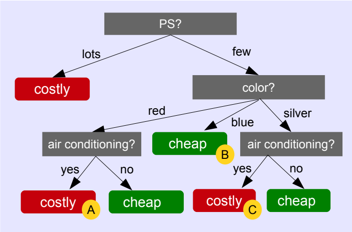
Outline

Decision Trees: Extensions

Later variants of decision trees (here, C4.5 and CART) offer **improvements** and **extensions**

- dealing with **missing** feature values
- dealing with **real-valued** features
- better generalization by **pruning**
- application to **regression** problems
- different **node purity** measures (*Gini impurity*)

Missing Feature Values



- Decision trees can classify test samples with **missing features**!
- **Approach**: **Traverse** *all* children and conduct a **voting** over the resulting labels
- **Example**: Classify a sample with *few PSs*, *no airconditioning*, and **unknown color**
  - unknown color $\rightarrow$ traverse leaves A, B and C
  - 2 votes for costly, 1 for cheap $\rightarrow$ decision for class ‘costly’

Missing Feature Values: Training (C4.5)

Training with Missing Features (ID3)

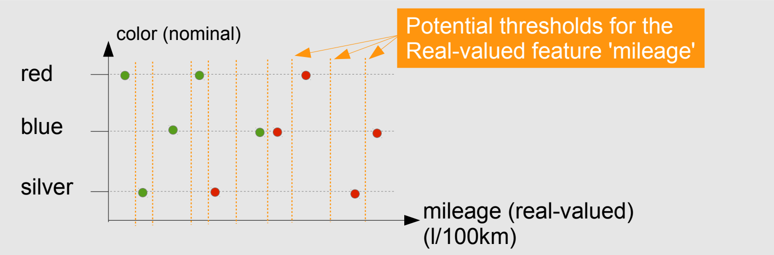
- Decision trees can **train** on samples **missing some features**!
- Samples that miss a feature f are ignored when computing the information gain for f, $Gain(X, f)$
- In ID3, when splitting by f, all samples missing f are dropped.
- **Problem?** $\rightarrow$ Sample size may decrease rapidly when descending into the tree.

Training with Missing Features (C4.5)

- When splitting by f, we distribute samples with missing feature f over the child nodes
- This means: The missing feature is **estimated**
- To do so, different strategies exist
  - ... use the most frequent value in the **class**
  - ... use the most frequent value in the **node**
  - ... distribute samples partly over the child nodes

Real-valued Features (C4.5)

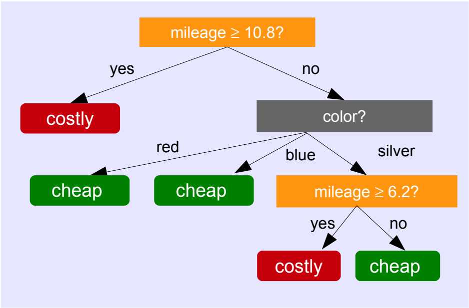
- ID3 only supports features with a **finite number** of realizations. In practice, however, many features are **real-valued**.
- **Approach**: for real-valued features f, choose a **threshold t** and do a **binary split**: $f(x) \geq t$ vs. $f(x) < t$
- Learning gets **more expensive**: For real-valued features, **all potential thresholds** t between *any two values* in X need to be checked

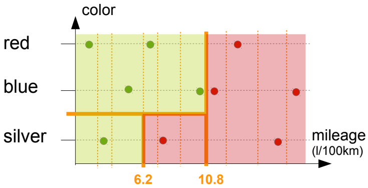


Real-valued Features (cont'd)

Note: With real-valued features, we can use **multiple splits** with the same feature, but *different thresholds!*

Example





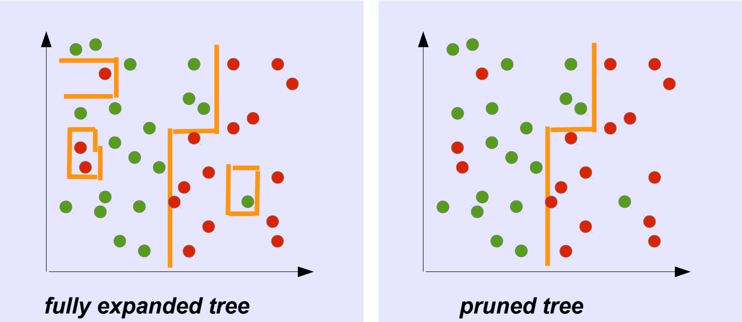
Decision Trees and Overfitting

- By using information gain, we try to achieve **small** (i.e., **simple**) trees
- On the other hand, we split until nodes are **pure** *(which makes the trees large and complex)*
- **Should we** really split until nodes are pure?

Example

Pruning

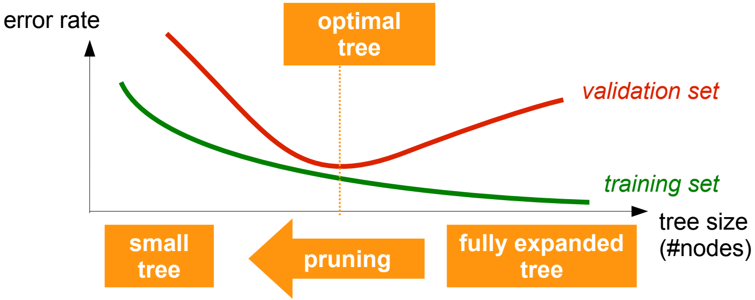
- Fully-expanded trees tend to **overfit**
- Goal: reduce size/complexity by **pruning**
  (which simplifies the decision boundary)
- Pruning means to **remove nodes**, starting at the leaves
- In the resulting *mixed* nodes, we classify by **majority voting**



Pruning 1: Validation

First Strategy: Use a Validation Set

- divide training samples into a training set and a **validation set**
- train a **fully expanded** tree on the training set
- successively remove leave nodes, as long as the error on the **validation set** decreases



Pruning 2: Statistical Tests

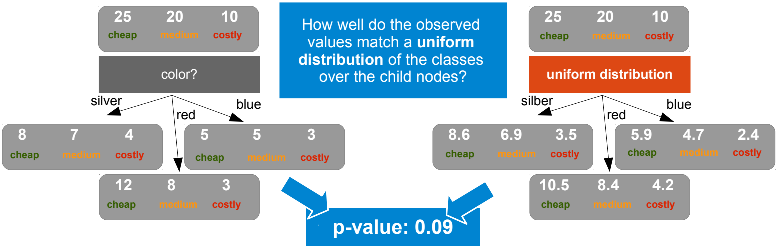
Excursion: χ^2 Testing

Pruning 2: Statistical Tests

Pruning 2: Statistical Tests

- We want to decide whether a split is **useful**
- **Key question**: Are feature and class label **independent**?
- We can check using a χ^2 **independence test**
- The test's **p-value** is the probability of *observing the given distribution*, assuming that feature and class were *independent*
- Choose a **threshold t** and remove a split if $p > t$
- t can be set manually, or learned on a validation set

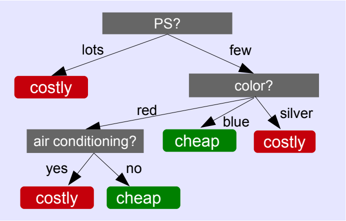
Example



Approach 3: Rule-based “Post Pruning” (C4.5)

Approach

- Transform the tree into a **set of if-then-rules** (each path from **root to leaf** becomes one rule)
- **Remove parts** from each rule’s if-condition and check if accuracy on **validation set** improves
- **Sort rules** by their accuracy and apply them sequentially



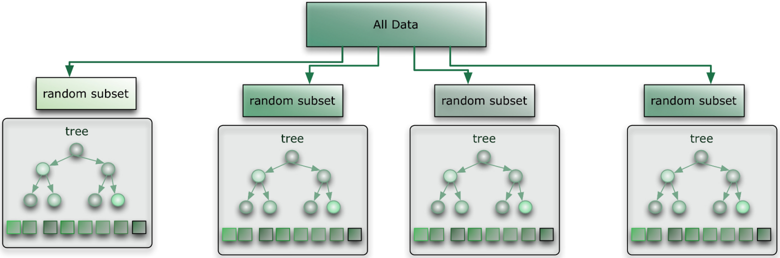

```
1)  ( PS=lots )                            -> costly
2)  ( PS=few ^ color=blue )                -> cheap
3)  ( PS=few ^ color=silver )              -> costly
4)  ( PS=few ^ color=red ^
      airconditioning=yes )                -> costly
5)  ( PS=few ^ color=red ^
      airconditioning=yes )                -> cheap
```

Evaluate rule (3) versus ...

```
3a)  ( PS=few )                            -> costly
3b)  ( color=silver )                      -> costly
```

and keep the best

Approach 4: Random Forests image from [2]



- Use fully expanded trees... but **many**! *(random forests)*
- The construction of the single trees is randomized *(random forests)*
- Test samples are classified with each tree, and a **voting** over all trees is conducted
- Random forests are an **ensemble method**. This means: many simple classifiers (=trees) are **combined** to reach a more accurate decision

Approach 4: Random Forests (cont'd)

What is a **good strategy** to construct the single trees?

- **Goal 1**: The single trees should be as *accurate* as possible
- **Goal 2**: The single trees should be as *independent* as possible

Approaches

- Choose a random feature for each split
- Choose random training data *(bagging)*
- Pre-select a **subset of features** randomly, and pick the best feature from the subset *(random input selection)*

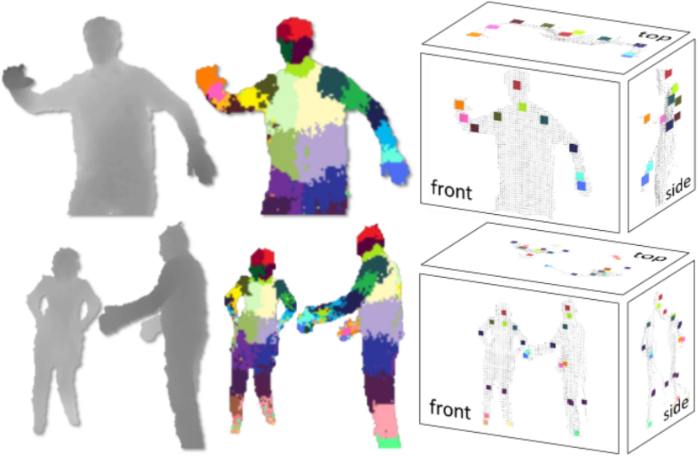
Approach 4: Random Forests (cont'd)

Example Evaluation [5]: Error rates on a variety of standard datasets from the *(UCI Machine Learning Repository)*

Data set	Adaboost	Selection	Forest-RI single input	One tree
Glass	22.0	20.6	21.2	36.9
Breast cancer	3.2	2.9	2.7	6.3
Diabetes	26.6	24.2	24.3	33.1
Sonar	15.6	15.9	18.0	31.7
Vowel	4.1	3.4	3.3	30.4
Ionosphere	6.4	7.1	7.5	12.7
Vehicle	23.2	25.8	26.4	33.1
German credit	23.5	24.4	26.2	33.3
Image	1.6	2.1	2.7	6.4
Ecoli	14.8	12.8	13.0	24.5
Votes	4.8	4.1	4.6	7.4
Liver	30.7	25.1	24.7	40.6
Letters	3.4	3.5	4.7	19.8
Sat-images	8.8	8.6	10.5	17.2
Zip-code	6.2	6.3	7.8	20.6
Waveform	17.8	17.2	17.3	34.0
Twonorm	4.9	3.9	3.9	24.7
Threenorm	18.8	17.5	17.5	38.4
Ringnorm	6.9	4.9	4.9	25.7

Approach 4: Random Forests (cont'd)

Application Example: Kinekt Body Part Recognition[2]



[2]Shotton et al.: Real-Time Human Pose Recognition in Parts from Single Depth Images (Microsoft Research), CVPR 2011.

Decision Trees for Regression

We can also apply **decision trees** for regression
(CART: "Classification And Regression Trees")

Applying a decision tree

- **Classification**: choose a class label per leaf by voting
- **Regression**: choose a value per leaf: The **average** $\bar{y}$

Training

- Classification: Pick feature with **maximum information gain**
- Regression: Pick feature that **minimizes the prediction error**
- A feature splits a node X into subnodes $X_1, ..., X_k$. Within each subnode, we define the average $\bar{y}_k := \frac{1}{\#X_k} \sum_{(x,y) \in X_k} y$

$$f^* := \arg\min_{f \in F} \sum_{k=1}^{K} \sum_{(x,y) \in X_k} \left(y - \bar{y}_k \right)^2$$

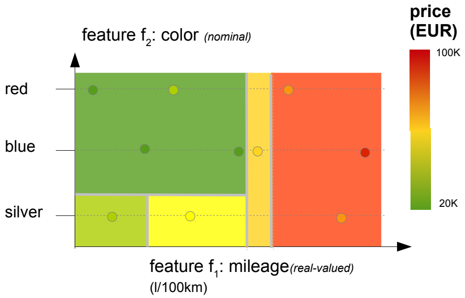
Decision Trees for Regression

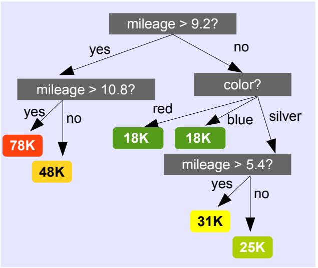
Training (cont'd)

- When do we **stop splitting**? When the error in a node X falls below a certain threshold T:

$$\frac{1}{\#X} \sum_{(x,y)\in X} \left(y - \bar{y}\right)^2 \quad < \quad T$$

Example: Predicting Car Prizes





Decision Trees: Discussion

References I

[1] Michael Langer: Entropy is a lower bound on average code length (lecture slides, Jan 2008).
http://www.cim.mcgill.ca/~langer/423/lecture5.pdf (retrieved: Oct 2016).

[2] RandomForest.
http://randomforest2013.blogspot.de/2013/05/randomforest-definicion-random-forests.html (retrieved: Oct 2016).

[3] Scikit-Learn Landing Page.
http://scikit-learn.org (retrieved: Oct 2016).

[4] USA mathematician and electronic engineer Claude Shannon.
https://commons.wikimedia.org/wiki/File:Claude_Shannon_graffiti.jpg (retrieved: Oct 2016).

[5] L. Breiman.
Random forests.
Mach. Learn., 45(1):5–32, Oct. 2001.

[6] P. Harrington.
Machine Learning in Action.
Manning Publications Co., 2012.

[7] M. Rosenthal.
Computer Repair with Diagnostic Flowcharts: Troubleshooting PC Hardware Problems from Boot Failure to Poor Performance.
Foner Books, 2003.

[8] G. Seni and J. Elder.
Ensemble Methods in Data Mining: Improving Accuracy through Combining Predictions.
Morgan and Claypool, 2010.