Nearest neighbors. Kernel functions, SVM. Decision trees.

Petr Pošík

Czech Technical University in Prague
Faculty of Electrical Engineering
Dept. of Cybernetics

Method of k nearest neighbors

- Simple, non-parametric, instance-based method for supervised learning, applicable for both *classification* and *regression*.
- Do not confuse k-NN with
  - k-means (a clustering algorithm)
  - NN (neural networks)
- Training: Just *remember* the whole training dataset T.
- Prediction: To get the model prediction for a new data point $\boldsymbol{x}$ (query),
  - find the set $N_k(\boldsymbol{x})$ of k nearest neighbors of $\boldsymbol{x}$ in T using certain distance measure,
  - in case of **classification**, determine the predicted class $\hat{y} = h(\boldsymbol{x})$ as the majority vote among the nearest neighbors, i.e.

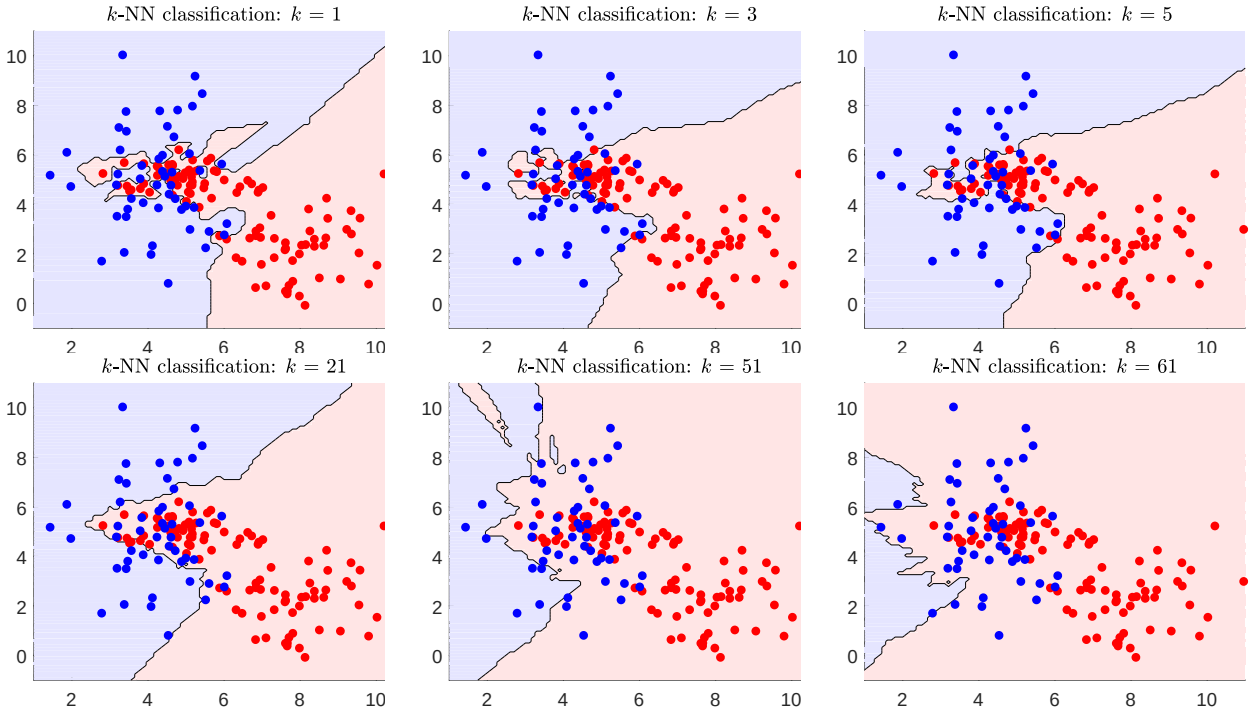
$$\hat{y} = h(\boldsymbol{x}) = \arg\max_y \sum_{(\boldsymbol{x}',y')\in N_k(\boldsymbol{x})} I(y' = y),$$

    where $I(P)$ is an indicator function (returns 1 if P is true, 0 otherwise).
  - in case of **regression**, determine the predicted value $\hat{y} = h(\boldsymbol{x})$ as the average of values y of the nearest neighbors, i.e.

$$\hat{y} = h(\boldsymbol{x}) = \frac{1}{k} \sum_{(\boldsymbol{x}',y')\in N_k(\boldsymbol{x})} y',$$

- What is the influence of k to the final model?

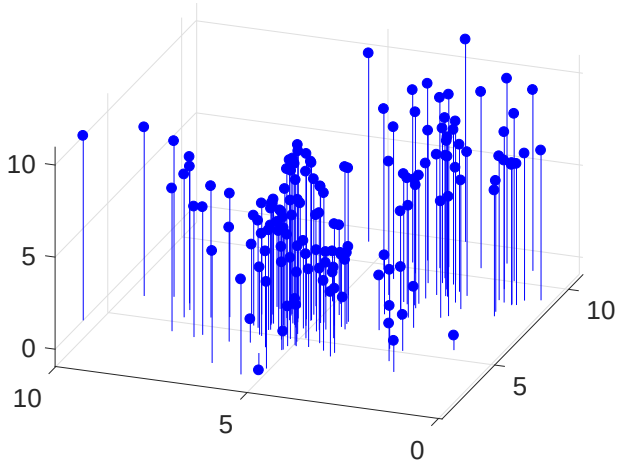
KNN classification: Example



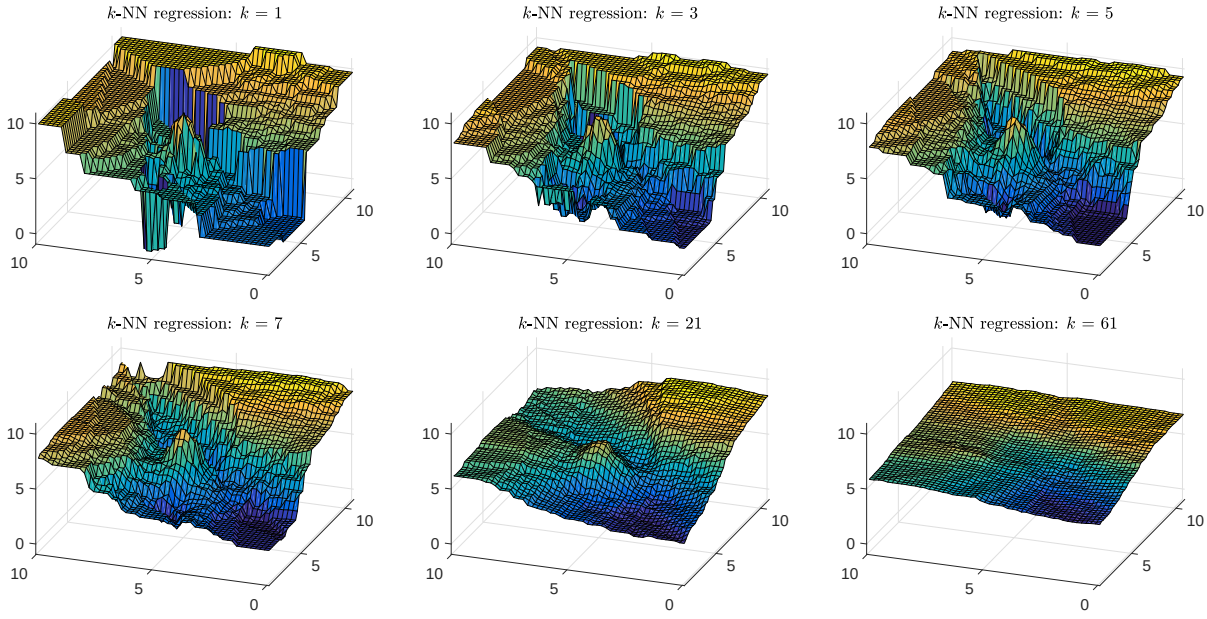
- Only in 1-NN, all training examples are classified correctly (unless there are two exactly the same observations with a different evaluation).
- Unbalanced classes may be an issue: the more frequent class takes over with increasing k.

k-NN Regression Example

The training data:



k-NN regression example



- For small k, the surface is rugged.
- For large k, too much averaging (smoothing) takes place.

k-NN Summary

Comments:

- For 1-NN, the division of the input space into convex cells is called a *Voronoi tassellation*.
- A *weighted variant* can be constructed:
  - Each of the k nearest neighbors has a weight inversely proportional to its distance to the query point.
  - Prediction is then done using weighted voting (in case of classification) or weighted averaging (in case of regression).
- In regression tasks, instead of averaging you can use e.g. (weighted) linear regression to compute the prediction.

Advantages:

- Simple and widely applicable method.
- For both classification and regression tasks.
- For both categorial and continuous predictors (independent variables).

Disadvantages:

- Must store the whole training set (there are methods for training set reduction).
- During prediction, it must compute the distances to all the training data points (can be alleviated e.g. by using KD-tree structure for the training set).

Overfitting prevention:

- Choose the right value of k e.g. using crossvalidation.

Support vector machine 8 / 40

Revision

Optimal separating hyperplane:

- A way to find a linear classifier optimal in certain sense by means of a quadratic program (dual task for soft margin version):

$$\text{maximize } \sum_{i=1}^{|T|} \alpha_i - \sum_{i=1}^{|T|}\sum_{j=1}^{|T|} \alpha_i \alpha_j y^{(i)} y^{(j)} \boldsymbol{x}^{(i)} {\boldsymbol{x}^{(j)}}^T \text{ w.r.t. } \alpha_1, \ldots, \alpha_{|T|}, \mu_1, \ldots, \mu_{|T|},$$

$$\text{subject to } \alpha_i \geq 0,\ \mu_i \geq 0,\ \alpha_i + \mu_i = C, \text{ and } \sum_{i=1}^{|T|} \alpha_i y^{(i)} = 0.$$

- The parameters of the hyperplane are given in terms of a weighted linear combination of support vectors:

$$\boldsymbol{w} = \sum_{i=1}^{|T|} \alpha_i y^{(i)} \boldsymbol{x}^{(i)}, \qquad w_0 = y^{(k)} - \boldsymbol{x}^{(k)} \boldsymbol{w}^T,$$

Basis expansion:

- Instead of a linear model $\langle \boldsymbol{w}, \boldsymbol{x} \rangle$, create a linear model of nonlinearly transformed features $\langle \boldsymbol{w}', \Phi(\boldsymbol{x}) \rangle$ which represents a nonlinear model in the original space.

What if we put these two things together?

Optimal separating hyperplane combined with the basis expansion

Using the optimal sep. hyperplane, the examples $\boldsymbol{x}$ occur only in the form of dot products:

the optimization criterion $\sum_{i=1}^{|T|} \alpha_i - \frac{1}{2} \sum_{i=1}^{|T|} \sum_{j=1}^{|T|} \alpha_i \alpha_j y^{(i)} y^{(j)} \boldsymbol{x}^{(i)} {\boldsymbol{x}^{(j)}}^T$

and in the decision rule $f(\boldsymbol{x}) = \text{sign}\left(\sum_{i=1}^{|T|} \alpha_i y^{(i)} \boldsymbol{x}^{(i)} \boldsymbol{x}^T + w_0 \right)$.

Application of the basis expansion changes

the optimization criterion to $\sum_{i=1}^{|T|} \alpha_i - \frac{1}{2} \sum_{i=1}^{|T|} \sum_{j=1}^{|T|} \alpha_i \alpha_j y^{(i)} y^{(j)} \Phi(\boldsymbol{x}^{(i)}) \Phi(\boldsymbol{x}^{(j)})^T$

and the decision rule to $f(\boldsymbol{x}) = \text{sign}\left(\sum_{i=1}^{|T|} \alpha_i y^{(i)} \Phi(\boldsymbol{x}^{(i)}) \Phi(\boldsymbol{x})^T + w_0 \right)$.

What if we use a scalar function $K(\boldsymbol{x}, \boldsymbol{x}')$ instead of the dot product in the image space?

The optimization criterion: $\sum_{i=1}^{|T|} \alpha_i - \frac{1}{2} \sum_{i=1}^{|T|} \sum_{j=1}^{|T|} \alpha_i \alpha_j y^{(i)} y^{(j)} K(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)})$

The discrimination function: $f(\boldsymbol{x}) = \text{sign}\left(\sum_{i=1}^{|T|} \alpha_i y^{(i)} K(\boldsymbol{x}^{(i)}, \boldsymbol{x}) + w_0 \right)$.

Kernel trick

Kernel function, or just a **kernel**:

- A generalized inner product (dot product, scalar product).
- A function of 2 vector arguments $K(\boldsymbol{a}, \boldsymbol{b})$ which provides values equal to the dot product $\Phi(\boldsymbol{a})\Phi(\boldsymbol{b})^T$ of the images of the vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ in certain high-dimensional image space.

Kernel trick:

- Let's have a linear algorithm in which the examples $\boldsymbol{x}$ occur only in dot products.
- Such an algorithm can be made non-linear by replacing the dot products of examples $\boldsymbol{x}$ with *kernels*.
- The result is the same as if the algorithm was trained in some high-dimensional image space with the coordinates given by many non-linear basis functions.
- Thanks to kernels, we do not need to explicitly perform the mapping from the input space to the highdimensional image space; the algorithm is much more efficient.

Frequently used kernels:

Polynomial: $K(\boldsymbol{a}, \boldsymbol{b}) = (\boldsymbol{a}\boldsymbol{b}^T + 1)^d$, where d is the degree of the polynom.

Gaussian (RBF): $K(\boldsymbol{a}, \boldsymbol{b}) = \exp\left(-\frac{|\boldsymbol{a} - \boldsymbol{b}|^2}{\sigma^2} \right)$, where σ^2 is the „width" of kernel.

Support vector machine

Support vector machine (SVM)

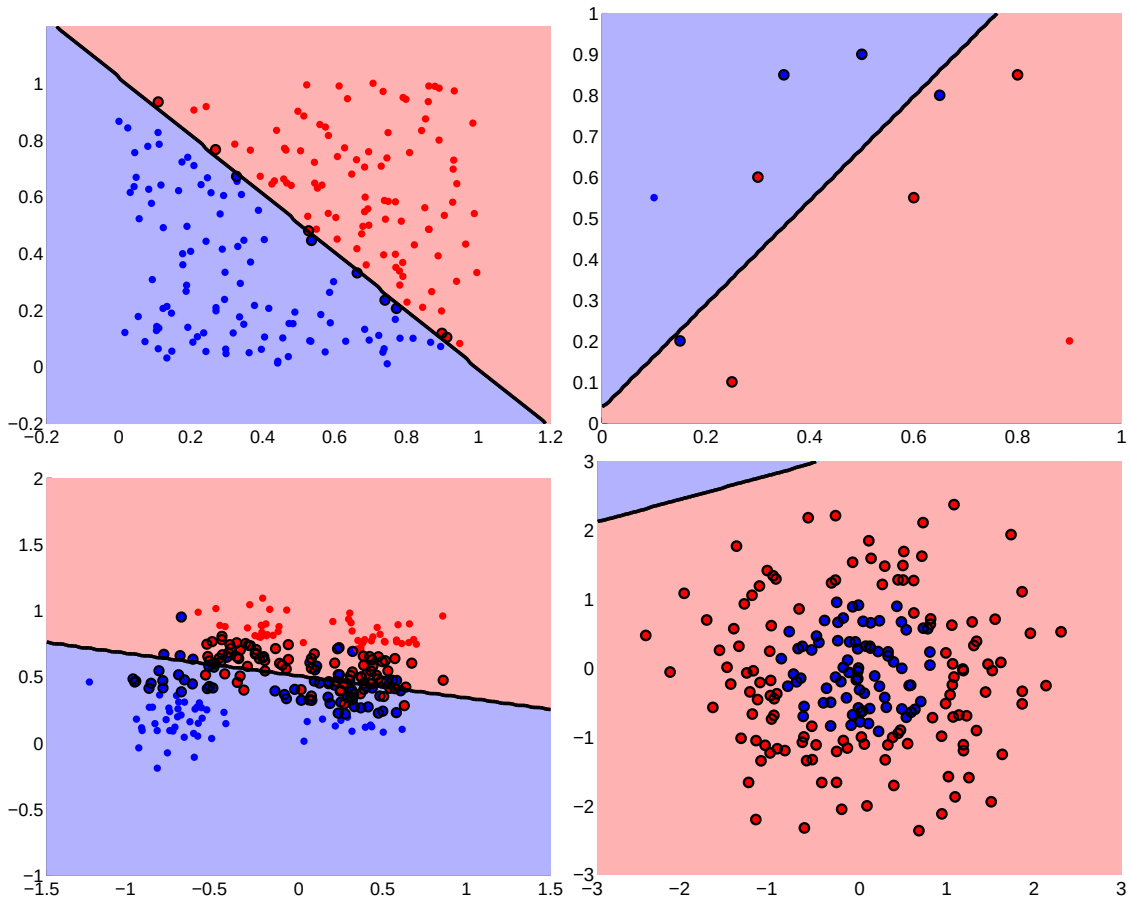
=

optimal separating hyperplane
learning algorithm

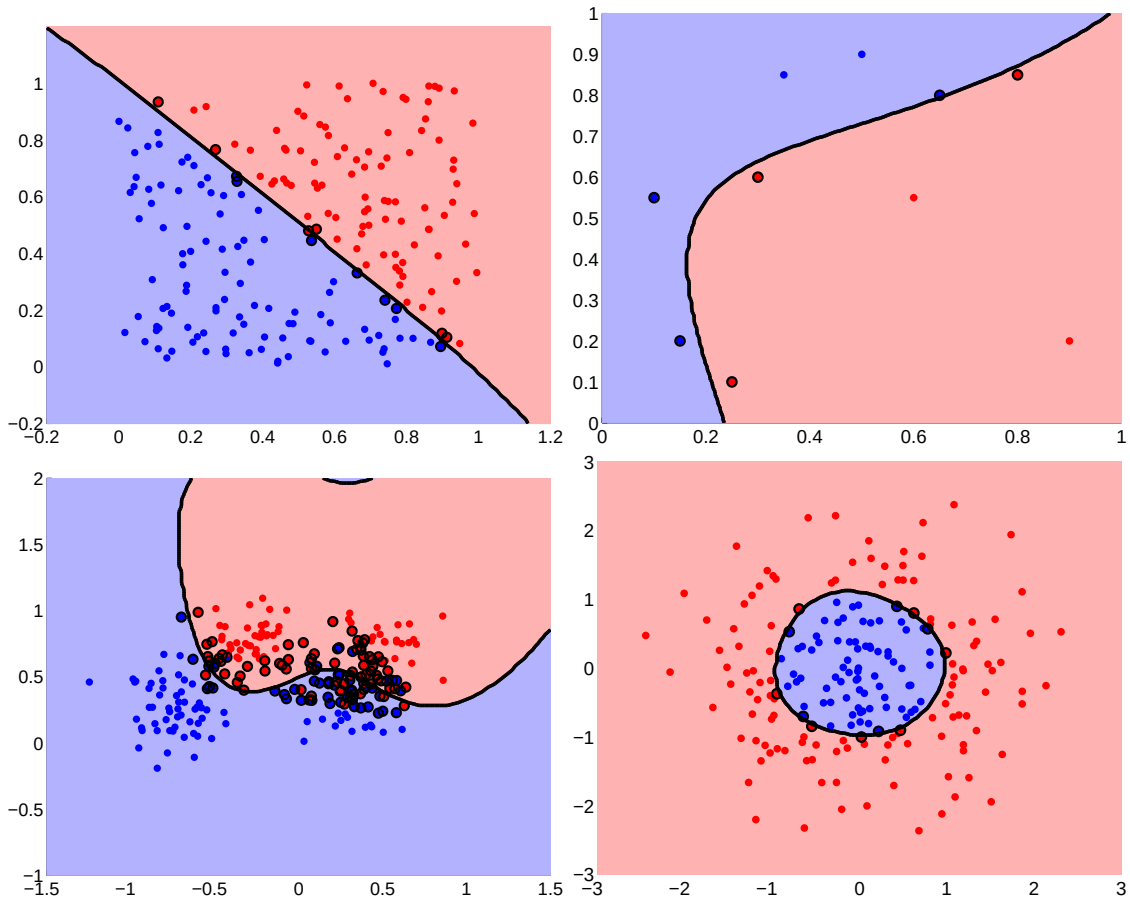
+

the kernel trick

Demo: SVM with linear kernel



Demo: SVM with RBF kernel



SVM: Summary

- SVM is a very popular model; in the past, the best performance for many tasks was achieved by SVM (nowadays, boosting or deep NN often perform better).
- When using SVM, you usually have to set
  - the kernel type,
  - kernel parameter(s), and
  - the (regularization) constant C,

  or use a method to find them automatically.
- Support vector regression (SVR) exists as well.
- There are many other (originally linear) methods that were kernelized:
  - kernel PCA,
  - kernel logistic regression,
  - ...

What is a decision tree?

Decision tree

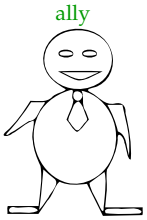
- is a function that
  - takes a vector of attribute values as its input, and
  - returns a "decision" as its output.
  - Both input and output values can be measured on a nominal, ordinal, interval, and ratio scales, can be discrete or continuous.
- The decision is formed via a sequence of tests:
  - each internal node of the tree represents a test,
  - the branches are labeled with possible outcomes of the test, and
  - each leaf node represents a decision to be returned by the tree.

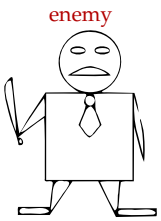
Decision trees examples:

- classification schemata in biology (cz: určovací klíče)
- diagnostic sections in illness encyclopedias
- online troubleshooting section on software web pages
- …

Attribute description

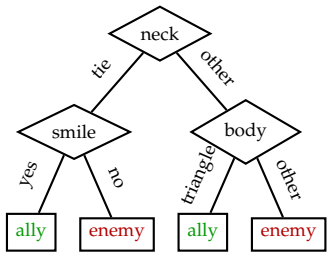
Example: A computer game.
The main character of the game meets various robots along his way. Some behave like allies, others like enemies.





head	body	smile	neck	holds	class
circle	circle	yes	tie	nothing	ally
circle	square	no	tie	sword	enemy
…	…	…	…	…	…

The game engine may use e.g. the following tree to assign the ally or enemy attitude to the generated robots:



Expressiveness of decision trees

The tree on previous slide is a Boolean decision tree:

- the decision is a binary variable (true, false), and
- the attributes are discrete.
- It returns ally iff the input attributes satisfy one of the paths leading to an ally leaf:

$$ally \Leftrightarrow (neck = tie \wedge smile = yes) \vee (neck = \neg tie \wedge body = triangle),$$

i.e. in general

- $Goal \Leftrightarrow (Path_1 \vee Path_2 \vee \ldots)$, where
- $Path$ is a conjuction of attribute-value tests, i.e.
- the tree is equivalent to a DNF (disjunctive normal form) of a function.

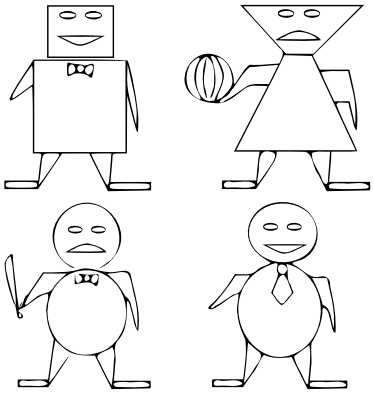
Any function in propositional logic can be expressed as a dec. tree.

- Trees are a suitable representation for some functions and unsuitable for others.
- What is the cardinality of the set of Boolean functions of n attributes?
  - It is equal to the number of truth tables that can be created with n attributes.
  - The truth table has 2^n rows, i.e. there is 2^{2^n} different functions.
  - The set of trees is even larger; several trees represent the same function.
- We need a clever algorithm to find good hypotheses (trees) in such a large space.

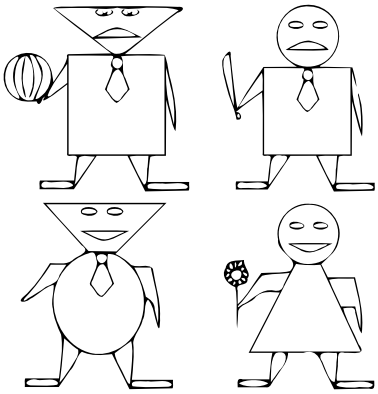
A computer game

Example 1:
Can you distinguish between allies and enemies after seeing a few of them?

Allies



Enemies

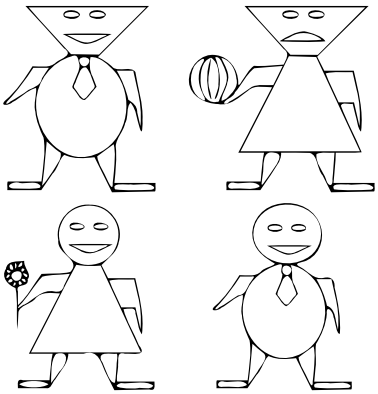


Hint: concentrate on the shapes of heads and bodies.
Answer: Seems like allies have the same shape of their head and body.
How would you represent this by a decision tree? (Relation among attributes.)
How do you know that you are right?

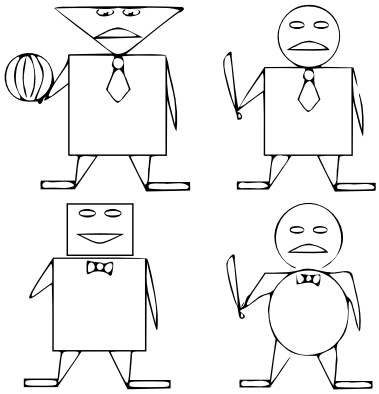
A computer game

Example 2:
Some robots changed their attitudes:

Allies



Enemies



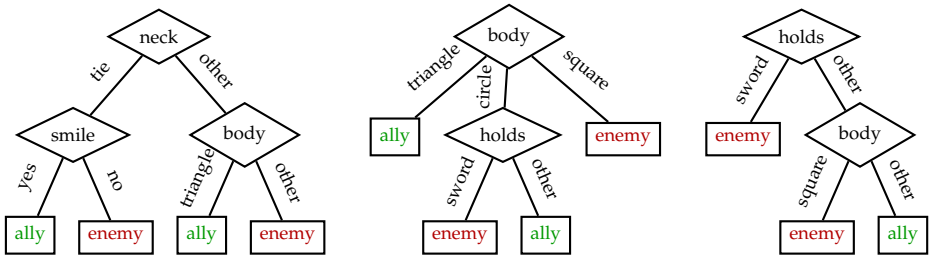
No obvious simple rule.
How to build a decision tree discriminating the 2 robot classes?

Alternative hypotheses

Example 2: Attribute description:

head	body	smile	neck	holds	class
triangle	circle	yes	tie	nothing	ally
triangle	triangle	no	nothing	ball	ally
circle	triangle	yes	nothing	flower	ally
circle	circle	yes	tie	nothing	ally
triangle	square	no	tie	ball	enemy
circle	square	no	tie	sword	enemy
square	square	yes	bow	nothing	enemy
circle	circle	no	bow	sword	enemy

Alternative hypotheses (suggested by an oracle for now): Which of the trees is the best (right) one?



How to choose the best tree?

We want a tree that is

- **consistent** with the training data,
- is as **small** as possible, and
- which also **works for new data**.

Consistent with data?

- All 3 trees are consistent.

Small?

- The right-hand side one is the simplest one:

	left	middle	right
depth	2	2	2
leaves	4	4	3
conditions	3	2	2

Will it work for new data?

- We have no idea!
- We need a set of new testing data (different data from the same source).

Learning a Decision Tree

It is an intractable problem to find **the smallest consistent tree** among $> 2^{2^n}$ trees.
We can find approximate solution: **a small (but not the smallest) consistent tree**.

Top-Down Induction of Decision Trees (TDIDT):

- A greedy divide-and-conquer strategy.
- Progress:
  1. Find the most important attribute.
  2. Divide the data set using the attribute values.
  3. For each subset, build an independent tree (recursion).
- "Most important attribute": attribute that makes the most difference to the classification.
- All paths in the tree will be short, the tree will be shallow.

Attribute importance

head	body	smile	neck	holds	class
triangle	circle	yes	tie	nothing	ally
triangle	triangle	no	nothing	ball	ally
circle	triangle	yes	nothing	flower	ally
circle	circle	yes	tie	nothing	ally
triangle	square	no	tie	ball	enemy
circle	square	no	tie	sword	enemy
square	square	yes	bow	nothing	enemy
circle	circle	no	bow	sword	enemy

triangle:	2:1	triangle:	2:0	yes:	3:1	tie:	2:2	ball:	1:1
circle:	2:2	circle:	2:1	no:	1:3	bow:	0:2	sword:	0:2
square:	0:1	square:	0:3			nothing:	2:0	flower:	1:0
								nothing:	2:1

A perfect attribute divides the examples into sets containing only a single class. (Do you remember the simply created perfect attribute from Example 1?)

A useless attribute divides the examples into sets containing the same distribution of classes as the set before splitting.

None of the above attributes is perfect or useless. Some are more useful than others.

Choosing the best attribute

Information gain:

- Formalization of the terms "useless", "perfect", "more useful".
- Based on entropy, a measure of the uncertainty of a random variable V with possible values v_i:

$$H(V) = -\sum_i p(v_i) \log_2 p(v_i)$$

- Entropy of the target variable C (usually a class) measured on a data set S (a finite-sample estimate of the true entropy):

$$H(C,S) = -\sum_i \hat{p}(c_i) \log_2 \hat{p}(c_i),$$

where $\hat{p}(c_i) = \frac{N_S(c_i)}{|S|}$, and $N_S(c_i)$ is the number of examples in S that belong to class c_i.

- The entropy of the target variable C **remaining in the data set S after splitting** into subsets S_k using values of attribute A (weighted average of the entropies in individual subsets):

$$H(C,S,A) = \sum_k \hat{p}(S_k) H(C,S_k), \qquad \text{where } \hat{p}(S_k) = \frac{|S_k|}{|S|}$$

- The information gain of attribute A for a data set S is

$$Gain(A,S) = H(C,S) - H(C,S,A).$$

Choose the attribute with the highest information gain, i.e. the attribute with the lowest $H(C,S,A)$.

Choosing the test attribute (special case: binary classification)

- For a Boolean random variable V which is true with probability q, we can define:

$$H_B(q) = -q \log_2 q - (1-q) \log_2 (1-q)$$

- Specifically, for $q = 0.5$,

$$H_B(0.5) = -\frac{1}{2} \log_2 \frac{1}{2} - \left(1 - \frac{1}{2}\right) \log_2 \left(1 - \frac{1}{2}\right) = 1$$

- Entropy of the target variable C measured on a data set S with N_p positive and N_n negative examples:

$$H(C,S) = H_B\left(\frac{N_p}{N_p + N_n}\right) = H_B\left(\frac{N_p}{|S|}\right)$$

Choosing the test attribute (example)

head		body		smile		neck		holds	
triangle:	2:1	triangle:	2:0	yes:	3:1	tie:	2:2	ball:	1:1
circle:	2:2	circle:	2:1	no:	1:3	bow:	0:2	sword:	0:2
square:	0:1	square:	0:3			nothing:	2:0	flower:	1:0
								nothing:	2:1

head:
$p(S_{\text{head=tri}}) = \frac{3}{8}$; $H(C, S_{\text{head=tri}}) = H_B\left(\frac{2}{2+1}\right) = 0.92$
$p(S_{\text{head=cir}}) = \frac{4}{8}$; $H(C, S_{\text{head=cir}}) = H_B\left(\frac{2}{2+2}\right) = 1$
$p(S_{\text{head=sq}}) = \frac{1}{8}$; $H(C, S_{\text{head=sq}}) = H_B\left(\frac{0}{0+1}\right) = 0$
$H(C,S,head) = \frac{3}{8} \cdot 0.92 + \frac{4}{8} \cdot 1 + \frac{1}{8} \cdot 0 = 0.84$
$Gain(head, S) = 1 - 0.84 = 0.16$

body:
$p(S_{\text{body=tri}}) = \frac{2}{8}$; $H(C, S_{\text{body=tri}}) = H_B\left(\frac{2}{2+0}\right) = 0$
$p(S_{\text{body=cir}}) = \frac{3}{8}$; $H(C, S_{\text{body=cir}}) = H_B\left(\frac{2}{2+1}\right) = 0.92$
$p(S_{\text{body=sq}}) = \frac{3}{8}$; $H(C, S_{\text{body=sq}}) = H_B\left(\frac{0}{0+3}\right) = 0$
$H(C,S,body) = \frac{2}{8} \cdot 0 + \frac{3}{8} \cdot 0.92 + \frac{3}{8} \cdot 0 = 0.35$
$Gain(body, S) = 1 - 0.35 = 0.65$

smile:
$p(S_{\text{smile=yes}}) = \frac{4}{8}$; $H(C, S_{\text{yes}}) = H_B\left(\frac{3}{3+1}\right) = 0.81$
$p(S_{\text{smile=no}}) = \frac{4}{8}$; $H(C, S_{\text{no}}) = H_B\left(\frac{1}{1+3}\right) = 0.81$
$H(C,S,smile) = \frac{4}{8} \cdot 0.81 + \frac{4}{8} \cdot 0.81 + \frac{3}{8} \cdot 0 = 0.81$
$Gain(smile, S) = 1 - 0.81 = 0.19$

neck:
$p(S_{\text{neck=tie}}) = \frac{4}{8}$; $H(C, S_{\text{neck=tie}}) = H_B\left(\frac{2}{2+2}\right) = 1$
$p(S_{\text{neck=bow}}) = \frac{2}{8}$; $H(C, S_{\text{neck=bow}}) = H_B\left(\frac{0}{0+2}\right) = 0$
$p(S_{\text{neck=no}}) = \frac{2}{8}$; $H(C, S_{\text{neck=no}}) = H_B\left(\frac{2}{2+0}\right) = 0$
$H(C,S,neck) = \frac{4}{8} \cdot 1 + \frac{2}{8} \cdot 0 + \frac{2}{8} \cdot 0 = 0.5$
$Gain(neck, S) = 1 - 0.5 = 0.5$

holds:
$p(S_{\text{holds=ball}}) = \frac{2}{8}$; $H(C, S_{\text{holds=ball}}) = H_B\left(\frac{1}{1+1}\right) = 1$
$p(S_{\text{holds=swo}}) = \frac{2}{8}$; $H(C, S_{\text{holds=swo}}) = H_B\left(\frac{0}{0+2}\right) = 0$
$p(S_{\text{holds=flo}}) = \frac{1}{8}$; $H(C, S_{\text{holds=flo}}) = H_B\left(\frac{1}{1+0}\right) = 0$
$p(S_{\text{holds=no}}) = \frac{3}{8}$; $H(C, S_{\text{holds=no}}) = H_B\left(\frac{2}{2+1}\right) = 0.92$
$H(C,S,holds) = \frac{2}{8} \cdot 1 + \frac{2}{8} \cdot 0 + \frac{1}{8} \cdot 0 + \frac{3}{8} \cdot 0.92 = 0.6$
$Gain(holds, S) = 1 - 0.6 = 0.4$

The **body** attribute

- brings us the largest information gain, thus
- it shall be chosen for the first test in the tree!

Choosing subsequent test attribute

No further tests are needed for robots with triangular and squared bodies.
Dataset for robots with circular bodies:

head		body	smile		neck		holds		class
triangle		circle	yes		tie		nothing		ally
circle		circle	yes		tie		nothing		ally
circle		circle	no		bow		sword		enemy
triangle:	1:0		yes:	2:0	tie:	2:0	nothing:	2:0	
circle:	1:1		no:	0:1	bow:	0:1	sword:	0:1	

All the attributes **smile**, **neck**, and **holds**

- take up the remaining entropy in the data set, and
- are equally good for the test in the group of robots with circular bodies.

Decision tree building procedure

Algorithm 1: BuildDT

Input : the set of examples S,
the set of attributes $\mathcal{A}$,
majority class of the parent node C_P
Output: a decision tree

```
1  begin
2      if S is empty then
3          return leaf with C_P
4      C ← majority class in S
5      if all examples in S belong to the same class C then
6          return leaf with C
7      if A is empty then
8          return leaf with C
9      A ← arg max_{a∈A} Gain(a, S)
10     T ← a new decision tree with root test on attribute A
11     foreach value v_k of A do
12         S_k ← {x | x ∈ S ∧ x.A = v_k}
13         t_k ← BuildDT(S_k, A − A, C)
14         add branch to T with label A = v_k and attach a subtree t_k
15     return tree T
```

Algorithm characteristics

- There are many hypotheses (trees) consistent with the dataset S; the algorithm will return any of them, unless there is some bias in choosing the tests.
- The current set of considered hypotheses has always only 1 member (greedy selection of the successor). The algorithm cannot provide answer to the question how many hypotheses consistent with the data exist.
- The algorithm does not use backtracking; it can get stuck in a local optimum.
- The algorithm uses batch learning, not incremental.

How to prevent overfitting for trees?

Tree pruning:

- Let's have a fully grown tree T.
- Choose a test node having only leaf nodes as descensdants.
- If the test appears to be irrelevant, remove the test and replace it with a leaf node with the majority class.
- Repeat, until all tests seem to be relevant.

How to check if the split is (ir)relevant?

1. Using statistical χ^2 test:
   - If the distribution of classes in the leaves does not differ much from the distribution of classes in their parent, the split is irrelevant.
2. Using an (independent) validation data set:
   - Create a temporary tree by replacing a subtree with a leaf.
   - If the error on validation set decreased, accept the pruned tree.

Early stopping:

- Hmm, if we grow the tree fully and then prune it, why cannot we just stop the tree building when there is no good attribute to split on?
- Prevents us from recognizing situations when
  - there is no single good attribute to split on, but
  - there are combinations of attributes that lead to a good tree!

Missing data

Decision trees are one of the rare model types **able to handle missing attribute values**.

1. Given a complete tree, how to classify an example with a missing attribute value needed for a test?
   - Pretend that the object has all possible values for this attribute.
   - Track all possible paths to the leaves.
   - The leaf decisions are weighted using the number of training examples in the leaves.
2. How to build a tree if the training set contains examples with missing attribute values?
   - Introduce a new attribute value: "Missing" (or N/A).
   - Build tree in a normal way.

Multivalued attributes

What if the training set contains e.g. name, social insurance number, or other id?

- When each example has a unique value of an attribute A, the information gain of A is equal to the entropy of the whole data set!
- Attribute A is chosen for the tree root; yet, such a tree is useless (overfitted).

Solutions:

1. Allow only Boolean test of the form $A = v_k$ and allow the remaining values to be tested later in the tree.
2. Use a different split importance measure instead of $Gain$, e.g. $GainRatio$:
   - Normalize the information gain by a maximal amount of information the split can have:

$$GainRatio(A, S) = \frac{Gain(A, S)}{H(A, S)},$$

   where $H(A, S)$ is the entropy of attribute A and represents the largest information gain we can get from splitting using A.

Attributes with different prices

What if the tests in the tree also cost us some "money"?

- Then we would like to have the cheap test close to the root.
- If we have $Cost(A) \in \langle 0, 1 \rangle$ then we can use e.g.

$$\frac{Gain^2(A,S)}{Cost(A)},$$

or

$$\frac{2^{Gain(A,S)} - 1}{(Cost(A)+1)^w}$$

to bias the preference for cheaper tests.

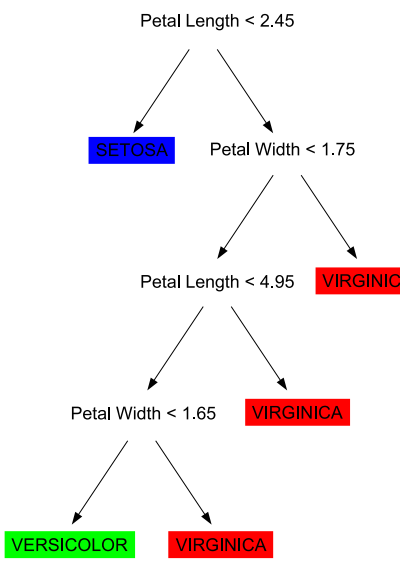
Continuous input attributes

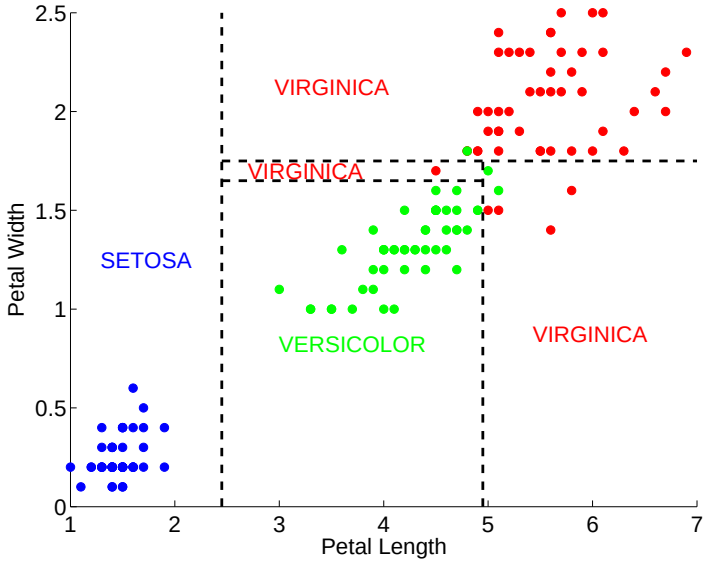
Continuous or integer-valued input attributes:

- Use binary splits with the highest information gain.
  - Sort the values of the attribute.
  - Consider only split points lying between 2 examples with different classification.

Temperature	-20	-9	-2	5	16	26	32	35
Go out?	No	No	Yes	Yes	Yes	Yes	No	No

- Previously used **attributes can be used again** in subsequent tests!

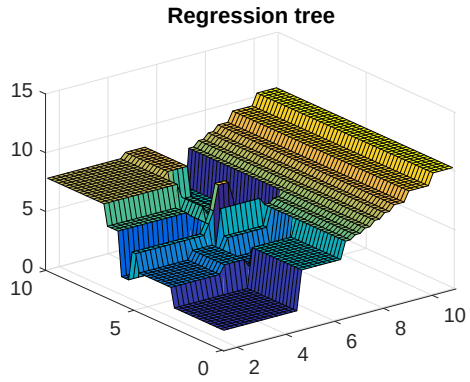




Continuous output variable

Regression tree:

- In each leaf, it can have
  - a **constant value** (usually an average of the output variable over the training set), or
  - a **linear function** of some subset of numerical input attributes
- The learning algorithm must decide when to stop splitting and begin applying linear regression.



Trees: Summary

- Decision trees belong to the simplest, most universal and most widely used prediction models.
- They are often used in ensemble methods as a building block.
- They are not suitable for all modeling problems (relations, etc.).
- TDIDT is the most widely used technique to build a tree from data.
- It uses greedy divide-and-conquer approach.
- Individual tree variants differ mainly
  - in what type of attributes they are able to handle,
  - in the attribute importance measure (information gain, gain ratio, Gini index, χ^2, etc.),
  - if they make enumerative or just binary splits,
  - if and how they can handle missing data,
  - whether they do only axis-parallel splits, or allow for oblique trees,
  - etc.

Competencies

After this lecture, a student shall be able to ...

- explain, use, and implement the method of k nearest neighbors for both classification and regression;
- explain the influence of k on the form of the final model;
- describe advantages and disadvantages of k-NN, and suggest a way hot to find a suitable value of k;
- show how to force the algorithm for learning the optimal separating hyperplane to find a nonlinear model using basis expansion, and using a kernel function;
- explain the meaning of kernels, and their advantages compared to basis expansion;
- explain the principle of support vector machine;
- describe the structure of a classification and regression tree, and the way it is used to determine a prediction;
- know a lower bound on the number of Boolean decision trees for a dataset with n attributes;
- describe TDIDT algorithm and its features, and know whether it will find the optimal tree;
- explain how to choose the best attribute for a split, and be able to manually perform the choice for simple examples;
- describe 2 methods to prevent tree overfitting, and argue which of them is better;
- explain how a decision tree can handle missing data during training and during prediction;
- describe what happens in a tree-building algorithm and what to do if the dataset contains an attribute with unique value for each observation;
- explain how to handle continuous input and output variables (as opposed to the discrete attributes).