# Introduction to Machine Learning
## NPFL 054

http://ufal.mff.cuni.cz/course/npfl054

Barbora Hladká
hladka@ufal.mff.cuni.cz

Martin Holub
holub@ufal.mff.cuni.cz

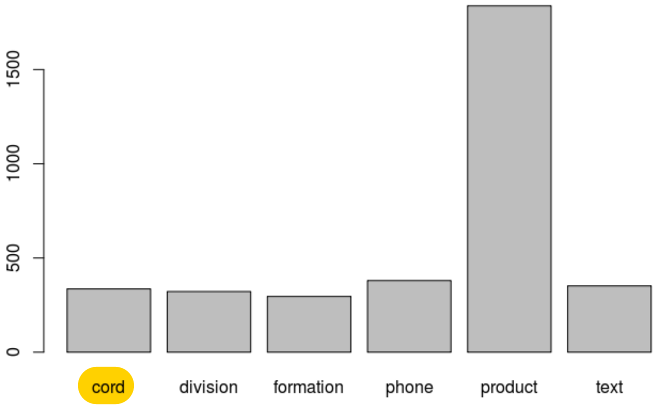
Charles University in Prague,
Faculty of Mathematics and Physics,
Institute of Formal and Applied Linguistics

# Outline

- **Entropy and conditional entropy**
  — definition, calculation, and application for feature selection
- **Decision Trees**
  — building decision trees and using them as prediction function
- **Random Forests**
  — extension of Decision Trees

# WSD task — distribution of target class values

```
> plot(examples$SENSE)
>
```



# Amount of information contained in a value?

How much information do you gain when you observe a random event?
According to the **Information Theory**, **amount of information** contained in an event is given by

$$\boxed{I = \log_2 \frac{1}{p} = -\log_2 p}$$

where $p$ is probability of the event occurred.
Thus, the lower probability, the more information you get when you observe an event (e.g. a feature value). If an event is certain ($p = 100\,\%$), then the amount of information is zero.

# Amount of information in SENSE values

```
### probability distribution of SENSE
> round(table(examples$SENSE)/nrow(examples), 3)

     cord  division formation     phone   product      text 
    0.095     0.091     0.084     0.108     0.522     0.100 
>

### amount of information contained in SENSE values
> round(-log2(table(examples$SENSE)/nrow(examples)), 3)

     cord  division formation     phone   product      text 
    3.391     3.452     3.574     3.213     0.939     3.324 
>
```

**What is the average amount of information that you get when you observe values of the attribute SENSE?**

# Entropy

The average amount of information that you get when you observe random values is

$$\sum_{value} \Pr(value) \cdot \log_2 \frac{1}{\Pr(value)} \quad = \quad -\sum_{value} \Pr(value) \cdot \log_2 \Pr(value)$$

**This is what information theory calls entropy.**

- Entropy of a random variable $X$ is denoted by $\mathrm{H}(X)$
  - or, $\mathrm{H}(p_1, p_2, \ldots, p_n)$ where $\sum_{i=1}^{n} p_i = 1$
- Entropy is a measure of the uncertainty in a random variable
  - or, measure of the uncertainty in a probability distribution
- The unit of entropy is bit; entropy says how many bits *on average* you necessarily need to encode a value of the given random variable

# Properties of entropy

**Normality**

$$H(\frac{1}{2}, \frac{1}{2}) = 1$$

**Continuity**

$$H(p, 1-p) \text{ is a continuous function}$$

**Non negativity and maximality**

$$0 \leq H(p_1, p_2, \ldots, p_n) \leq H(\frac{1}{n}, \frac{1}{n}, \ldots, \frac{1}{n})$$

**Symmetry**

$$H(p_1, p_2, \ldots, p_n) \text{ is a symmetric function of its arguments}$$

**Recursivity**

$$H(p_1, p_2, p_3, \ldots, p_n) = H(p_1 + p_2, p_3, \ldots, p_n) + (p_1 + p_2)H(\frac{p_1}{p_1 + p_2}, \frac{p_2}{p_1 + p_2})$$

# Entropy of SENSE

**Entropy of SENSE is 2.107129 bits.**

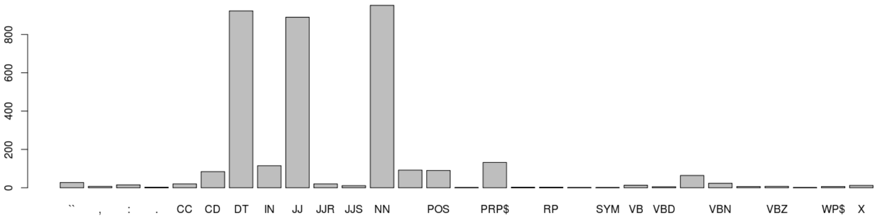
```
### probability distribution of SENSE
> p.sense <- table(examples$SENSE)/nrow(examples)
>
### entropy of SENSE
> H.sense <-  - sum( p.sense * log2(p.sense) )
> H.sense
[1] 2.107129
```

**The maximum entropy value would be** $\log_2(6) = 2.584963$
**if and only if the distribution of the 6 senses was uniform.**

```
> p.uniform <- rep(1/6, 6)
> p.uniform
[1] 0.1666667 0.1666667 0.1666667 0.1666667 0.1666667 0.1666667
>
### entropy of uniformly distributed 6 senses
> - sum( p.uniform * log2(p.uniform) )
[1] 2.584963
```

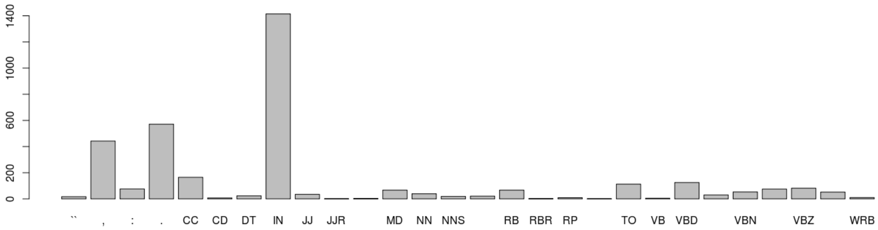
# Distribution of feature values – A16

```
> levels(examples$A16)
 [1] "``"     ","      ":"      "."      "CC"     "CD"     "DT"     "IN"     "JJ"
[10] "JJR"    "JJS"    "NN"     "NNS"    "POS"    "PRP"    "PRP$"   "RB"     "RP"
[19] "-RRB-"  "SYM"    "VB"     "VBD"    "VBG"    "VBN"    "VBP"    "VBZ"    "WDT"
[28] "WP$"    "X"
> plot(examples$A16)
>
```



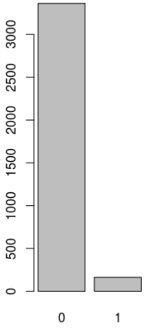
# Distribution of feature values – A17

```
> levels(examples$A17)
 [1] "``"     ","      ":"      "."      "CC"     "CD"     "DT"     "IN"     "JJ"
[10] "JJR"    "-LRB-" "MD"     "NN"     "NNS"    "PRP"    "RB"     "RBR"    "RP"
[19] "-RRB-" "TO"     "VB"     "VBD"    "VBG"    "VBN"    "VBP"    "VBZ"    "WDT"
[28] "WRB"
> plot(examples$A17)
>
```



# Distribution of feature values – A4

```
> levels(examples$A4)
[1] "0" "1"
>
```



# Entropy of features

**Entropy of A16 is** 2.78 **bits.**

```
> p.A16 <- table(examples$A16)/nrow(examples)
> H.A16 <-  - sum( p.A16 * log2(p.A16) )
> H.A16
[1] 2.777606
```

**Entropy of A17 is** 3.09 **bits.**

```
> p.A17 <- table(examples$A17)/nrow(examples)
> H.A17 <-  - sum( p.A17 * log2(p.A17) )
> H.A17
[1] 3.093003
```

**Entropy of A4 is** 0.27 **bits.**

```
> p.A4 <- table(examples$A4)/nrow(examples)
> H.A4 <-  - sum( p.A4 * log2(p.A4) )
> H.A4
[1] 0.270267
```

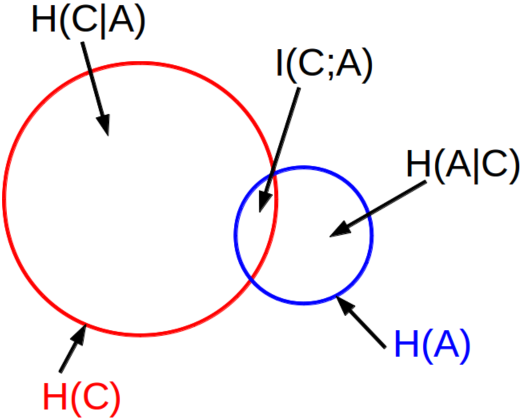
# Conditional entropy H(C | A)

**How much does target class entropy decrease if we have the knowledge of a feature?**

**The answer is conditional entropy:**

$$\boxed{\mathrm{H}(C \mid A) = -\sum_{y \in C, x \in A} \Pr(y, x) \cdot \log_2 \Pr(y \mid x)}$$

# Conditional entropy and mutual information




**WARNING**
**There are NO SETS in this picture! Entropy is a quantity, only a number!**

# Conditional entropy and mutual information

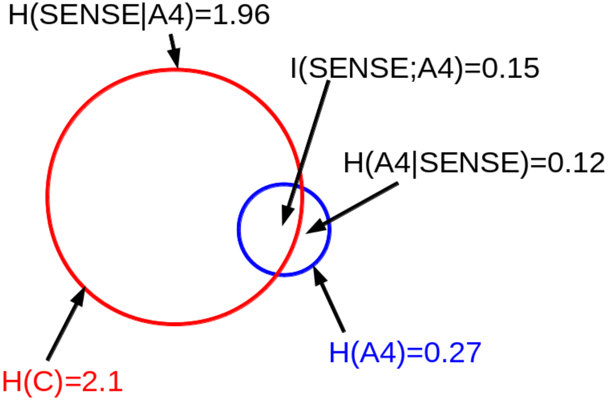
**Mutual information** **measures the amount of information that can be obtained about one random variable by observing another.**

Mutual information is a symmetrical quantity.

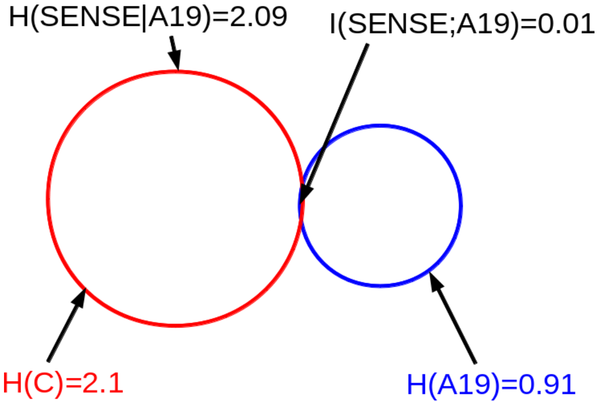
$$\boxed{\mathrm{H}(C) - \mathrm{H}(C \mid A) \quad = \quad \mathrm{I}(C; A) \quad = \quad \mathrm{H}(A) - \mathrm{H}(A \mid C)}$$

Another name for mutual information is **information gain**.

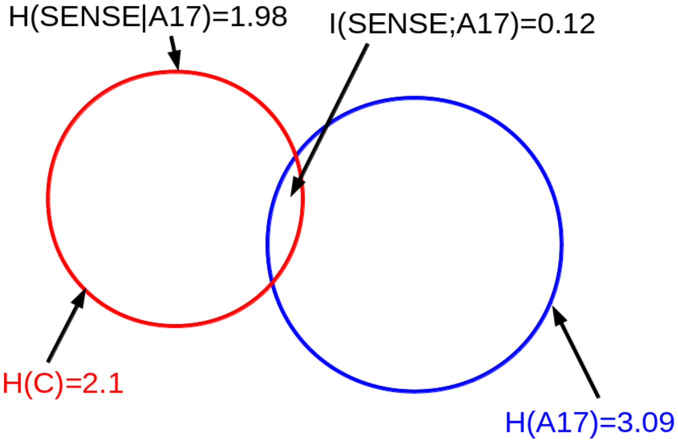
# Conditional entropy – feature A4



# Conditional entropy – feature A19



# Conditional entropy – feature A17



# User-defined functions in R

**Structure of a user-defined function**

```
myfunction <- function(arg1, arg2, ... ){
   ... statements ...
   return(object)
}
```

Objects in a function are local to the function.

# User-defined functions in R

### Structure of a user-defined function

```
myfunction <- function(arg1, arg2, ... ){
   ... statements ...
   return(object)
}
```

Objects in a function are local to the function.

**Example** – a function to calculate entropy

```
> entropy <- function(x){
+   p <- table(x) / NROW(x)
+   return( -sum(p * log2(p)) )
+ }
>

# invoking the function
> entropy(examples$SENSE)
[1] 2.107129
```

# Conditional entropy and feature selection

## Summary

- **Information theory provides a measure** for comparing how features contribute to the knowledge about target class.
- The lower conditional entropy $\mathrm{H}(C \mid A)$, the better chance that $A$ is a useful feature.
- However, since features typically interact, conditional entropy $\mathrm{H}(C \mid A)$ should NOT be the only criterion when you do feature selection. You need experiments to see if a feature with high information gain really helps.

**Note**

Also, decision tree learning algorithm makes use of entropy when it computes purity of training subsets.

# Homework

- Write your own function for computing conditional entropy in R.
  New function `entropy.cond(x,y)` will take two factors of the same length and will compute $\mathrm{H}(x \mid y)$.

  Example use: `entropy.cond(examples$SENSE, examples$A4)`

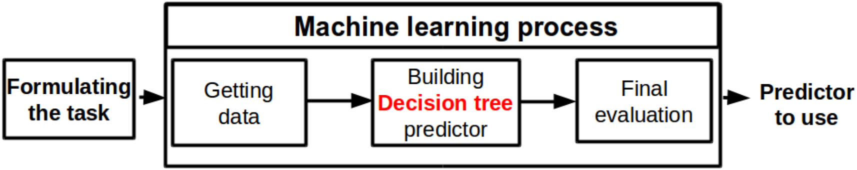
# Entropy – Summary of Examination Requirements

**You should understand and be able to explain and practically use**

- entropy
  - motivation
  - definition
  - main properties
  - calculation in R
- conditional entropy
  - definition and meaning
  - relation to mutual information
  - calculation in R
  - information gain – application in feature selection

# Decision Tree — a learning method

**Decision Tree is a learning method suitable for both classification and regression tasks**



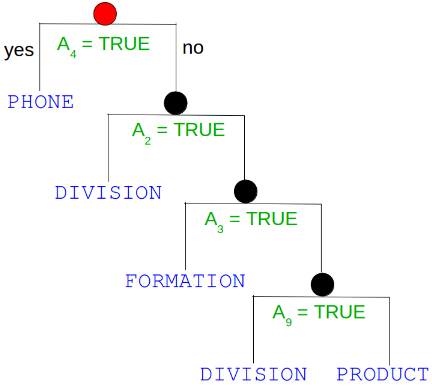

**Example classification task: WSD**

see the NPFL054 web page → Materials → wsd-attributes.pdf

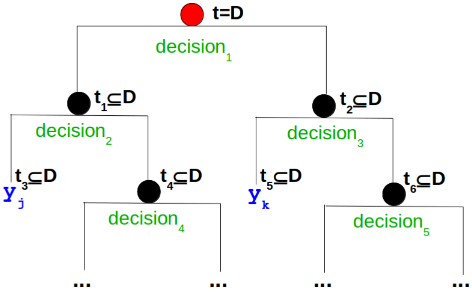
# Decision tree structure

A **decision tree** $T = (V, E)$ is a rooted tree where $V$ is composed of internal **decision nodes** and terminal **leaf nodes**.

- Nodes
  - Root node
  - Internal nodes
  - Leaf nodes with `TARGET OUTPUT VALUES`
- Decisions



**Decision tree learning** corresponds to building a decision tree $T_D = (V, E)$ based on a training data set $D = \{\langle \mathbf{x}, y \rangle : \mathbf{x} \in X, y \in Y\}$.
When building a tree, each node is associated with a set $t$, $t \subseteq D$. The root node is associated with $t = D$.
Each leaf node is designated by an output value.

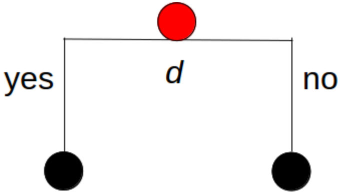


# Building a decision tree from training data

**A very basic idea**: Assume binary decisions

- **Step 1** Create a root node.

- **Step 2** Select decision *d* and add child nodes to an existing node.



# Building a decision tree from training data

**Example**

Associate the root node with the training set *t*.

**Example**

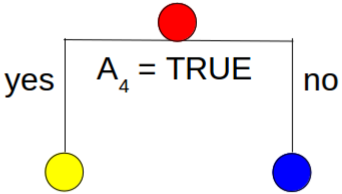
1. Assume decision
   `if` $\mathrm{A}_4 = \mathit{TRUE}$.
2. Split the training set *t* according to this decision into two subsets – "yellow" and "blue".

*t*

| SENSE | ... | A4 | ... |
|---|---|---|---|
| FORMATION | | TRUE | |
| FORMATION | | FALSE | |
| PHONE | | TRUE | |
| CORD | | TRUE | |
| DIVISION | | FALSE | |
| ... | ... | ... | |

# Building a decision tree from training data

3. Add two child nodes, "yellow" and "blue", to the root. Associate each of them with the corresponding subset $t_L$, $t_R$, resp.




$t_L$

| SENSE | ... | A4 | ... |
|---|---|---|---|
| FORMATION | | TRUE | |
| CORD | | TRUE | |
| PHONE | | TRUE | |
| ... | ... | ... | |

$t_R$

| SENSE | ... | A4 | ... |
|---|---|---|---|
| FORMATION | | FALSE | |
| DIVISION | | FALSE | |
| ... | ... | ... | |

# Building a decision tree from training data

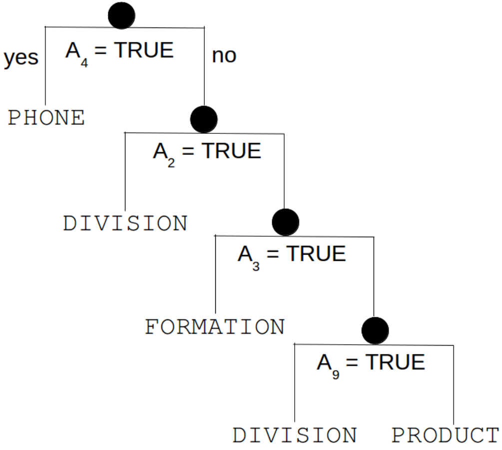
- **Step 4** Repeat recursively steps (2) and (3) for both child nodes and their associated training subsets.
- **Step 5** Stop recursion for a node if a stopping criterion is fulfilled. Create a leaf node with an output value.

# Prediction on test data

Once the decision tree predictor is built, an unseen instance is predicted by starting at the root node and moving down the tree branch corresponding to the feature values asked in decisions.

# Prediction on test data

**Decision tree predictor for the WSD-*line* task**



# Prediction on test data

**Decision tree predictor for the WSD-*line* task**
Assign the correct sense of *line* in the sentence "Draw a line between the points P and Q."
**True prediction**: `DIVISION`

# Prediction on test data

**Decision tree predictor for the WSD-*line* task**

First, get twenty feature values from the sentence

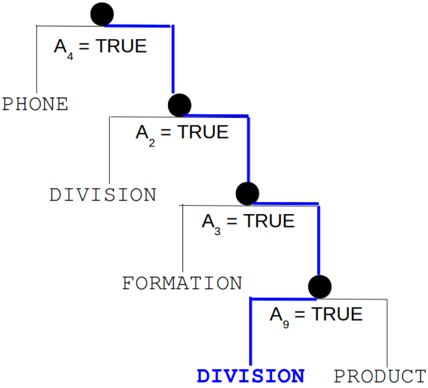
| $A_1$ | $A_2$ | $A_3$ | $A_4$ | $A_5$ | $A_6$ | $A_7$ | $A_8$ | $A_9$ | $A_{10}$ | $A_{11}$ |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |

| $A_{12}$ | $A_{13}$ | $A_{14}$ | $A_{15}$ | $A_{16}$ | $A_{17}$ | $A_{18}$ | $A_{19}$ | $A_{20}$ |
|---|---|---|---|---|---|---|---|---|
| a | draw | X | between | DT | IN | DT | line | dobj |

# Prediction on test data

**Decision tree predictor for the WSD-*line* task**

Second, get the classification of the instance using the decision tree



# Prediction on test data

**Decision tree predictor for the WSD-*line* task**
Assign the correct sense of *line* in the sentence "Draw a line that passes through the points P and Q."
**True prediction**: `DIVISION`

# Prediction on test data

**Decision tree predictor for the WSD-*line* task**

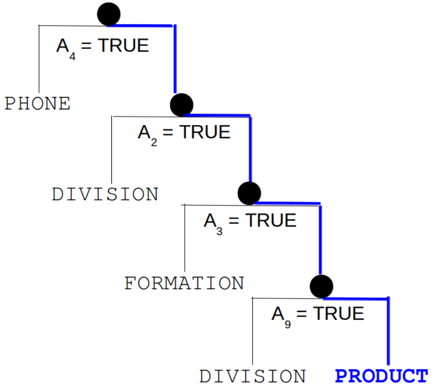
First, get twenty feature values from the sentence

| $A_1$ | $A_2$ | $A_3$ | $A_4$ | $A_5$ | $A_6$ | $A_7$ | $A_8$ | $A_9$ | $A_{10}$ | $A_{11}$ |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| $A_{12}$ | $A_{13}$ | $A_{14}$ | $A_{15}$ | $A_{16}$ | $A_{17}$ | $A_{18}$ | $A_{19}$ | $A_{20}$ |
|---|---|---|---|---|---|---|---|---|
| a | draw | X | that | DT | WDT | VB | line | dobj |

# Prediction on test data

**Decision tree predictor for the WSD-*line* task**
Second, get the classification of the instance using the decision tree



# Decision trees

## Classification trees

- $Y$ is a categorical output feature

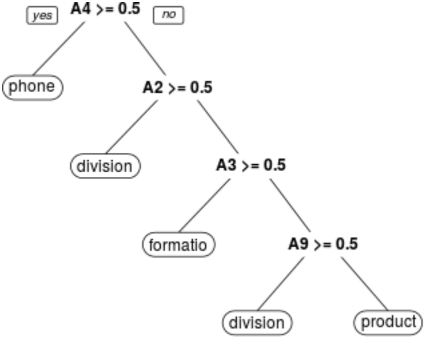



**Figure :** Tree for predicting the sense of *line* based on binary features.

## Regression trees

- $Y$ is a numerical output feature

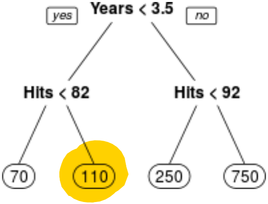
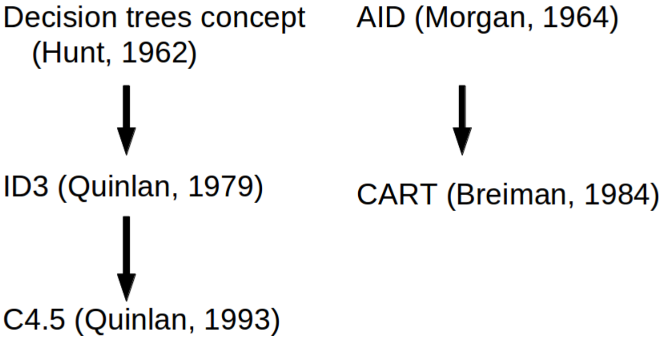



**Figure :** Tree for predicting the salary of a baseball player based on the number of years that he has played in the major leagues (`Year`) and the number of hits that he made in the previous year (`Hits`). See the `ISLR Hitters` data set.

# Historical excursion




Note

- Automatic Interaction Detection(AID)
- ClAssification and Regression trees(CART)

# Building a decision tree from training data

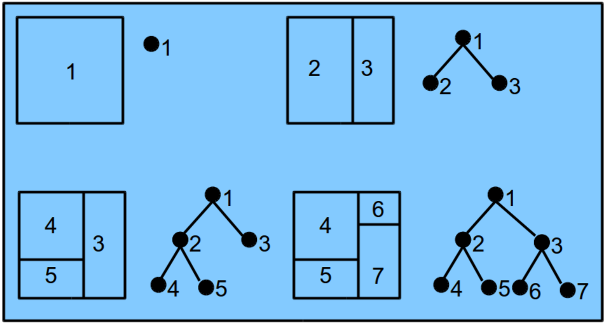
1. Tree growing
2. Tree pruning

**Basic idea:** First, grow a large tree that fits the training data. Second, prune this tree to avoid overfitting.

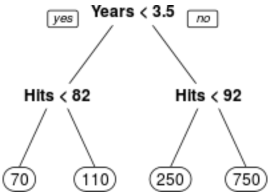
# Building a decision tree from training data

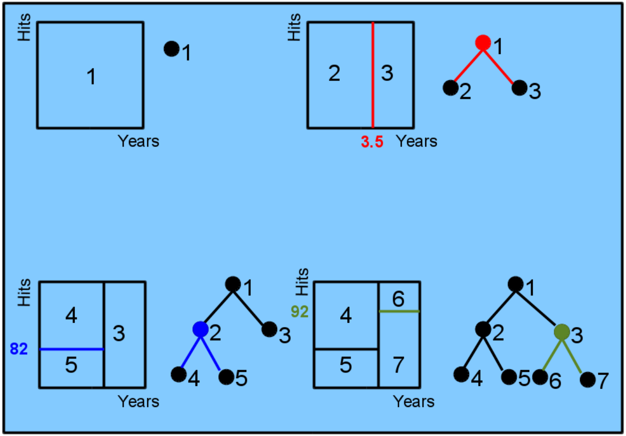
1. Tree growing
2. Tree pruning

The growing process is based on subdividing the feature space recursively into non-overlapping regions.



# Building a decision tree from training data



# Classification and Regression trees

Each terminal node in the decision tree is associated with one of the regions in the feature space. Then

**Classification trees**

- **output value**: the most common class in the data associated with the terminal node

**Regression trees**

- **output value**: the mean output value of the training instances associated with the terminal node

**Notation**

- $Attr = \{A_1, A_2, ..., A_m\}$,
- $Y = \{y_1, y_2, \ldots, y_k\}$
- $Values(A_i)$ is a set of all possible values for feature $A_i$.
- $D_{i,v} = \{\langle \mathbf{x}, y \rangle \in D | x_i = v\}$.

| ... | ... | $A_i$ | ... |
|---|---|---|---|
| ... | ... | v | ... |
| ... | ... | ... | ... |
| ... | ... | ... | ... |
| ... | ... | v | ... |
| ... | ... | v | ... |
| ... | ... | ... | ... |

# Building a classification tree from training data

**We work with decisions on the value of only a single feature**

- For each categorical feature $A_j$ having values $Values(A_j) = \{b_1, b_2, ..., b_L\}$

$$\boxed{\text{is } x_j = b_i\text{? as } i = 1, ..., L}$$

- For each categorical feature $A_j$

$$\boxed{\text{is } x_j \in \text{a subset} \in 2^{Values(A_j)}\text{?}}$$

- For each numerical feature $A_j$

$$\boxed{\text{is } x_j \leq k\text{?}, \ k \in (-\infty, +\infty)}$$

# Building a classification tree from training data

**Which decision is the best?**

- Focus on a distribution of target class values in associated subsets of training examples.
- Then select the decision that splits training data into subsets as pure as possible.

# Building a classification tree from training data

**Which decision is the best?**

We say a data set is **pure** (or **homogenous**) if it contains only a single class. If a data set contains several classes, then the data set is **impure** (or **heterogenous**).

| $\oplus$: 5, $\ominus$: 5 | | $\oplus$: 9, $\ominus$: 1 |
|---|---|---|
| heterogenous<br>high degree of impurity | | almost homogenous<br>low degree of impurity |

# Building a classification tree from training data

**Which decision is the best?**

1. **Define** a candidate set $S$ of splits at each node using possible decisions. $s \in S$ splits $t$ into $L$ subsets $t_1, t_2, \dots t_L$.
2. **Define** the node proportions $p(y_j|t)$, $j = 1, \dots, k$, to be the proportion of instances $\langle \mathbf{x}, y_j \rangle$ in $t$.
3. **Define** an **impurity measure** $i(t)$, i.e. **splitting criterion**, as a nonnegative function $\Phi$ of the $p(y_1|t), p(y_2|t), \dots, p(y_k|t)$,

$$i(t) = \Phi(p(y_1|t), p(y_2|t), \dots, p(y_k|t)), \tag{1}$$

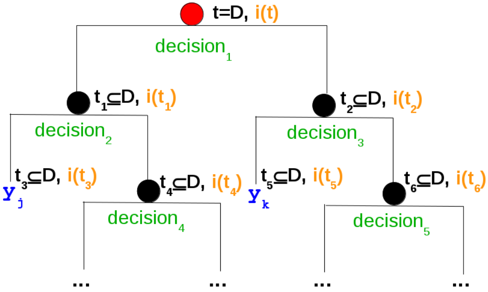
such that

- $\Phi(\frac{1}{k}, \frac{1}{k}, ..., \frac{1}{k}) = max$, i.e. the node impurity is largest when all examples are equally mixed together in it.
- $\Phi(1, 0, ..., 0) = 0, \Phi(0, 1, ..., 0) = 0, ..., \Phi(0, 0, ..., 1) = 0$, i.e. the node impurity is smallest when the node contains instances of only one class

# Building a classification tree from training data

**Which decision is the best?**

4. **Define** the **goodness of split** $s$ to be the decrease in impurity $\Delta i(s,t) = i(t) - \sum_{l=1}^{L} p_l * i(t_l)$, where $p_l$ is the proportion of instances in $t$ that go to $t_l$.
5. **Find** split $s^*$ with the largest decrease in impurity: $\Delta i(s^*, t) = max_{s \in S} \Delta i(s,t)$.
6. **Use** splitting criterion $i(t)$ to compute $\Delta i(s,t)$ and get $s^*$.



# Building a classification tree from training data

**Which decision is the best?**

**Splitting criteria** – examples that are really used

- Misclassification Error – $i(t)_{ME}$
- Information Gain – $i(t)_{IG}$
- Gini Index – $i(t)_{GI}$

# Building a classification tree from training data

**Which decision is the best?**
**Splitting criteria**

$$i(t)_{ME} = 1 - max_{j=1,\dots,k} p(y_j|t) \quad (2)$$

| | $\oplus$: 0, $\ominus$: 6 | $\oplus$: 1, $\ominus$: 5 | $\oplus$: 2, $\ominus$: 4 | $\oplus$: 3, $\ominus$: 3 |
|---|---|---|---|---|
| ME | $1 - \frac{6}{6} = 0$ | $1 - \frac{5}{6} = 0.17$ | $1 - \frac{4}{6} = 0.33$ | $1 - \frac{3}{6} = 0.5$ |

# Building a classification tree from training data

**Which decision is the best?**
**Splitting criteria**

$$i(t)_{IG} = -\sum_{j=1}^{k} p(y_j|t) * \log p(y_j|t). \tag{3}$$

Recall the notion of entropy $H(t)$, $i(t)_{IG} = H(t)$.

$$Gain(s,t) = \Delta i(s,t)_{IG} \tag{4}$$

# Building a classification tree from training data

**Which decision is the best?**
**Splitting criteria**

$$i(t)_{GI} = 1 - \sum_{j=1}^{k} p^2(y_j|t) = \sum_{j=1}^{k} p(y_j|t)(1 - p(y_j|t)). \quad (5)$$

# Building a classification tree from training data

**Which decision is the best?**
**Splitting criteria**

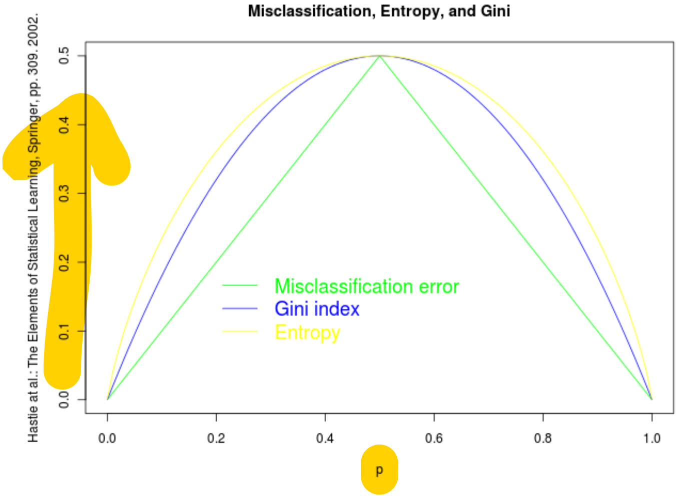
| | $\oplus$: 0<br>$\ominus$: 6 | $\oplus$: 1<br>$\ominus$: 5 | $\oplus$: 2<br>$\oplus$: 4 | $\oplus$: 3<br>$\oplus$: 3 |
|---|---|---|---|---|
| Gini | 0 | 0.278 | 0.444 | 0.5 |
| Entropy | 0 | 0.65 | 0.92 | 1.0 |
| ME | 0 | 0.17 | 0.333 | 0.5 |

For two classes ($k = 2$), if $p$ is the proportion of the class "1", the measures are:

- Misclassification error: $1 - max(p, 1 - p)$
- Entropy: $-p * \log p - (1 - p) * \log(1 - p)$
- Gini: $2p * (1 - p)$

# Building a classification tree from training data

**Which decision is the best?**
**Splitting criteria**



# Classification and Regression trees

Each terminal node in the decision tree is associated with one of the regions in the feature space. Then

**Classification trees**

- Output value: the most common class in the data associated with the terminal node
- A criterion for making splits, e.g.
  - Misclassification error
  - Information gain
  - Gini index

**Regression trees**

- Output value: the mean output value of the training instances associated with the terminal node

# Building a REGRESSION tree from training data

**Notation**

- $Attr = \{A_1, A_2, ..., A_m\}$
- $Y = \mathcal{R}$
- $Values(A_i)$ is a set of all possible values for feature $A_i$

# Building a regression tree from training data

**Again, we work with decisions on the value of only a single feature**

**Which decision is the best?**

**Splitting criterion** – usually used

- Squared Error – $i(t)_{SE}$

$$i(t)_{SE} = \frac{1}{|t|} \sum_{\mathbf{x}_i \in t} (y_i - y^t)^2,$$

where $y^t = \frac{1}{|t|} \sum_{\mathbf{x}_i \in t} y_i$.

# Classification and Regression trees

Each terminal node in the decision tree is associated with one of the regions in the feature space. Then

**Classification trees**

- Output value: the most common class in the data associated with the terminal node
- A criterion for making splits, e.g.
    - Misclassification error
    - Information gain
    - Gini index

**Regression trees**

- Output value: the mean output value of the training instances associated with the terminal node
- A criterion for making splits, e.g. Squared error

# Building decision tree from training data

The recursive binary splitting is stopped when a stopping criterion is fulfilled. Then a leaf node is created with an output value.

**Stopping criteria**, e.g.

- the leaf node is associated with less than five training instances
- the maximum tree depth has been reached
- the best splitting criteria is not greater than a certain threshold

# Decision tree learning algorithms — ID3

As a splitting criterion, ID3 algorithm uses information gain.
ID3 algorithm is nicely described on the Wikiedia.

**Main idea**

- Calculate the entropy of every attribute using the data set S
- Split the set S into subsets using the attribute for which entropy is minimum (or, equivalently, information gain is maximum)
- Make a decision tree node containing that attribute
- Recurse on subsets using remaining attributes

For more details see `https://en.wikipedia.org/wiki/ID3_algorithm`

# Decision tree learning algorithms — ID3

**ID3 algorithm — summary**

- ID3 is a recursive partitioning algorithm (divide & conquer), performs top-down tree construction.
- ID3 maintains only a single current hypothesis. So ID3 for example is not able to determine any other decision trees consistent with training data.
- ID3 does not employ backtracking.

# Decision tree learning algorithms — ID3 and C4.5

**ID3 $\longrightarrow$ C4.5**

ID3 is originally designed with two restrictions:

1. classification task
2. categorical features used to train a decision tree $\rightarrow$ Let's extend ID3 for the continuous-valued features

**C4.5 algorithm: Incorporating continuous-valued features**
For a continuous-valued feature $A$, define a boolean-valued feature $A_c$ so that if $A(\mathbf{x}) \leq c$ then $A_c(\mathbf{x}) = 1$ else $A_c(\mathbf{x}) = 0$.

**C4.5 algorithm: Incorporating continuous-valued features**

How to select the best value for the threshold $c$?
**Example** Choose such $c$ that produces the greatest information gain.

| *Temperature* | 20 | 22 | 24 | 26 | 28 | 30 |
|---|---|---|---|---|---|---|
| *EnjoySport* | No | No | Yes | Yes | Yes | No |

$c_1 = \frac{22+24}{2}$, $c_2 = \frac{28+30}{2}$
$Gain(D, Temperature_{\geq c_1}) = ?$, $Gain(D, Temperature_{\geq c_2}) = ?$

# Decision tree learning algorithms — ID3 and C4.5

**C4.5 algorithm: Handling training examples with missing feature values**

Consider the situation in which $Gain(t, A)$ is to be calculated at node associated with a training data set $t$ in the decision tree. Suppose that $\langle \mathbf{x}, y \rangle$ is one of the training examples in $t$ and that the value $A(\mathbf{x})$ is unknown.

**Possible solutions**

- Assign the value that is most common among training instances associated with the node.
- Alternatively, assign the most common value among instances associated with the node $t$ having the classification $y$.

# Building a decision tree from training data

1. Tree growing $\surd$
2. Tree pruning

**Basic idea:** First, grow a large tree that fits the training data. Second, prune this tree to avoid overfitting.

# Building a decision tree from training data

**Overfitting** can be avoided by

- applying a stopping criterion that prevents some sets of training instances from being subdivided,
- removing some of the structure of the decision tree after it has been produced.

# Building a decision tree from training data

**Overfitting**

Preferred strategy: Grow a large tree $T_0$, stop the splitting process when only some minimum node size (say 5) is reached. Then prune $T_0$ using some pruning criteria.

# Decision trees — implementation in R

There are two widely used packages in R

- `rpart`
- `tree`

The algorithms used are very similar.

**References**


- An Introduction to Recursive Partitioning Using the RPART Routines
  by Terry M. Therneau, Elizabeth J. Atkinson, and Mayo Foundation
  (available online)
- *An Introduction to Statistical Learning with Application in R*
  Chapters 8.1, 8.3.1, and 8.3.2
  by Gareth James, Daniela Witten, Trevor Hastie and Rob Tibshirani
  (available online)
- R packages documentation — `rpart`, `tree`
  (available online)

# Decision Trees – weak spots

- **data splitting**
  — deeper nodes can learn only from small data portions

- **sensitivity to training data set (unstable algorithm)**
  — learning algorithm is called unstable if small changes in the training set cause large differences in generated models

**General scheme of resampling methods**

- Distribute the training data into $K$ portions
- Run the learning process to get $K$ different models
- Collect the output of the $K$ models use a combining function to get a final output value

# Bootstrapping principle

- New data sets $Data_1, \ldots, Data_K$ are drawn from *Data* with replacement, each of the same size as the original *Data*, i.e. $n$.
- In the $i$-th step of the iteration, $Data_i$ is used as a training set, while the examples $\{\mathbf{x} \mid \mathbf{x} \in Data \wedge \mathbf{x} \notin Data_i\}$ form the test set.

# Bootstrapping principle

- New data sets $Data_1, \ldots, Data_K$ are drawn from *Data* with replacement, each of the same size as the original *Data*, i.e. $n$.
- In the $i$-th step of the iteration, $Data_i$ is used as a training set, while the examples $\{\mathbf{x} \mid \mathbf{x} \in Data \wedge \mathbf{x} \notin Data_i\}$ form the test set.
- The probability that we pick an instance is $1/n$, and the probability that we do not pick an instance is $1 - 1/n$. The probability that we do not pick it after $n$ draws is $(1 - 1/n)^n \approx e^{-1} \doteq 0.368$.
- It means that for training the system will not use 36.8 % of the data, and the error estimate will be pessimistic. So the solution is to repeat the process many times.

# Random Forests

- an ensemble method based on decision trees and bagging
- builds a number of random decision trees and then uses voting
- introduced by L. Breiman (2001), then developed by L. Breiman and A. Cutler
- very good (state-of-the-art) prediction performance
- a nice page with description
  `www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm`
- important: Random Forests helps to
  - avoid overfitting (by random sampling the training data set)
  - select important/useful features (by random sampling the feature set)

# Building Random Forests

**The algorithm for building a tree in the ensemble**

1. Having a training set of the size $n$, sample $n$ cases at random - but with replacement, and use the sample to build a decision tree.
2. If there are $M$ input features, choose a less number $m \ll M$ (fixed for the whole procedure). When building the tree, at each node m variables are selected at random out of the $M$ and the best split on these $m$ features is used to split the node.
3. Each tree is grown to the largest extent possible. There is no pruning.

**The more trees in the ensemble, the better.**
**There is no risk of overfitting!**

# R packages for Random Forests

- **randomForest**: Breiman and Cutler's random forests for classification and regression
  – Classification and regression based on a forest of trees using random inputs.
- **RRF**: Regularized Random Forest
  – Feature Selection with Regularized Random Forest. This package is based on the 'randomForest' package by Andy Liaw. The key difference is the RRF function that builds a regularized random forest.
  – `http://cran.r-project.org/web/packages/RRF/index.html`
- **party**: A Laboratory for Recursive Partytioning
  – a computational toolbox for recursive partitioning
  – `cforest()` provides an implementation of Breiman's random forests
  – extensible functionality for visualizing tree-structured regression models is available

# Summary of Examination Requirements

**You should understand and be able to explain**

- Decision tree structure: internal nodes, terminal nodes, decisions
- Basic ideas of decision tree learning
- Tree growing: splitting criteria, classification tree, regression tree, ID3 algorithm and its extension C4.5
- Tree pruning: against overfitting
- Practical usage of decision trees in R (packages `rpart` or `tree`)
- Random Forests – idea, algorithm and advantages