Introduction to Machine Learning

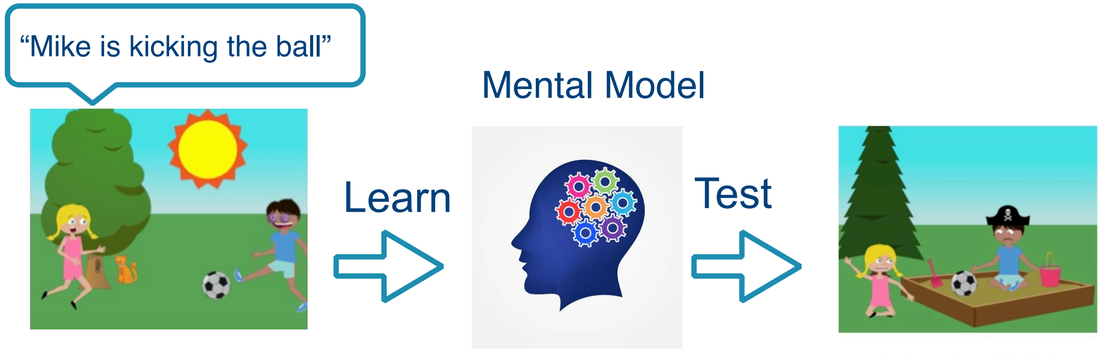
H. Blockeel

Machine Learning for Constraint Programming
Summer school, Leuven, July 2023

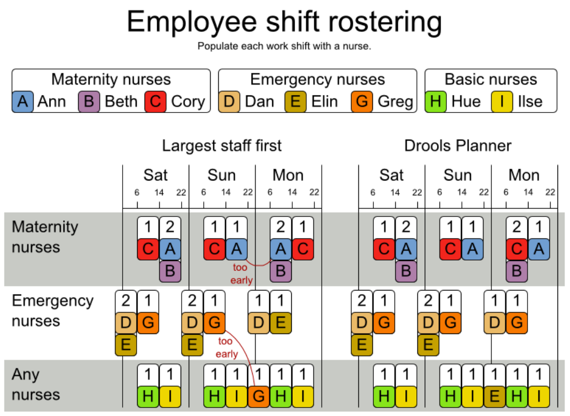
What is Machine Learning?

You probably heard of it…

How would you define machine learning?



1. mike is wearing a pirate hat
2. jenny is in the sandbox
3. mike is kicking the soccer ball
4. jenny is angry at mike
5. mike is in the sandbox



Many applications…

- **Recommender systems**: e.g., Google, Facebook, Amazon, … try to show you ads you might like
- **Email spam filters**: by observing which mails you flag as “spam”, try to learn your preferences
- **Natural language processing**: e.g., Sentiment analysis: is this movie review mostly positive or negative?
- **Vision**: learn to recognize pedestrians, …
- … and many, many more

P. Domingos’ bestseller *The Master Algorithm* provides an excellent account of how machine learning affects our daily life

KU LEUVEN

Definitions of machine learning?

- Tom Mitchell, 1996: Machine learning is the *study of how to make programs improve their performance on certain tasks from (own) experience*
  - “performance” = speed, accuracy, …
  - “experience” = earlier observations
- “Improve performance” in the most general meaning: this includes learning *from scratch.*
- Useful (in principle) for anything that we *don’t know how to program* - computer “programs itself”
  - Vision: recognizing faces, traffic signs, …
  - Game playing, e.g., AlphaGo
- Link to artificial intelligence : computer solves hard problems autonomously

Machine learning vs. other AI

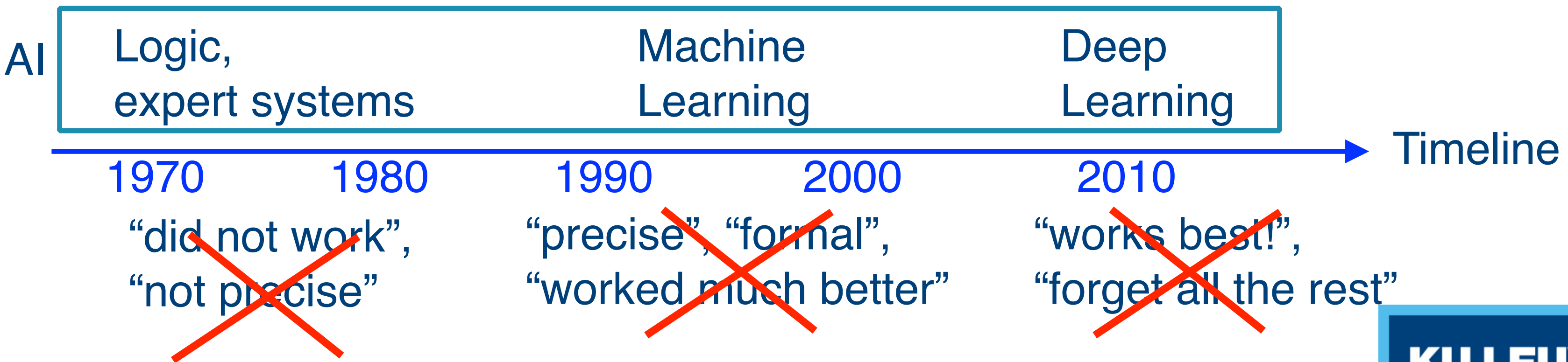
- In machine learning, the key is **data**
  - *Examples* of questions & their answer
  - *Observations* of earlier attempts to solve some problem

- Machine learning makes use of *inductive inference*: reasoning from *specific* to *general*
  - In statistics: sample → population
  - In philosophy of science: concrete observations → general theory
  - In machine learning: observations → any situation

- This aspect of machine learning links it to data mining, data analysis, statistics, …

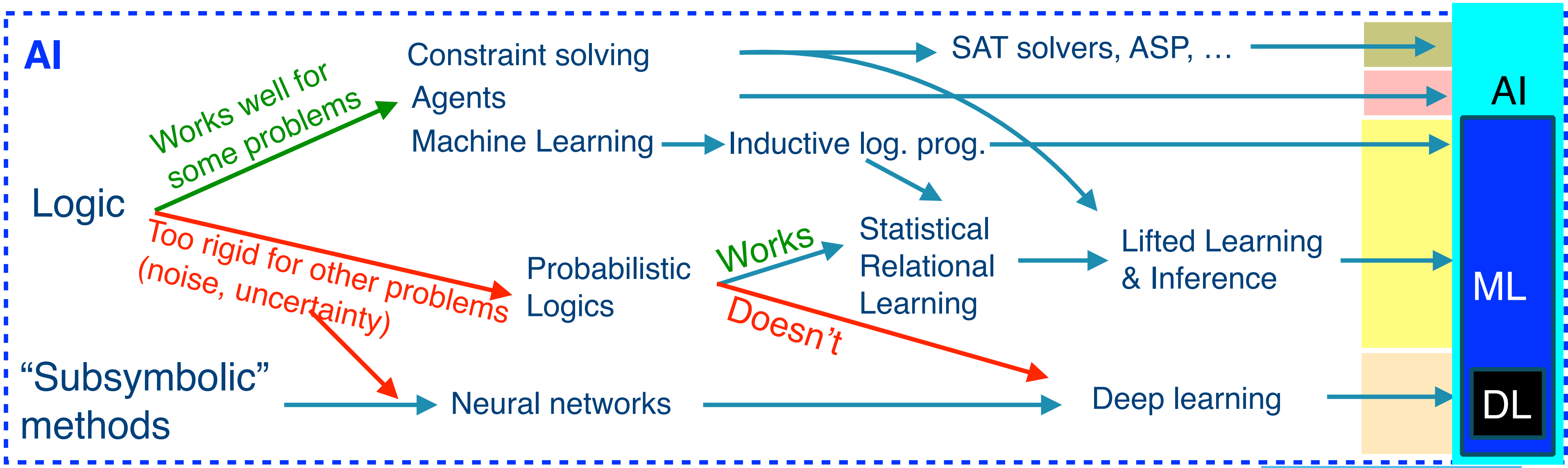
Machine learning and AI

- Many misconceptions about machine learning these days
- In the latest 15 years or so, "*Deep Learning*" has received a lot of attention: it revolutionized computer vision, speech recognition, natural language processing
- Avalanche of new researchers drawn to the field, without knowledge of the broader field of AI, or history of ML ("AI = ML = deep learning")
- See, e.g., A. Darwiche, https://www.youtube.com/watch?v=UTzCwCic-Do (also published in Communications of the ACM, October 2018)


AI
Logic, expert systems
Machine Learning
Deep Learning
Timeline
1970
1980
1990
2000
2010
~~"did not work", "not precise"~~
~~"precise", "formal", "worked much better"~~
~~"works best!", "forget all the rest"~~

Machine learning and AI

- There is *still* progress on *all* fronts, deep learning is just one of them
- This course reflects that viewpoint
- (schema below is incomplete, just serves to illustrate complexity of scientific impact)



Tasks Techniques Models Applications

The machine learning landscape

Automata

Statistical relational learning

Support vector machines

Neural networks

Natural language processing

Recommender systems

Regression

Deep learning

Nearest neighbors

Clustering

Decision trees

Convex optimization

Rule learners

Classification

Matrix factorization

Transfer learning

Greedy search

Bayesian learning

Vision

Probabilistic graphical models

Reinforcement learning

Learning theory

Speech

KU LEUVEN

Basic concepts and terminology

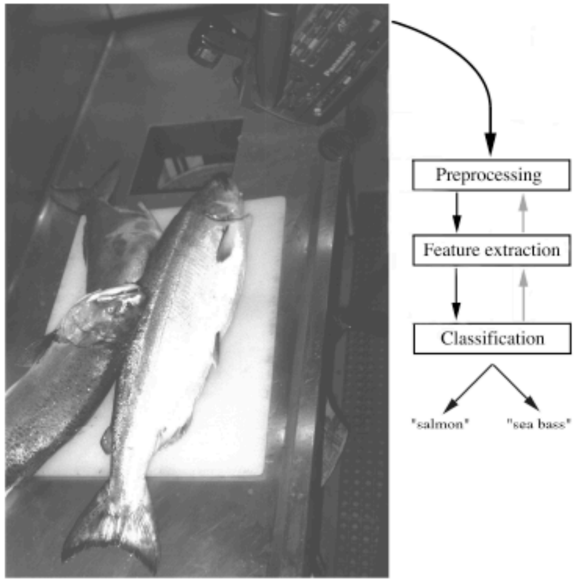
Machine learning

- ML in its typical form:
  - Input = dataset
  - Output = some kind of model
- ML in its most general form:
  - input = knowledge
  - output = a more general form of knowledge
- Learning = inferring general knowledge from more specific knowledge (observations ➔ model) = *inductive* inference

- Learning methods are often categorized according to the format of the input and output, and according to the goal of the learning process (but there are many more dimensions along which they can be categorized)

A typical task

- Given examples of pictures + label (saying what’s on the picture), infer a procedure that will allow you to correctly label new pictures
- E.g.: learn to classify fish as “salmon” or “sea bass”



e.g. contour extraction

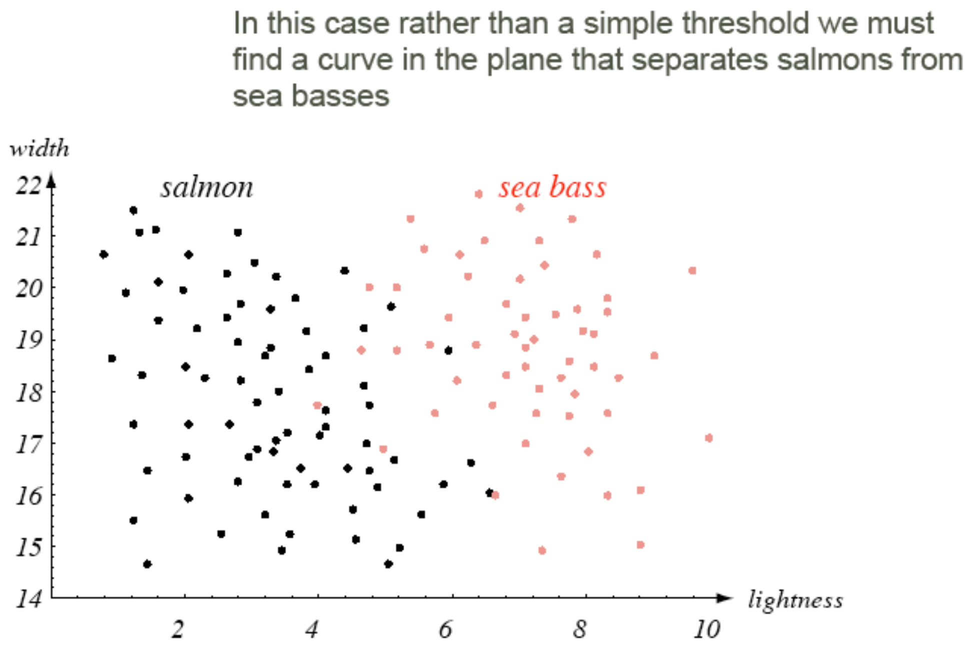
e.g. measure length and lightness

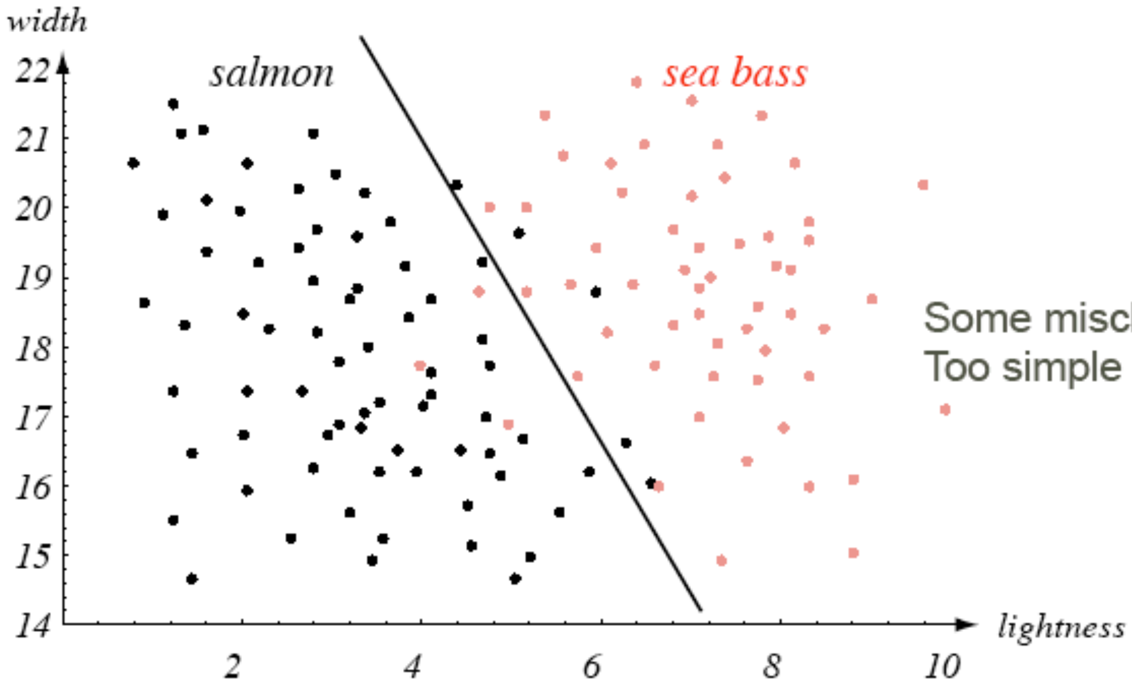
machine that makes decision given the above features

Example from: Pattern Classification (2nd ed) by R. O. Duda, P. E. Hart and D. G. Stork, John Wiley & Sons, 2000

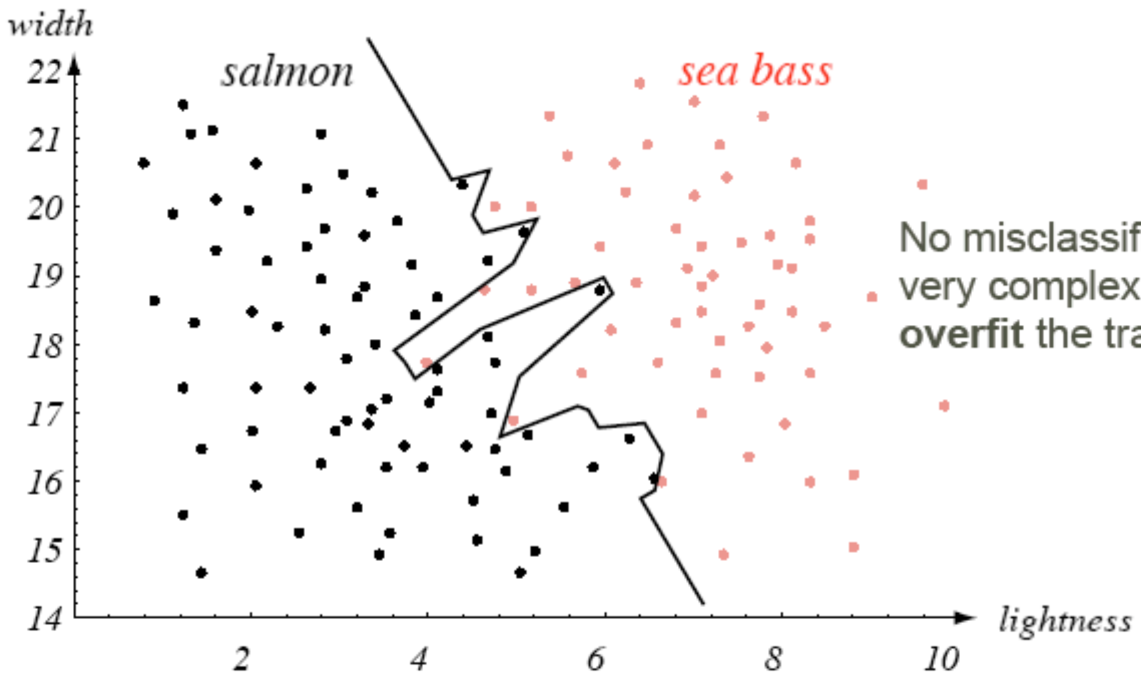
A generic approach

- Find informative **features** (here: lightness, width)
- Find a line/curve/hyperplane/… in this feature space that separates the classes


In this case rather than a simple threshold we must find a curve in the plane that separates salmons from sea basses

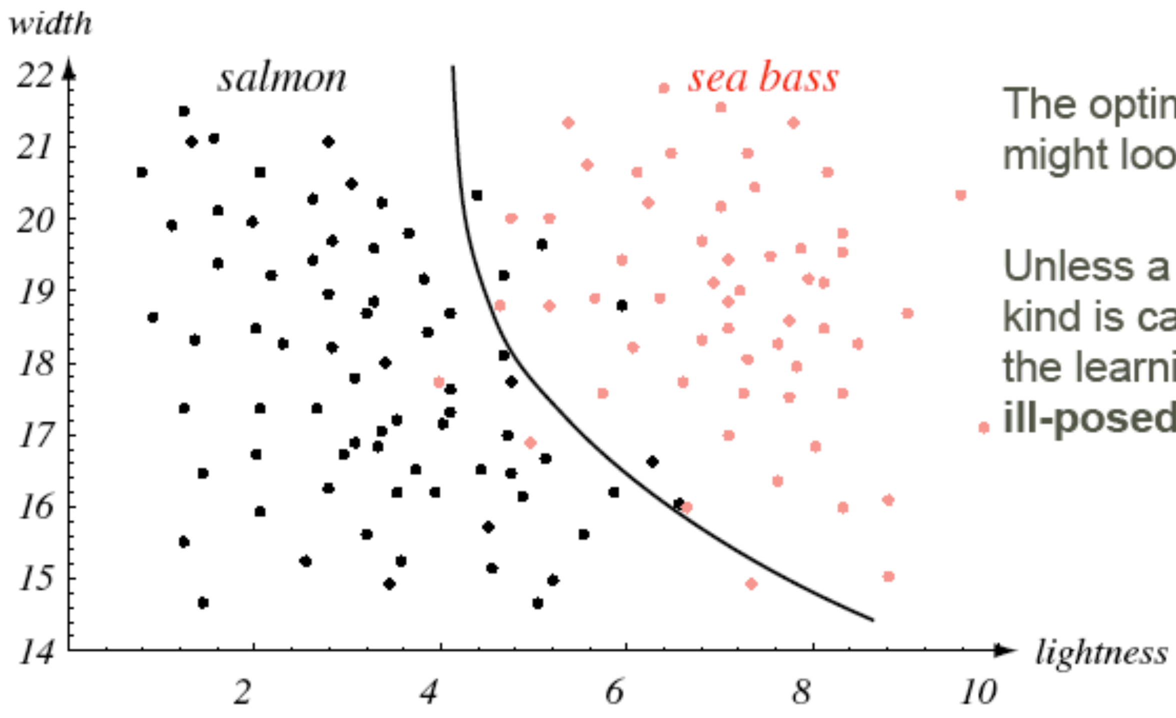


Some misclassified fishes
Too simple decision?



No misclassifications but...
very complex decisions can
overfit the training data

KU LEUVEN



The optimal tradeoff might look like this

Unless a tradeoff of this kind is carefully defined the learning problem is **ill-posed!**

Predictive versus descriptive

- **Predictive learning** : learn a model that can predict a particular property / attribute / variable from inputs
  - Many tasks are special cases of predictive learning
  - E.g., face recognition: given a picture of a face, say who it is
  - E.g., spam filtering: given an email, say whether it's spam or not

Name of task	*Learns a model that can …*
Concept learning / Binary classification	***Distinguish*** *instances of class C from other instances*
Classification	*Assign a* ***class*** *C (from a given set of classes) to an instance*
Regression	*Assign a* ***numerical*** *value to an instance*
Multi-label classification	*Assign a* ***set*** *of labels (from a given set) to an instance*
Multivariate regression	*Assign a* ***vector*** *of numbers to an instance*
Multi-target prediction	*Assign a* ***vector*** *of values (numerical, categorical) to an instance*
Ranking	*Assign* ***≤ or >*** *to a* ***pair*** *of instances*

Predictive versus <u>descriptive</u>

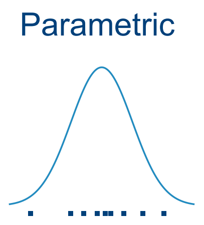
- **Descriptive learning** : given a data set, <u>describe</u> certain patterns in the dataset (or in the population it is drawn from)
- E.g., analyzing large databases:
  - "Bank X always refuses loans to people who earn less than 1200 euros per month"
  - "99.7% of all pregnant patients in this hospital are female"
  - "At supermarket X, people who buy cheese are twice as likely to also buy wine"

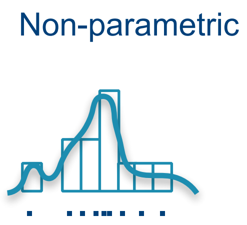
Function learning

- **Task** : learn a function X →Y that fits the given data (with X and Y sets of variables that occur in the data)
- Such a function will obviously be useful for predicting Y from X
- May also be descriptive, if we can understand the function
- Often, some **family of functions F** is given, and we need to estimate the parameters of the function f in F that best fits the data
  - e.g., linear regression : determine a and b such that $y = ax + b$ fits the data as well as possible
- What does "fit the data" mean? Measured by a so-called **loss function**
  - e.g., quadratic loss: $\sum_{(x,y)\in D} (f(x) - y)^2$ with f the learned function and D the dataset

Distribution learning

- **Task:** given a data set drawn from a distribution, estimate this distribution
- Often made distinction: *parametric* vs. *non-parametric*
  - Parametric: a family of distributions is given (e.g., “Gaussian”), we only need to estimate the parameters of the target distribution
  - Non-parametric: no specific family is assumed
- Often made distinction: *generative* vs. *discriminative*
  - Generative: learn the joint probability distribution (JPD) over all variables (once you have that, you can *generate* new instances by random sampling from it)
  - Discriminative: learn a conditional probability distribution of Y given X, for some given set of variables X (called input variables) and Y (called target variables)





These categorizations are somewhat fuzzy…

- A descriptive pattern may be useful for prediction
  - “Bank X always refuses loans to people who earn less than 1200 euros per month” (description)
  - Bob earns 1100 euros per month => Bank X will not give him a loan
- While functions are directly useful for prediction, a probability distribution can be used just as well
  - Given known information X, predict as value for Y, the value with the highest conditional probability given X

Parametric vs. non-parametric

- Parametric: a family of functions (or distributions, or …) is given, and each function is uniquely defined by the values of a fixed set of parameters
  - e.g. (function learning): linear regression
  - e.g. (distribution learning): fitting a gaussian
- Non-parametric: no specific family of functions is assumed
  - Typically, we are searching a space that contains models with varying structure, rather than just different parameter values
  - This often requires searching a discrete space
  - E.g.: decision trees, rules, …. (see later)

Link with "explainable AI"

- **Explainable AI** (**XAI**) refers to the study of AI systems that can explain their decisions / whose decisions we can understand
- Two different levels here:
  - We understand the (learned) model used for decision making [*global*]
  - We understand the individual decision [*local*]
- E.g. "I could not get a loan because I earn too little": we can understand this decision even if we don't know the whole decision process the bank uses
- A learned model that is not straightforward to interpret, is called a **black-box** model
- Machine learning poses additional challenges for XAI, as it often learns black-box models

Responsible AI : challenges

- **Privacy-preserving** data analysis
  - We need lots of data to learn from; this may include personal data
  - How can we guarantee that the analysis of these data will not violate the privacy of the people whose data this is?
  - Generally, when data is collected, consent is needed for a specific purpose, and data must be used solely for that purpose — how can we guarantee it won't be abused?

- Learning "**safe**" models : models that will not violate certain constraints that are imposed (including constraints on bias, discrimination, privacy, …)

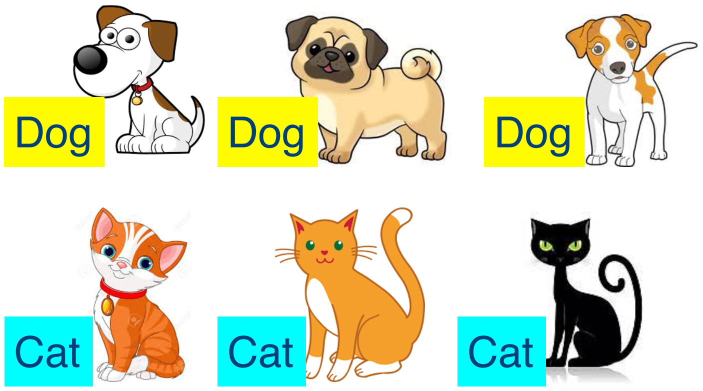
Predictive learning

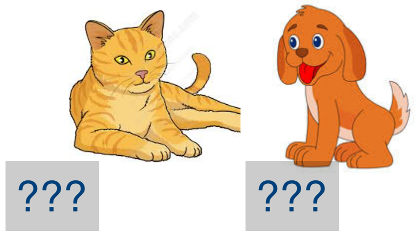
- A very large part of machine learning focuses on predictive learning
- The prediction task, in general:
  - Given: a description of some instance
  - Predict: some property of interest (the “target”)
- Examples:
  - classify emails as spam / non-spam
  - classify fish as salmon / bass
  - forecast tomorrow’s weather based on today’s measurements
- How? By analogy to cases seen before

Training & prediction sets

- **Training set**: a set of **examples**, instance descriptions that include the target property (a.k.a. **labeled** instances)
- **Prediction set**: a set of instance descriptions that do *not* include the target property ("unlabeled" instances)

- Prediction task : predict the labels of the unlabeled instances

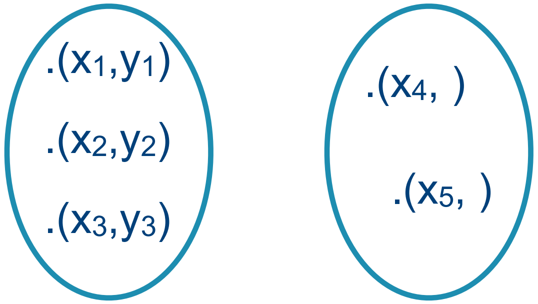




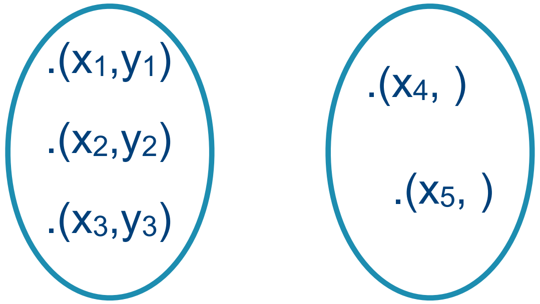
Inductive vs. transductive learning

We can consider as outcome of the learning process, either

- the predictions themselves: **transductive learning**
- or: a function that can predict the label of any unlabeled instance: **inductive learning**



Transduction: outcome=*predictions*

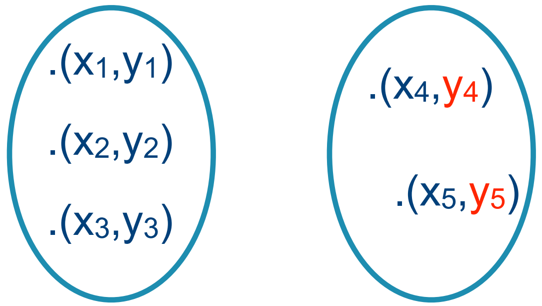


Induction: outcome = *function for making predictions*

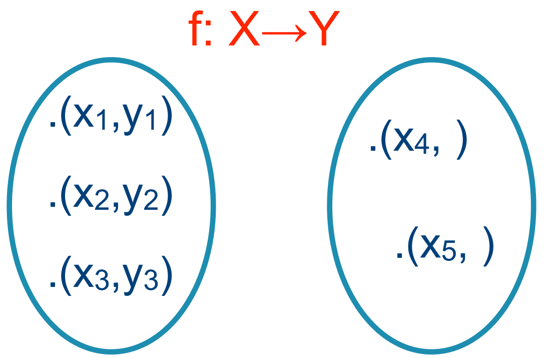
Inductive vs. transductive learning

We can consider as outcome of the learning process, either

- the predictions themselves: **transductive learning**
- or: a function that can predict the label of any unlabeled instance: **inductive learning**



Transduction: outcome=*predictions*

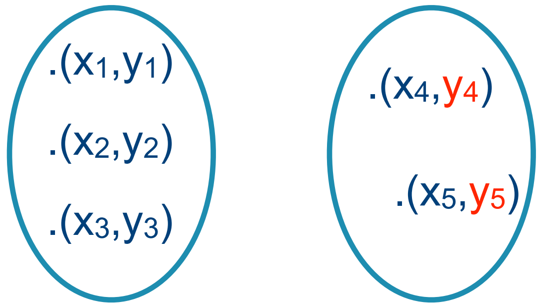


Induction: outcome = *function for making predictions*

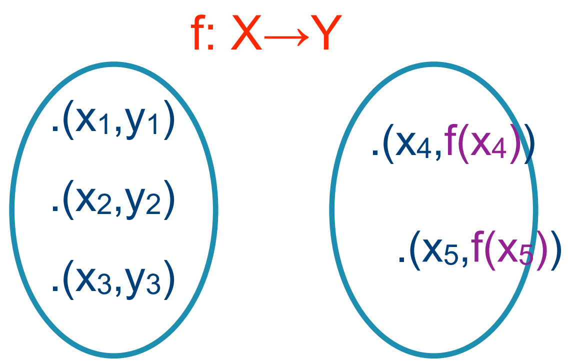
Inductive vs. transductive learning

We can consider as outcome of the learning process, either

- the predictions themselves: **transductive learning**
- or: a function that can predict the label of any unlabeled instance: **inductive learning**



Transduction: outcome=*predictions*



Induction: outcome = *function for making predictions*

Interpretable vs. black-box

The **predictive function** or **model** learned from the data may be represented in a format that we can easily interpret, or not

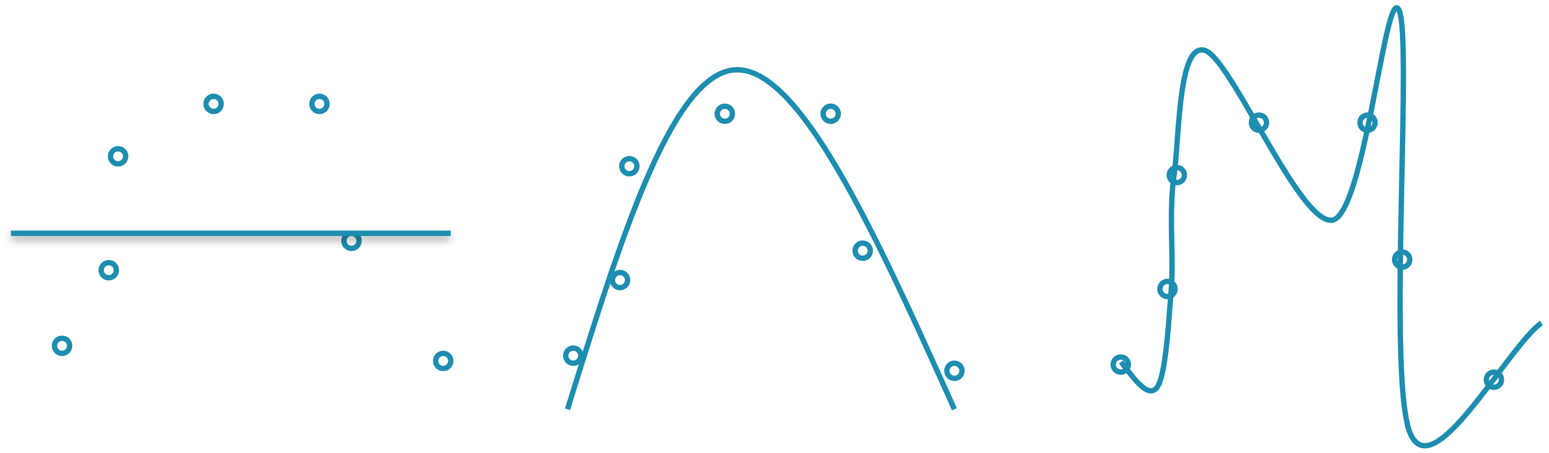
Non-interpretable models are also called **black-box** models

In some cases, it is crucial that predictions can be explained (e.g.: bank deciding whether to give you a loan)

Note difference between explaining a *model* and explaining a *prediction*

Overfitting and underfitting

- “Occam’s razor”: among equally accurate models, choose the simpler one
- Trade-off: explain data vs. simplicity
- Both overfitting and underfitting are harmful



Levels of supervision

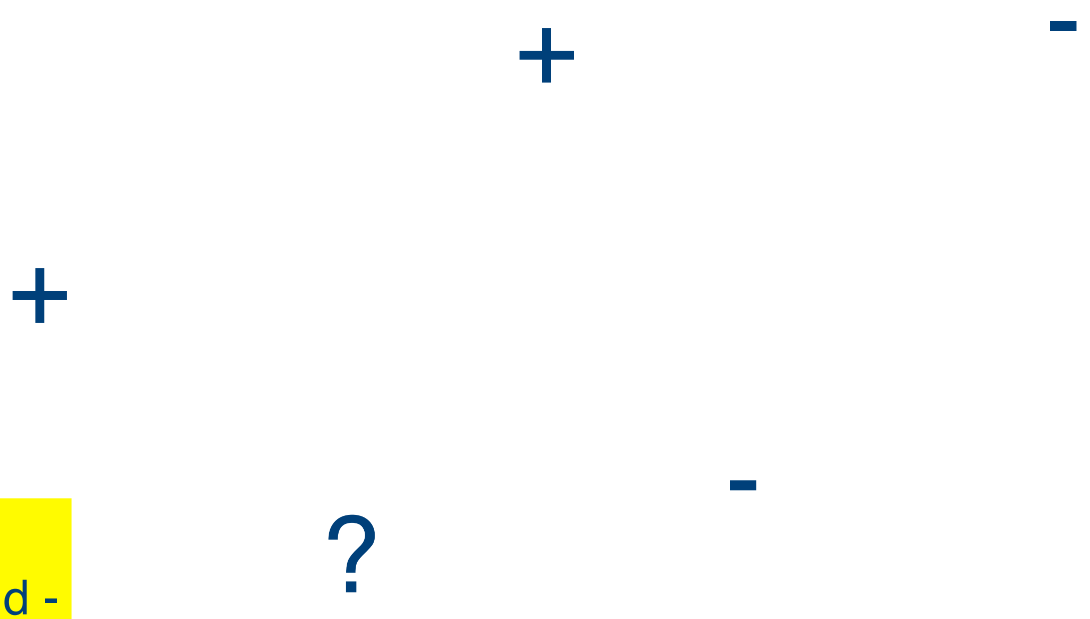
- **Supervised learning**: learning a (predictive) model from *labeled* instances (as in cats & dogs example)

- **Unsupervised learning**: learning a model from *unlabeled* instances
  - such models are usually not directly predictive (without *any* information on what to predict, how could you learn from that?)
  - still useful indirectly, or for non-predictive tasks: see later

- **Semi-supervised learning**: learn a predictive model from *a few labeled* and *many unlabeled* examples

Semi-supervised learning

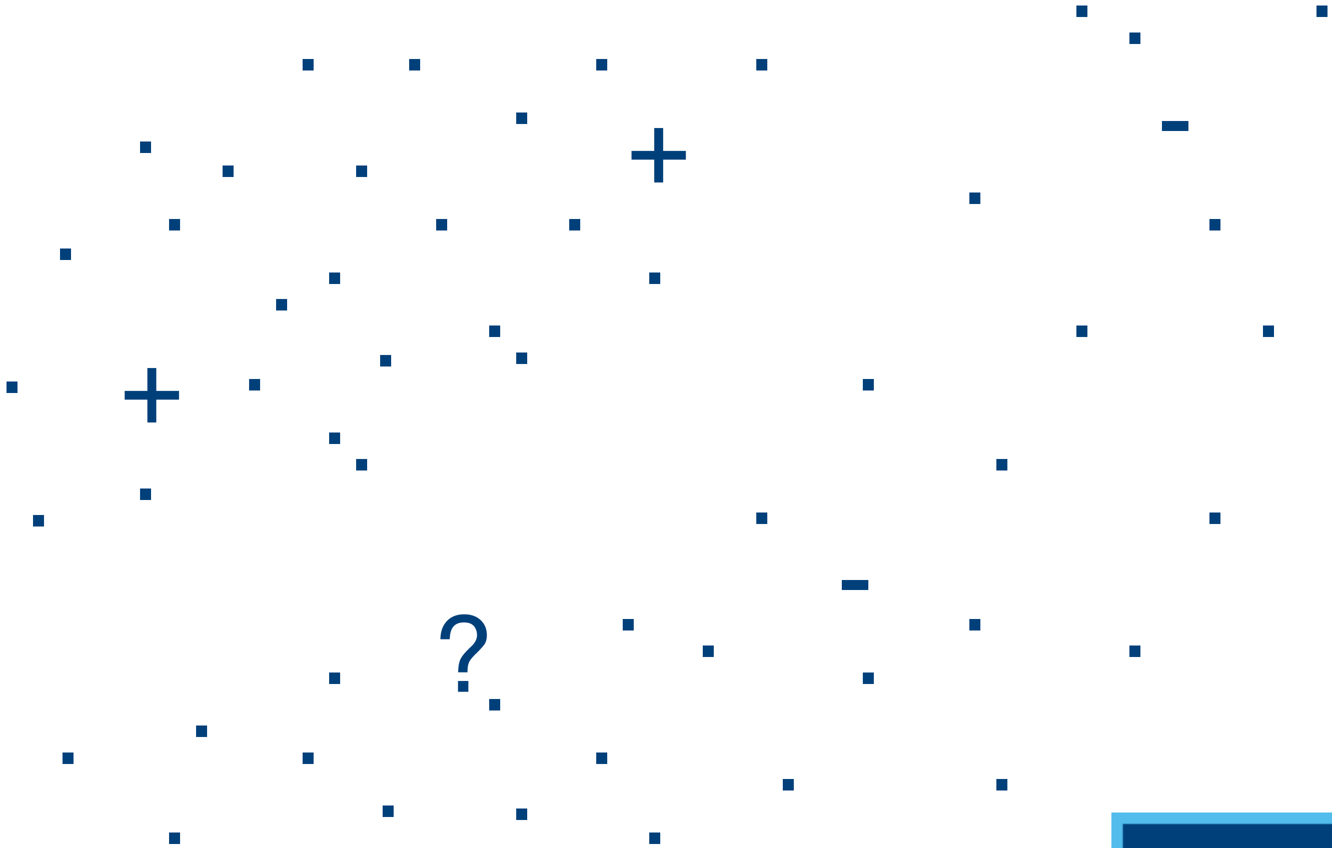
- How can unlabeled examples help learn a better model?



This illustration:
- 2 classes, called + and -
- Representing instances in a 2-dimensional space

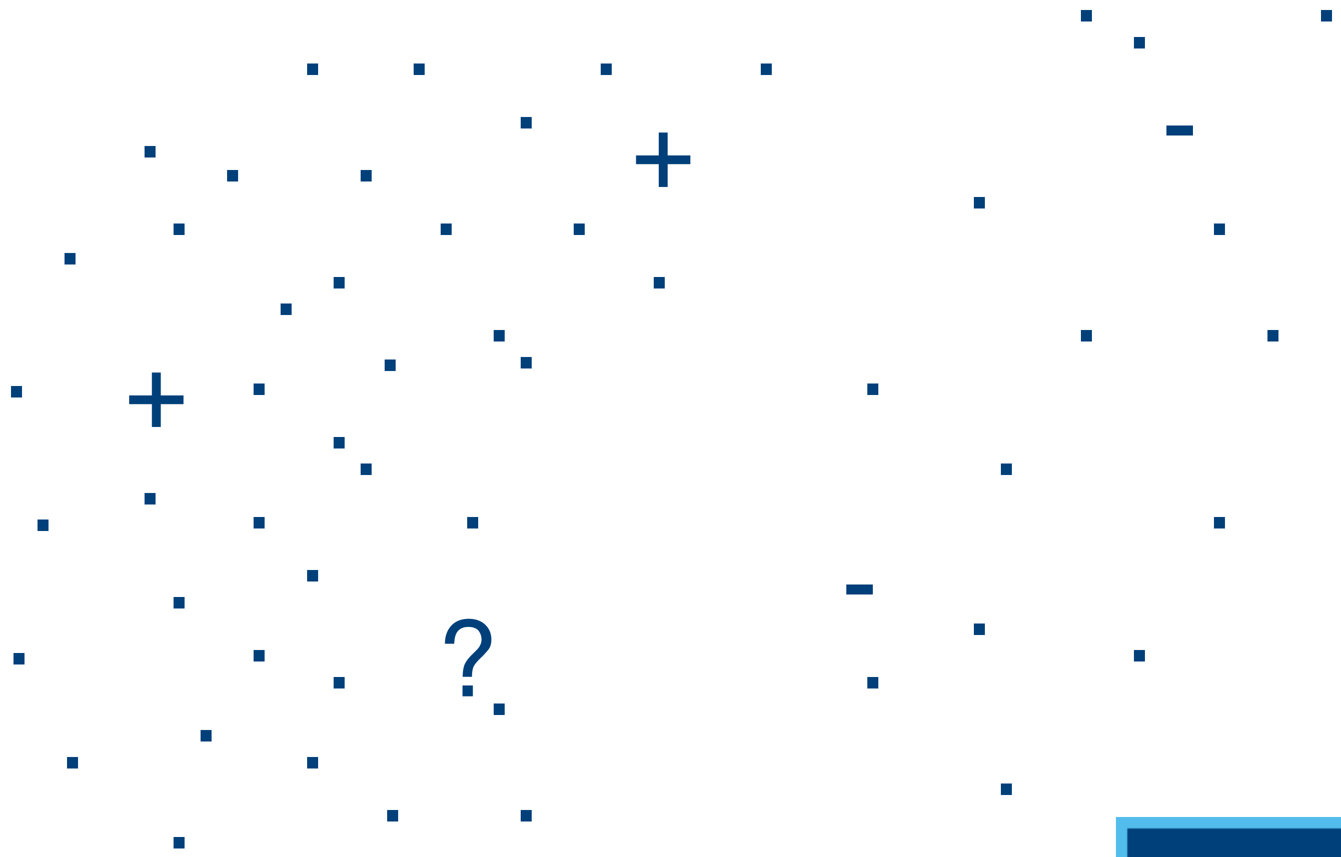
Semi-supervised learning

- How can unlabeled examples help learn a better model?



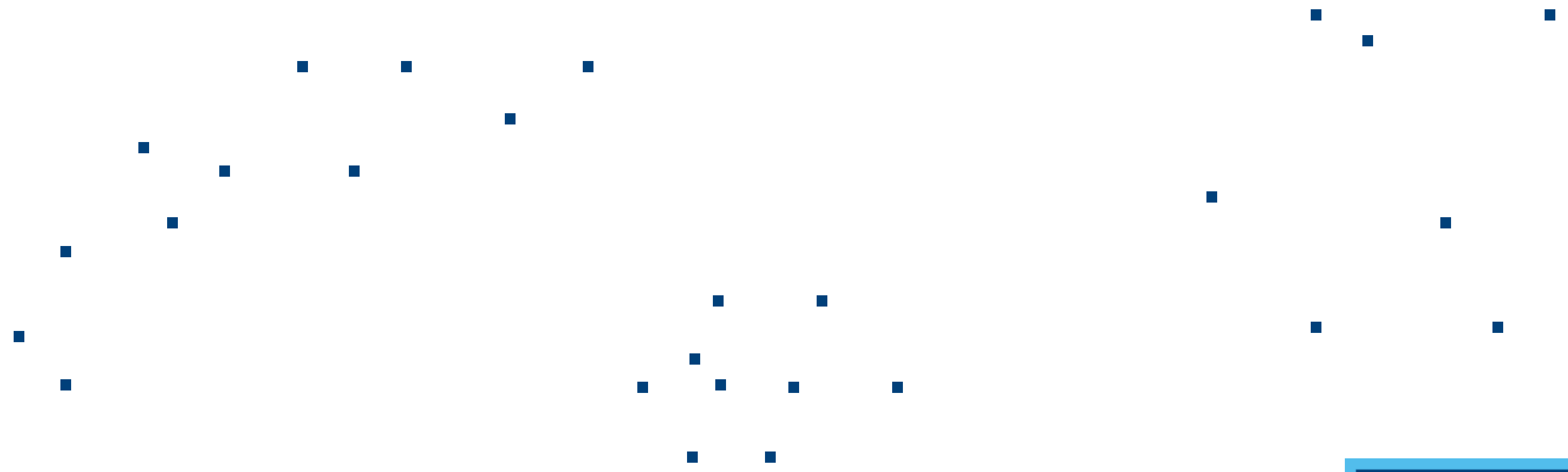
Semi-supervised learning

- How can unlabeled examples help learn a better model?



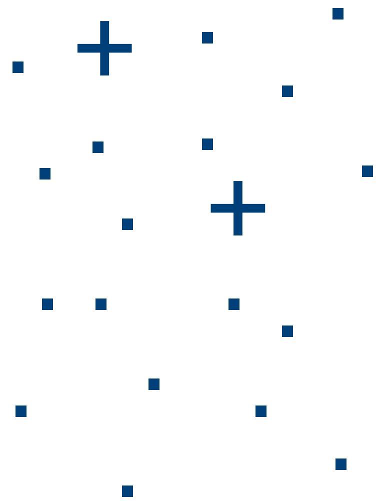
Unsupervised learning

- Can you see three classes here?
- Even though we don't know the *names* of the classes, we still see some structure (*clusters*) that we could use to predict which class a new instance belongs to
- Identifying this structure is called *clustering*
- From a predictive point of view, this is unsupervised learning



PU-learning

- PU-learning is a special case of semi-supervised learning
- PU stands for "positive / unlabeled"
- All the labeled examples belong to one class (called the "positive" class)



Learning the meaning of "kicking the ball" requires PU-learning because: When Mike kicks the ball, the sentence *may* mention this, or not. When Mike does not kick the ball, it is never mentioned that he does not.

"Mike is kicking the ball"

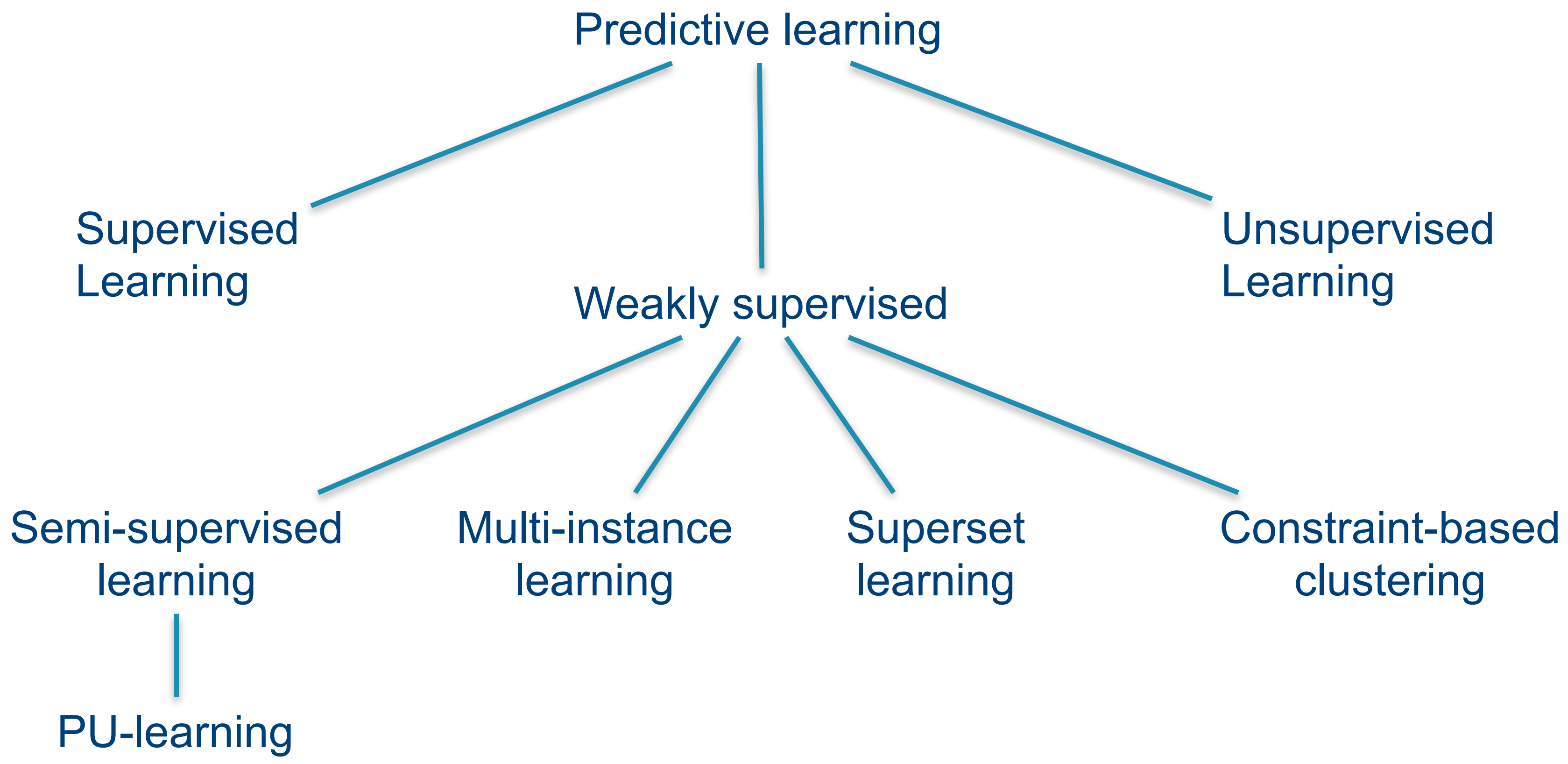
Weakly supervised learning

- Weakly supervised learning is a generalized form of semi-supervised learning
- Semi-supervised: for a single instance, we either know its label or we do not
- Weakly supervised: we may have partial information about a label
  - e.g., it is certainly a member of a given set (= **superset learning**)
  - e.g., at least one instance among a given set of instances has the label, but we do not know which one (= **multi-instance learning**)
  - e.g., we know two instances have the same label, but we don't know which one it is (= **constraint-based clustering**)
  - …

"This is either a Ferrari or a Lamborghini"

"There's a Lamborghini in this picture"

Relationship between different supervision settings



Reinforcement learning

- Sometimes viewed as another form of supervision, but it's different in more ways than just that
- The setting:
  - an agent can be in any of a number of states
  - it can take actions that change its state and may give some reward
  - it wants to maximize its rewards in the long run
  - How should it behave?
- Formally, given a state space S, transition function $\delta : S \times A \to S$, reward function $r : S \times A \to \mathbb{R}$, and discount factor $\gamma \in [0,1]$, find a policy that maximizes $\sum_{i=0}^{\infty} \gamma^i r(s_i, \pi(s_i))$

  where $s_{i+1} = \delta(s_i, \pi(s_i))$
- δ and r are initially unknown to the agent
- The agent generates its own training data (through exploration)

Illustration: Q-learning

- Agent updates a function $Q(s,a)$ that ultimately converges to $r(s,a) + \sum_{i=1}^{\infty} \gamma^i r(s_i, \pi^*(s_i))$ where π^* is the optimal policy
- … so that choosing the a that maximizes $Q(s,a)$ is by definition optimal (hence, defines π^*)

```
1: Initialise $Q$ randomly;
2: Initialize $s$;
3: while (true) do
4:      Choose an action $a$;
5:      Execute $a$ and observe $r$ and $s'$;
6:      $Q(s,a) := r + \gamma \max_{a'} Q(s',a')$;
7:      $s := s'$;
8:end while
```

$$Q(s,a) \leftarrow 0.0 + 0.9 \times 1.0 \quad Q(s',a')$$

Up

0	0	0	0
1	0	0	0
0.90	0	0	0
0.81	0	0	0

Down

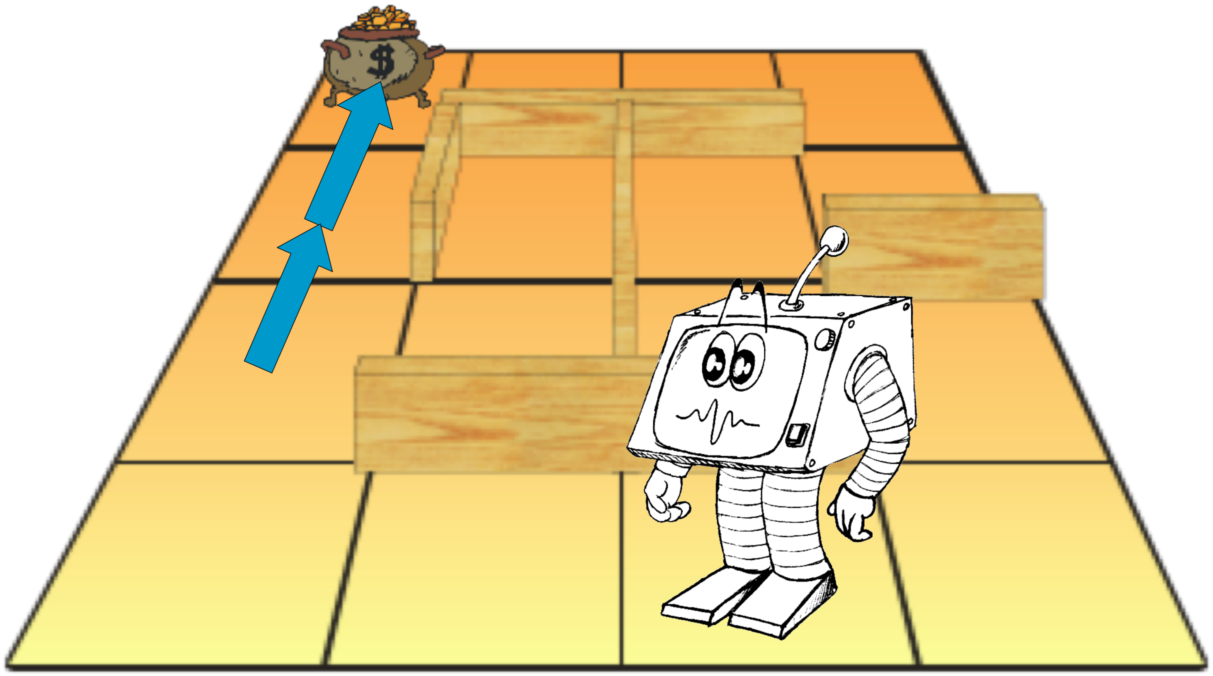
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

Left

0	0	0	0
0	0	0	0
0	0	0	0
0	0.73	0.66	0

Right

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0



Some comments

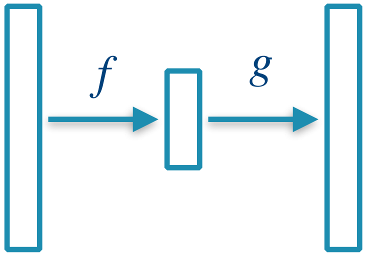
- In the illustration, the agent updated a table-based representation of $Q(s, a)$
- We can use that table as feedback to learn what *kind* of actions work well in what *kind* of states. That is, we learn a model of the Q function. This is **model-based reinforcement learning**

- We assumed a deterministic environment, but RL is often formulated in a probabilistic environment: Markov Decision Processes (MDP)

- We here assumed random exploration, but more clever versions exist (gradually increasing exploration of more promising parts of the state-action space)

Encoding data

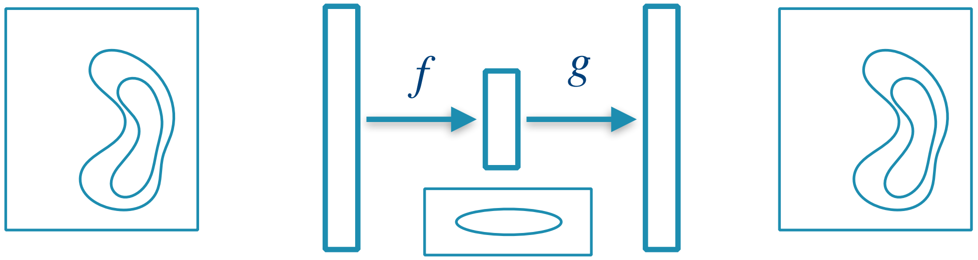
- An unsupervised learning task: learn functions $f : \mathscr{X} \to \mathbb{R}^k$ and $g : \mathbb{R} \to \mathscr{X}$ such that $g \circ f$ is the identity function
- When $\mathscr{X} = \mathbb{R}^m$, typically $k < < m$: dimensionality reduction
- When $\mathscr{X}$ is not a vector space, f is called an embedding
- f is called the encoder and g the decoder
- Encoding can be useful for understanding the structure of the input space



KU LEUVEN

Probabilistic encoders

- A “standard” encoder learns $f : \mathscr{X} \to \mathbb{R}^k$ and $g : \mathbb{R} \to \mathscr{X}$ such that $g \circ f$ is the identity function
- We can also consider *probabilistic* encoders, which are trained to reconstruct the input distribution over $\mathscr{X}$ but not individual cases
- That is: we learn $f : \mathscr{X} \to \mathbb{R}^k$ and $g : \mathbb{R} \to \mathscr{X}$ such that the distribution of $g \circ f(x)$ is equal to the distribution of x
- The distribution of $f(x)$ is called the latent distribution
- This is useful for training, e.g., generative models



Format of input data

Format of input data

- Input is often assumed to be a *set* of instances that are all described using the same variables (features, attributes)
  - The data are "i.i.d.": "independent and identically distributed"
  - The training set can be seen as a random sample from one distribution
  - The training set can be shown as a table (instances x variables) : *tabular data*
  - This is also called the **standard setting**

- There are other formats: instances can be
  - nodes in a graph
  - whole graphs
  - elements of a sequence
  - …

Format of input data: *tabular*

Training set

Sepal length	*Sepal width*	*Petal length*	*Petal width*	*Class*
5.1	*3.5*	*1.4*	*0.2*	*Setosa*
4.9	*3.0*	*1.4*	*0.2*	*Setosa*
7.0	*3.2*	*4.7*	*1.4*	*Versicolor*
6.3	*3.3*	*6.0*	*2.5*	*Virginica*

Prediction set

Sepal length	*Sepal width*	*Petal length*	*Petal width*	*Class*
4.8	*3.2*	*1.3*	*0.3*	*?*
7.1	*3.3*	*5.2*	*1.7*	*?*

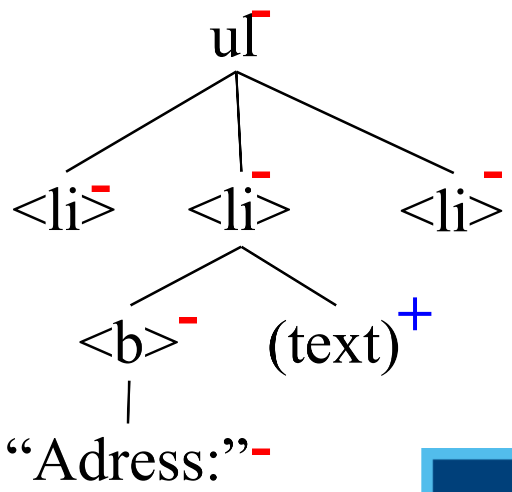
Format of input data: *sequences*

- Learning from sequences:
  - 1 prediction per sequence?
  - 1 prediction per element?
- 1 element in sequence can be …
  - A number (e.g., time series)
  - A symbol (e.g., strings)
  - A tuple
  - A more complex structure

```
abababab: +
aabbaabb: -
```

Format of input data: *trees*

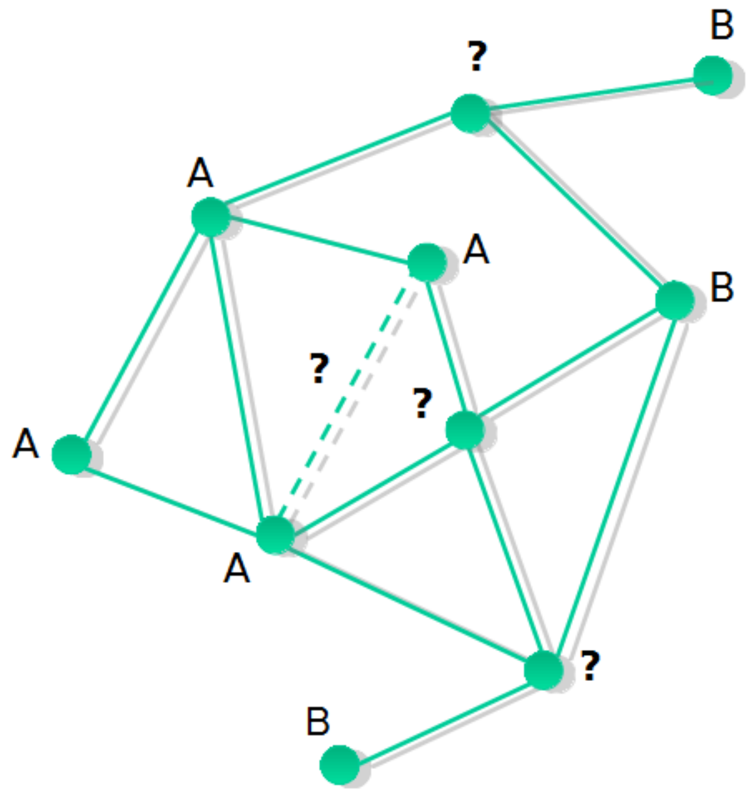
- 1 prediction per tree / per node in the tree
- Nodes can be …
  - Unlabeled
  - Labeled with symbols (e.g., HTML/XML structures)
  - …



E.g.: this tree indicates as “positive” a text field preceded by **Address:** inside a list (**<li>**) context

Format of input data: *graph*

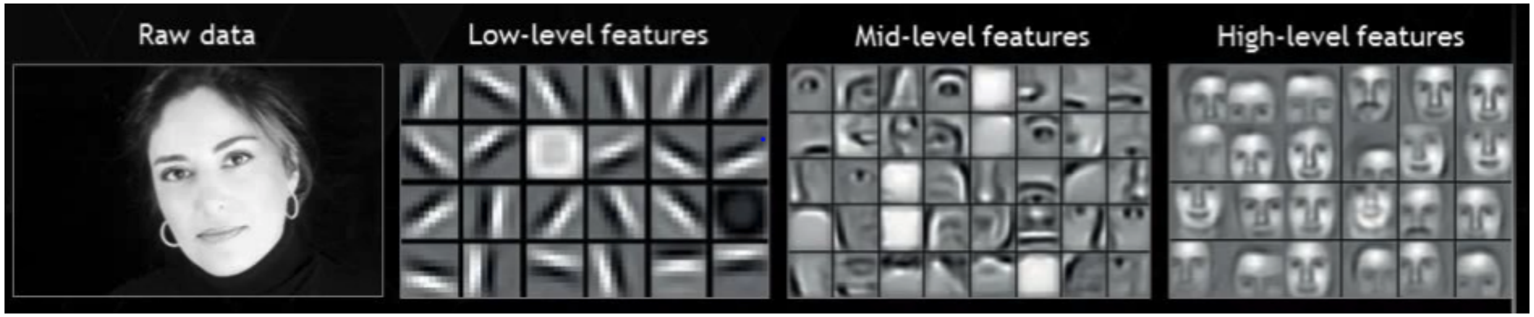
- Example: Social network
- Target value known for some nodes, not for others
- Predict node label
- Predict edge
- Predict edge label
- …
- Use *network structure* for these predictions



KU LEUVEN

Format of input data: *raw data*

- "Raw" data are in a format that seems simple (e.g., a vector of numbers), but components ≠ meaningful features
- Example: photo (vector of pixels)
- Raw data often need to be processed in a non-trivial way to obtain meaningful features; on the basis of these features, a function can be learned
- This is what deep learning excels at



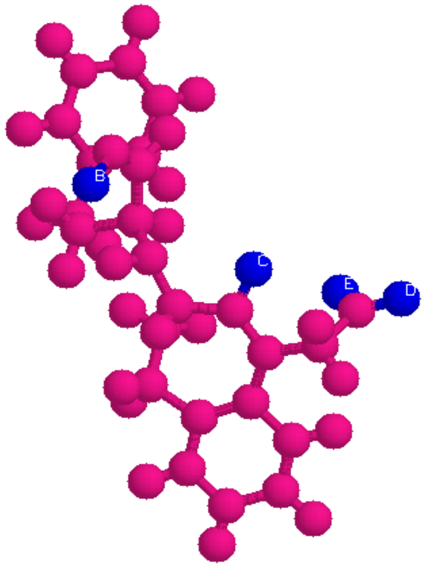
(Image: Nielsen, 2017, *Neural networks and deep learning*)

Format of input data: *knowledge*

- “Knowledge” can consist of facts, rules, definitions, ….
- We can represent knowledge about some domain in a *knowledge representation* language (such languages are often based on logic)

```
atm(m1,a1,o,2,3.43,-3.11,0.04).
atm(m1,a2,c,2,6.03,-1.77,0.67).
...
bond(m1,a2,a3,2).
bond(m1,a5,a6,1).
bond(m1,a6,a7,du).
...
```

```
...
hacc(M,A):- atm(M,A,o,2,_,_,_).
hacc(M,A):- atm(M,A,o,3,_,_,_).
hacc(M,A):- atm(M,A,s,2,_,_,_).
hacc(M,A):- atm(M,A,n,ar,_,_,_).
zincsite(M,A):-
      atm(M,A,du,_,_,_,_).
hdonor(M,A) :-
                  atm(M,A,h,_,_,_,_),
                  not(carbon_bond(M,A)), !.
...
```



Data preprocessing

- Data may not be in a format that your learner can handle
- **Data wrangling**: bring it into the right format
- Even if it’s in a format you learner can handle (e.g., tabular), the features it contains may not be very informative, or there may be very few relevant features among many irrelevant ones.
  - E.g.: individual pixels in an image are usually not very informative
- **Feature selection**: select among many input features the most informative ones
- **Feature construction**: construct new features, derived from the given ones

What learning method to use?

- Which learners are suitable for your problem, depends strongly (but not solely!) on the structure of the input data
- Many learners use the *standard* format
  - A set of instances, where each instance is described by a fixed set of *attributes* (a.k.a. *features*, *variables*)
  - also called *attribute-value* format or *tabular* format
- Other learners handle input knowledge with a more complex structure (sequences, graphs, …) or raw data
- Different learners have a different inductive bias

Output formats, methods (overview)

Output formats

- The output of a learning system is a *model*
- Many different types of model exist
- The learning algorithm or method is strongly linked to the type of model
- High-level overviews of machine learning methods often categorize them along this axis

Different views of the landscape

Domingos:
"five tribes"

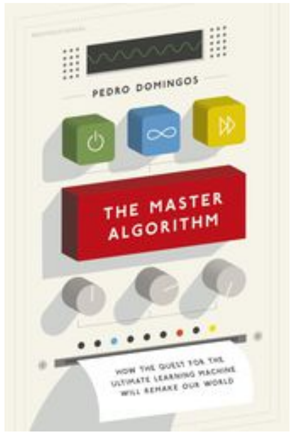
- Symbolists
- Connectionists
- Evolutionaries
- Bayesians
- Analogizers

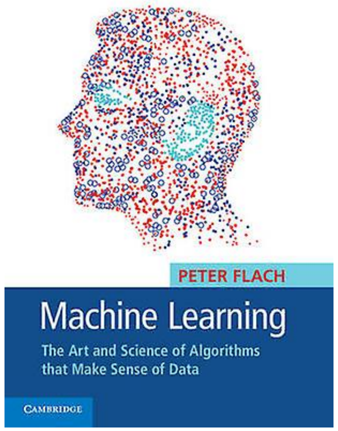
Flach:
"three types of models"

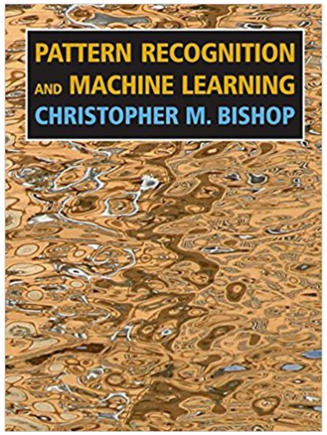
- Probabilistic
- Geometric
- Logical

Bishop:
"Ultimately, everything is Bayesian"

- Bayes


PEDRO DOMINGOS
THE MASTER ALGORITHM
HOW THE QUEST FOR THE ULTIMATE LEARNING MACHINE WILL REMAKE OUR WORLD

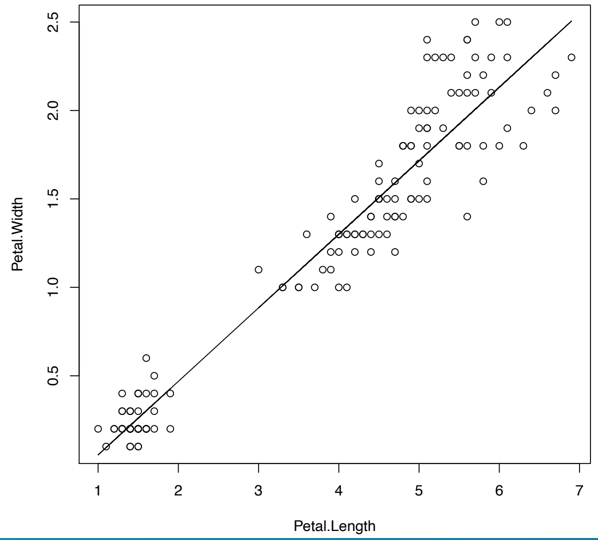




KU LEUVEN

Parametrized functions

- Typically, a certain format for the functions is provided; e.g.: linear functions of the inputs
- Within this set, we look for the parameter values that best fit the data
- Standard example: *linear regression*



$$y = ax + b$$

*width = 0.416*length - 0.363*

Conjunctive concepts

- A conjunctive concept is expressed as a set of conditions, all of which must be true
- “x has class C if and only if <condition1> *and* <condition2> *and* … *and* <condition k>”
- E.g.: accept application for mortgage if and only if :
  salary ≥ 3 * monthly payback ***and*** no other mortgage running

Rule sets

- A rule set is a set of rules of the form "if … then …" or "if … then … else …"
- Example: definition of leap years

Examples of leap years:
1900, 1992, 2004, …

Examples of non-leap years:
1993, 2000, 2011, 2018, …

Input

Output

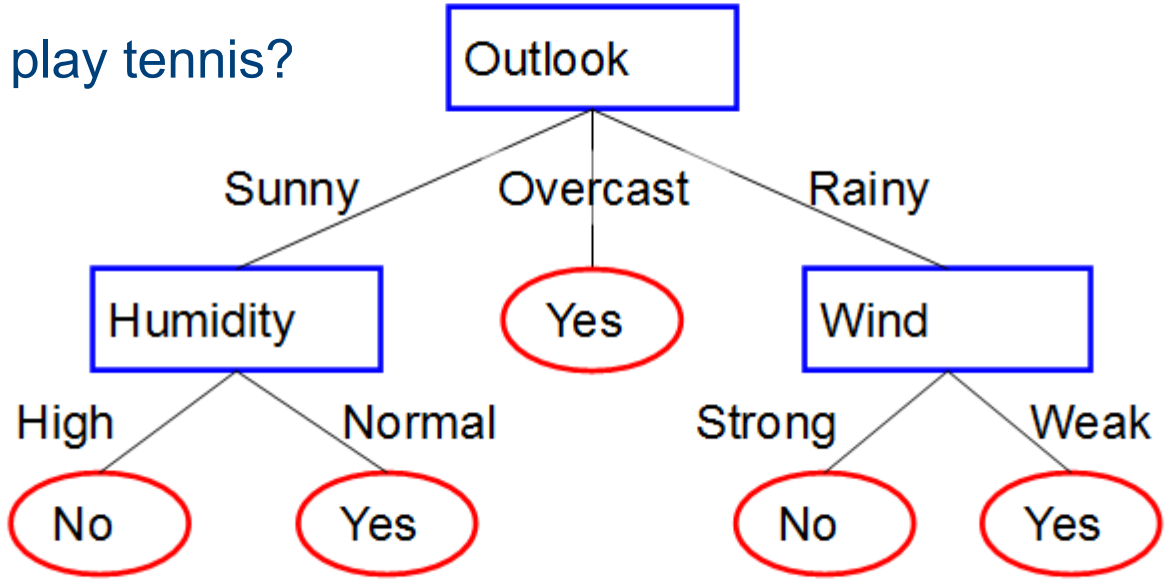
If year is a multiple of 400 **then** leap
else if year is a multiple of 100 **then** not leap
else if year is multiple of 4 **then** leap
else not leap

KU LEUVEN

Decision trees

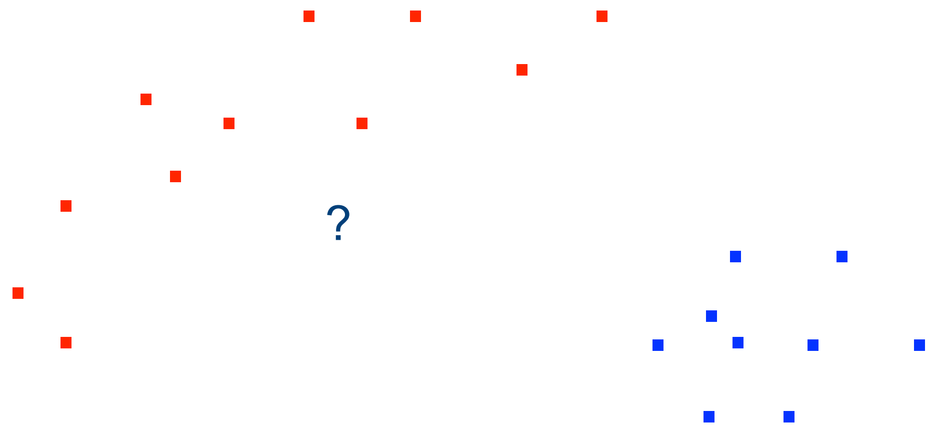
- A decision tree represents a stepwise procedure to arrive at some decision

Is today a good day to play tennis?



Instance-based learning (a.k.a. “nearest neighbor methods”)

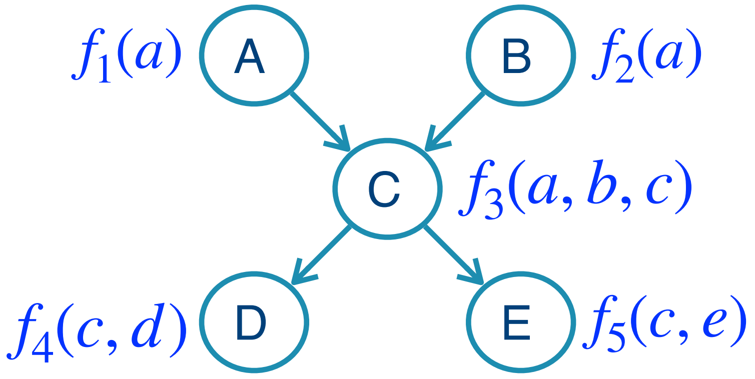
- The “model” is simply the data set itself
- Predictions for new cases are made by comparing it to earlier observed cases
- If it’s similar for observed features, it’s probably also similar for unobserved (to be predicted) features



Probabilistic graphical models

- A PGM represents a (high-dimensional) joint distribution over multiple variables as a product of (low-dimensional) factors
- Different type of PGMs: Bayesian networks, Markov networks, factor graphs, …

Example: Bayesian network



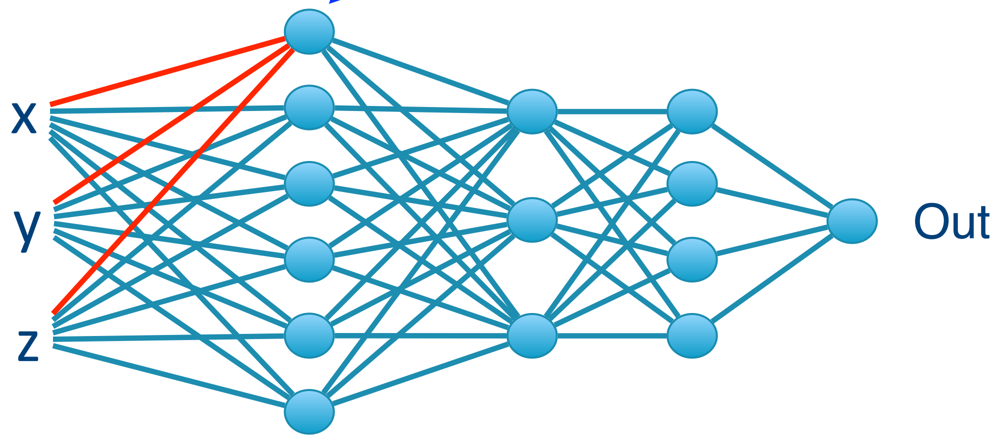
$$f(a,b,c,d,e) = f_1(a) \cdot f_2(b) \cdot f_3(a,b,c) \cdot f_4(c,d) \cdot f_5(c,e)$$

$$P(A,B,C,D,E) = P(A) \cdot P(B) \cdot P(C \mid A,B) \cdot P(D \mid C) \cdot P(E \mid C)$$

Neural networks

- A neural network is a complex structure of neurons, each of which aggregate multiple input signals into a single output signal

$$h_{11} = f(a_{11}x + b_{11}y + c_{11}z)$$

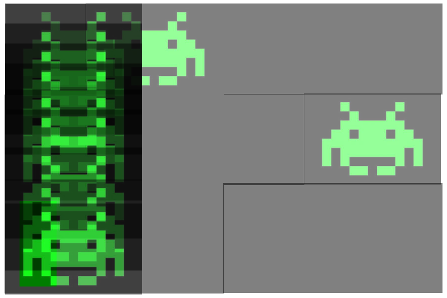


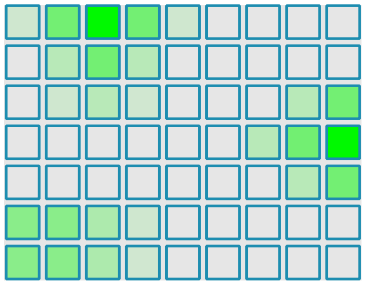
Advanced NN architectures

- The network we just saw just stacked a few layers of neurons on top of each other
- Much more advanced architectures exist, which serve a specific purpose
  - e.g. convolutional networks scan a picture for patterns (the scanning is hard-wired, the patterns to look for are learned)
  - LSTMs process sequences of data sequentially, and have memory cells that allow them to store info for re-use later on (the memory is hard-wired, *what* to remember *when* is learned)
  - Transformer models process sets/sequences by enriching their elements with context info ("attention") and processing everything simultaneously, not sequentially

Illustration: convolutional step

- In this example, the "map" essentially lights up in those positions where a sub-image similar to the kernel was found
- Note: convolution layers are translation-invariant (sub-image is found regardless of position in the picture) but not rotation- or scale-invariant: need to train on rotated / scaled versions separately





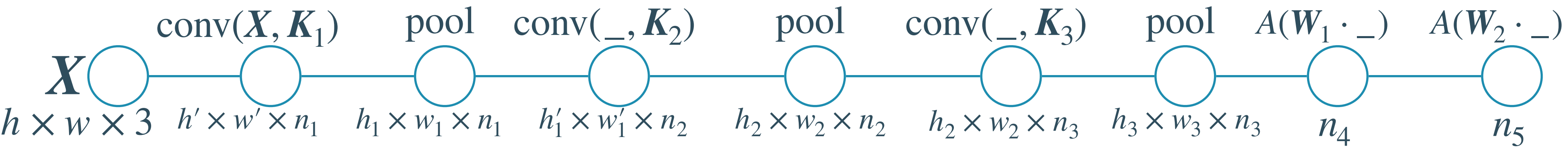
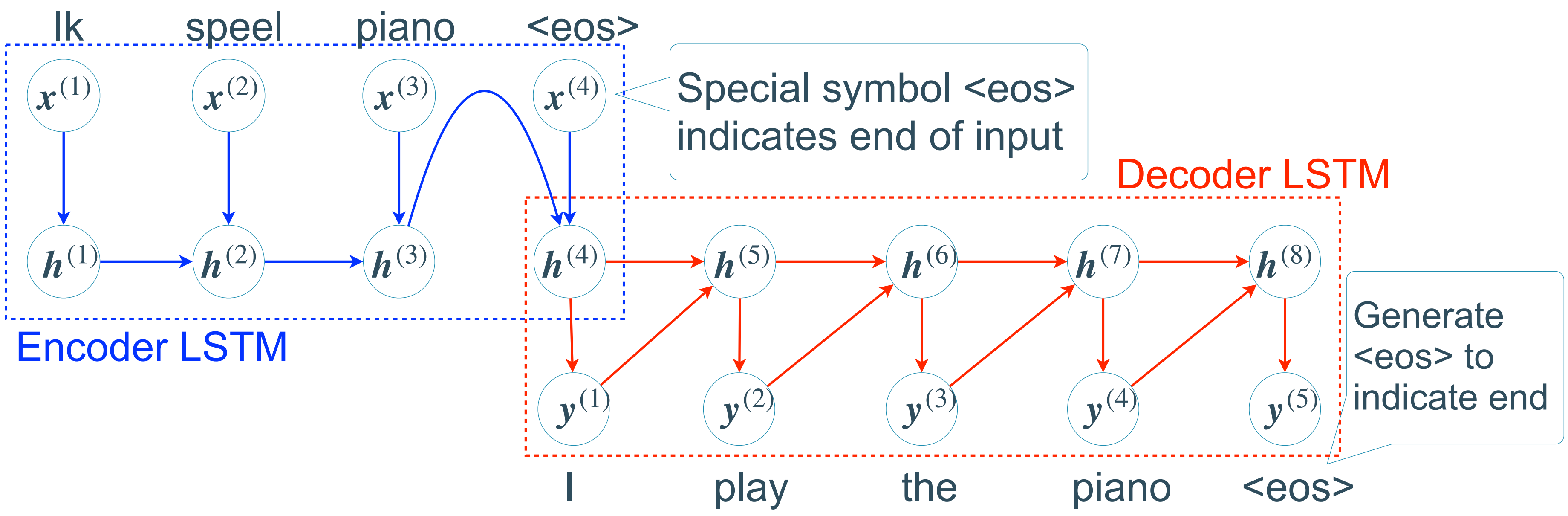

X
$h \times w \times 3$
$\text{conv}(\boldsymbol{X}, \boldsymbol{K}_1)$
$h' \times w' \times n_1$
pool
$h_1 \times w_1 \times n_1$
$\text{conv}(\_, \boldsymbol{K}_2)$
$h'_1 \times w'_1 \times n_2$
pool
$h_2 \times w_2 \times n_2$
$\text{conv}(\_, \boldsymbol{K}_3)$
$h_2 \times w_2 \times n_3$
pool
$h_3 \times w_3 \times n_3$
$A(\boldsymbol{W}_1 \cdot \_)$
n_4
$A(\boldsymbol{W}_2 \cdot \_)$
n_5

Illustration: Sutskever et al.’s “Sequence to sequence learning with neural networks” (2014)

- Example instance: *ik speel piano* <eos> *I play the piano* <eos>



Ik
speel
piano
<eos>
Special symbol <eos> indicates end of input
Decoder LSTM
Encoder LSTM
Generate <eos> to indicate end
I
play
the
piano
<eos>

$$\begin{tikzcd}
x^{(1)} \arrow[d] & x^{(2)} \arrow[d] & x^{(3)} \arrow[d] \arrow[rdd, bend left] & x^{(4)} \arrow[d] & & & & \\
h^{(1)} \arrow[r] & h^{(2)} \arrow[r] & h^{(3)} & h^{(4)} \arrow[r] \arrow[d] & h^{(5)} \arrow[r] \arrow[d] & h^{(6)} \arrow[r] \arrow[d] & h^{(7)} \arrow[r] \arrow[d] & h^{(8)} \arrow[d] \\
& & & y^{(1)} \arrow[ru] & y^{(2)} \arrow[ru] & y^{(3)} \arrow[ru] & y^{(4)} \arrow[ru] & y^{(5)}
\end{tikzcd}$$

Illustration: Transformer

- Figure to the right (from Vaswani et al.) shows the architecture of the full transformer model
- Left = encoder, right = decoder
- The link between encoder & decoder is an attention layer that works similar to Bahdanau's attention
- **The input and output self-attention layers (with feedforward NN stacked on top) replace the RNNs**
- The decoder uses *masked* attention: it is not allowed to look at words that will be generated later
- The positional encoding added to words uses periodic functions (sin/cos) with various periods
- More details in Vaswani et al. (incl. code & appendices)

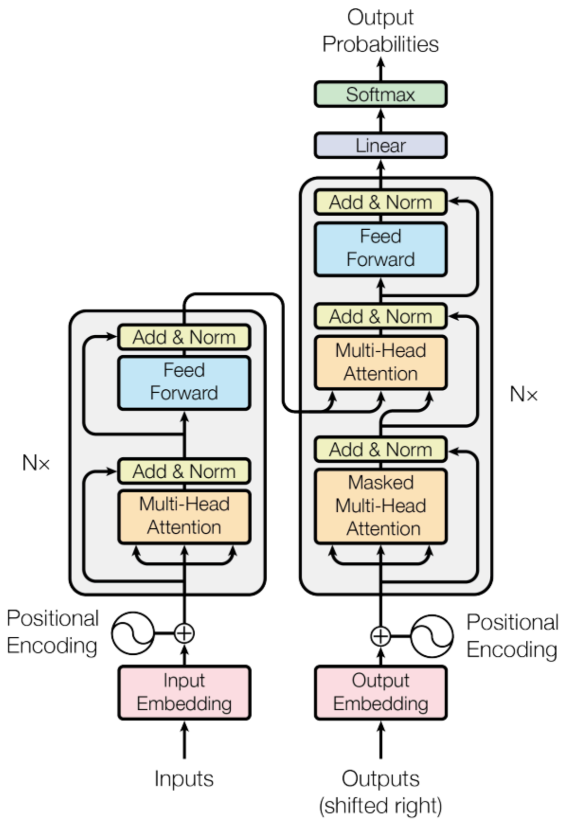


Figure 1: The Transformer - model architecture.

Illustration: BERT

- Input = a single sentence, or a pair of sentences
- Model is trained on multiple different pre-training tasks
  - “Masked LM”: mask some words, model must predict them
  - “Next sentence prediction”: labeled pairs of sentences (next, not-next)

Illustration from Devlin et al.

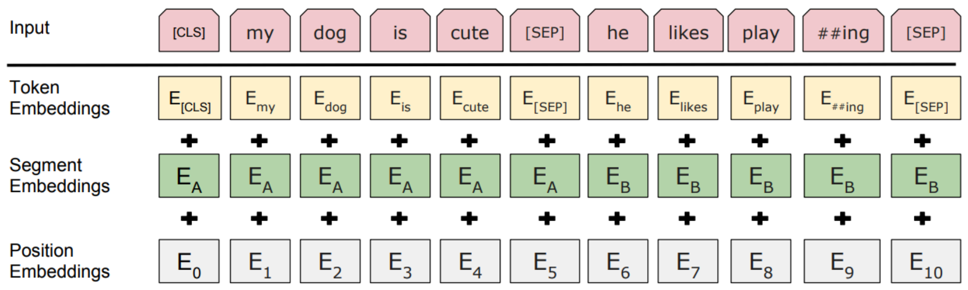
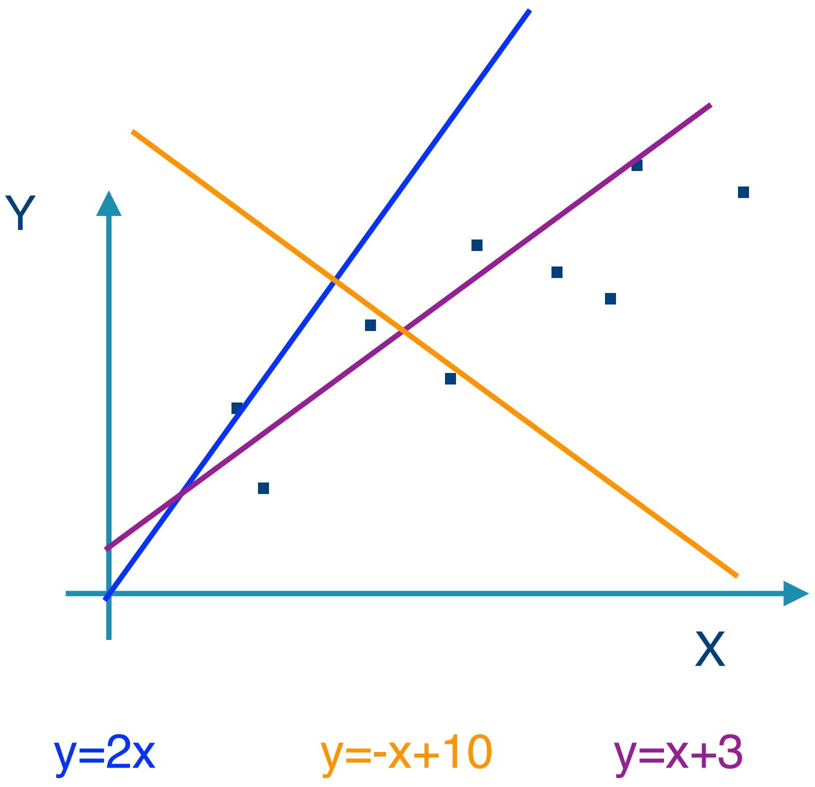

Input
[CLS]
my
dog
is
cute
[SEP]
he
likes
play
##ing
[SEP]
Token Embeddings
$E_{[CLS]}$ E_{my} E_{dog} E_{is} E_{cute} $E_{[SEP]}$ E_{he} E_{likes} E_{play} $E_{\#\#ing}$ $E_{[SEP]}$
Segment Embeddings
E_A E_A E_A E_A E_A E_A E_B E_B E_B E_B E_B
Position Embeddings
E_0 E_1 E_2 E_3 E_4 E_5 E_6 E_7 E_8 E_9 E_{10}

Figure 2: BERT input representation. The input embeddings are the sum of the token embeddings, the segmentation embeddings and the position embeddings.

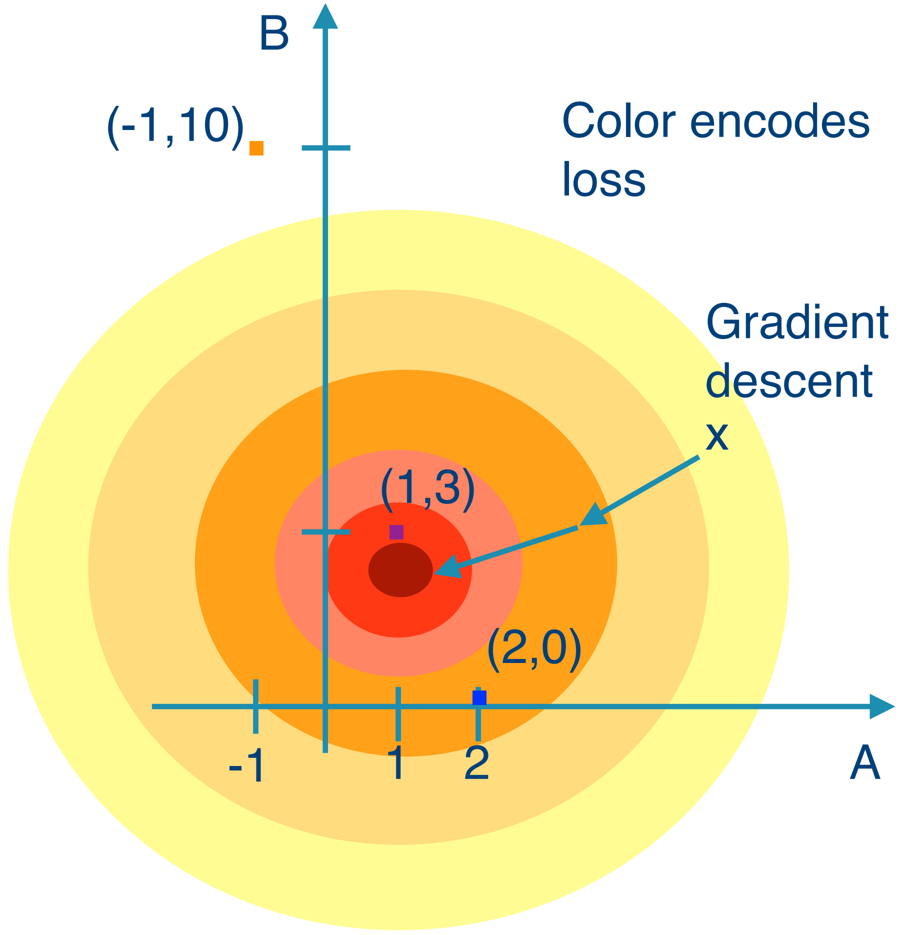
Search methods

- How do we find the most suitable model?
- Sometimes, there is a closed form solution (e.g., linear regression)
- If not, we typically need to search some *hypothesis space*
- Two very different types of spaces, each with their own search methods :
  - *Discrete spaces* (methods: hill-climbing, best-first, …)
  - *Continuous spaces* (methods: gradient descent, …)
- Typically:
  - Model structure not fixed in advanced => discrete
  - Fixed model structure, tune numerical parameters => continuous

Example: gradient descent in a continuous space



Input/output space



(a,b) represents y = ax + b

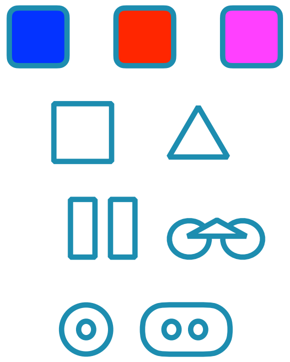
Parameter space

Example: Version Spaces

- The goal: identify a *conjunctive concept* from data
  - = a symbolic description of some concept, consisting of a conjunction of conditions
- More specific: given a hypothesis space H (all possible descriptions), return *all* concepts in H that are consistent with the data. This set is called the **Version Space**. When the version space is a singleton, the concept has been identified
- The algorithm called **Candidate Elimination** does this by exploiting a generality ordering over H, and returning only the most general and most specific hypotheses in H that are consistent with the data (the "borders" of H)
- This involves an exhaustive search in a *discrete* space

Candidate Elimination: illustration

- A company produces intelligent robots. Some robots misbehave. We suspect that one particular combination of features is the cause for this misbehavior.
- For ease of discussion, we here assume robots have four relevant characteristics:
  - Color : B R M
  - Body shape : S T
  - =Legs/wheels: L W
  - #“eyes” : 1 2



- Find the combination that misbehaves

Candidate Elimination: illustration

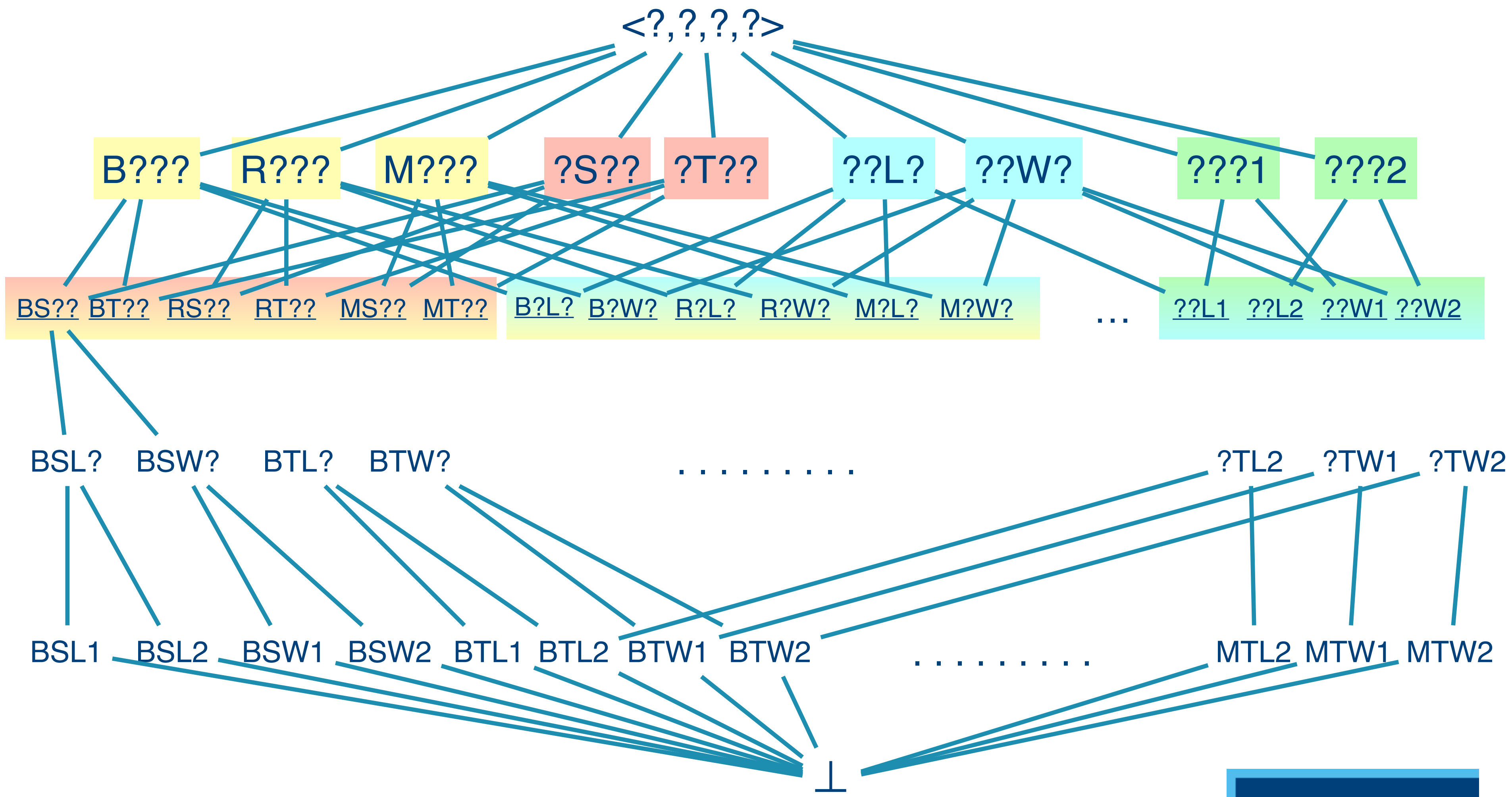
- We will represent a hypothesis as a tuple <color, body, legs, eyes> where color = B, R, M or ? (? means “any color”) etc.
- Hypothesis space: {B,R,M,?} x {S,T,?} x {L,W,?} x {1,2,?}
- Let S(h) be the set of robots characterized by a hypothesis h
- Hypothesis h_1 is more general than h_2 if and only if $S(h_2) \subseteq S(h_1)$

- Most general hypothesis is … censored

- Most specific hypothesis is … censored

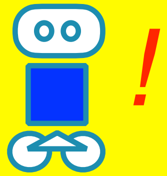
KU LEUVEN

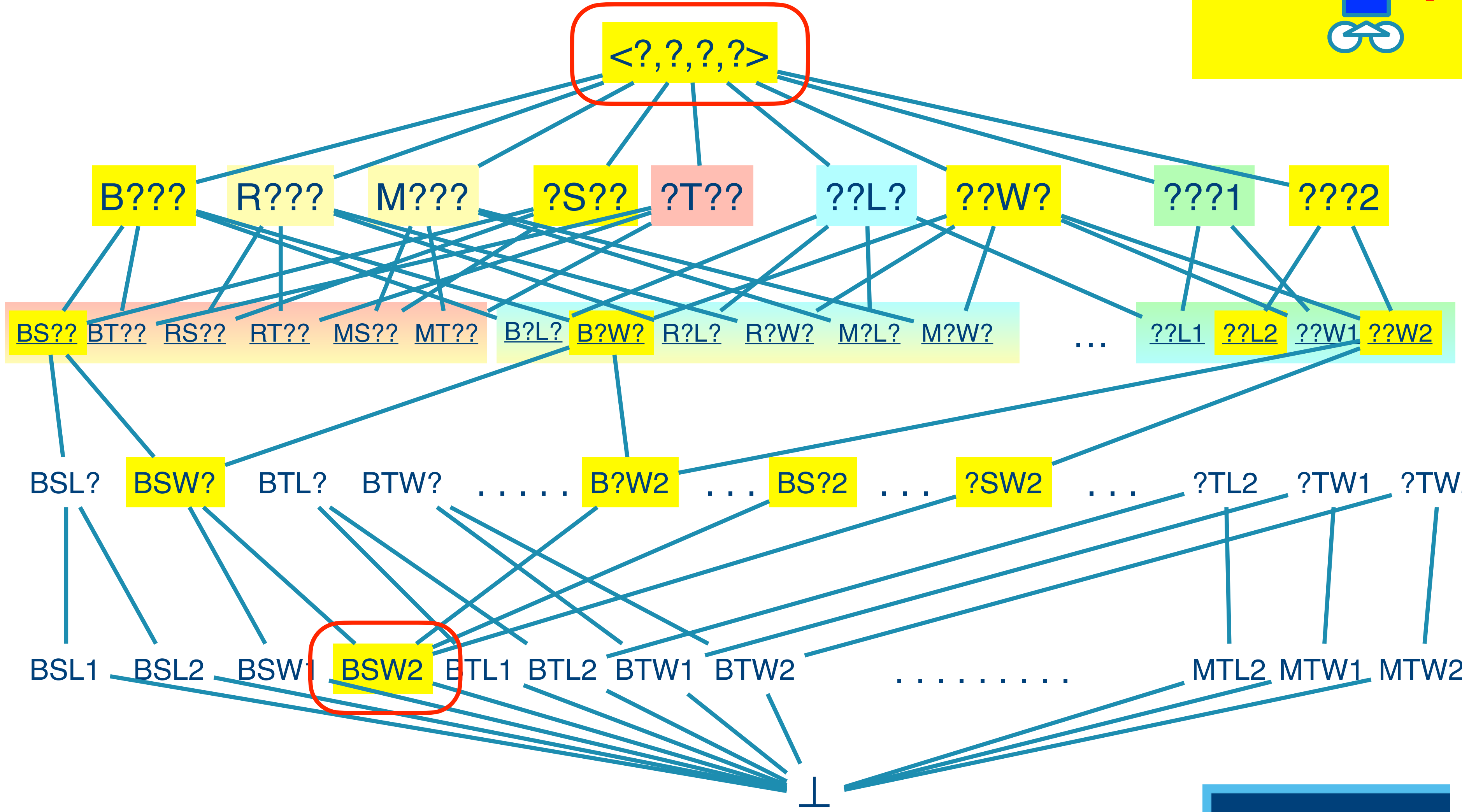
Search space is a *lattice*



Candidate Elimination

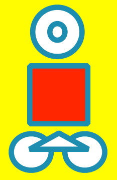
Observation 1:
Misbehaves

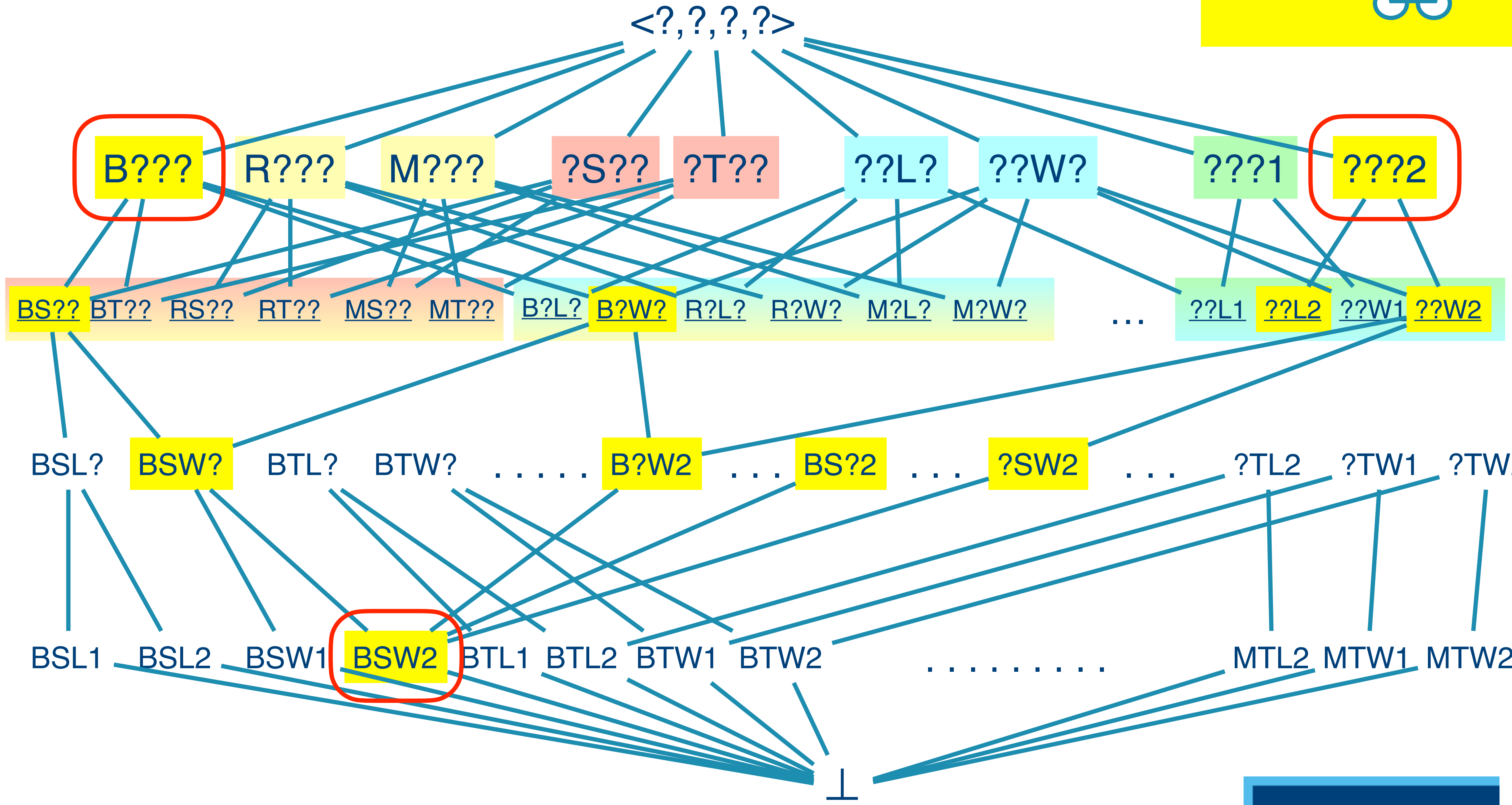




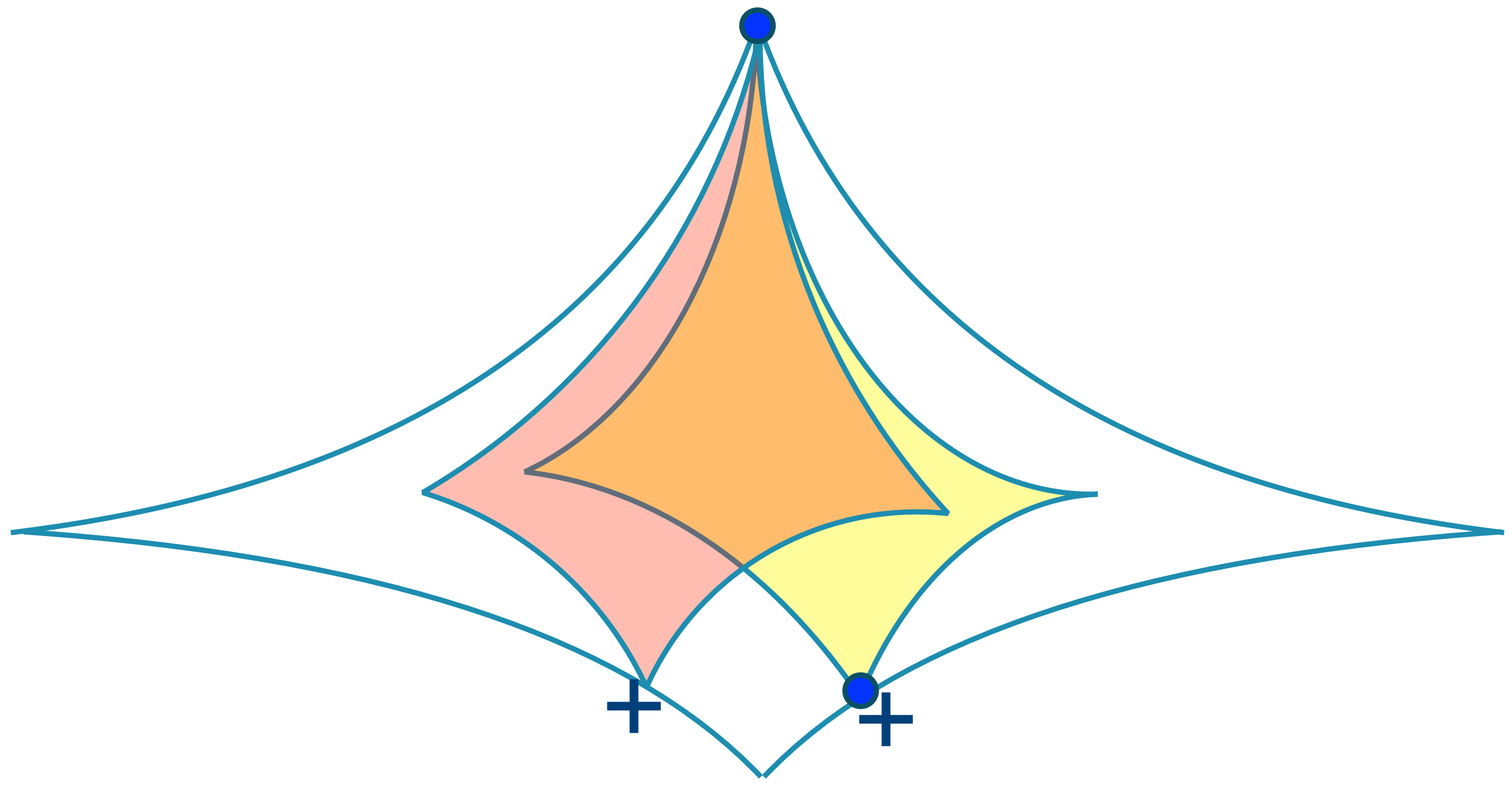
Candidate Elimination

Observation 2:
Does not misbehave

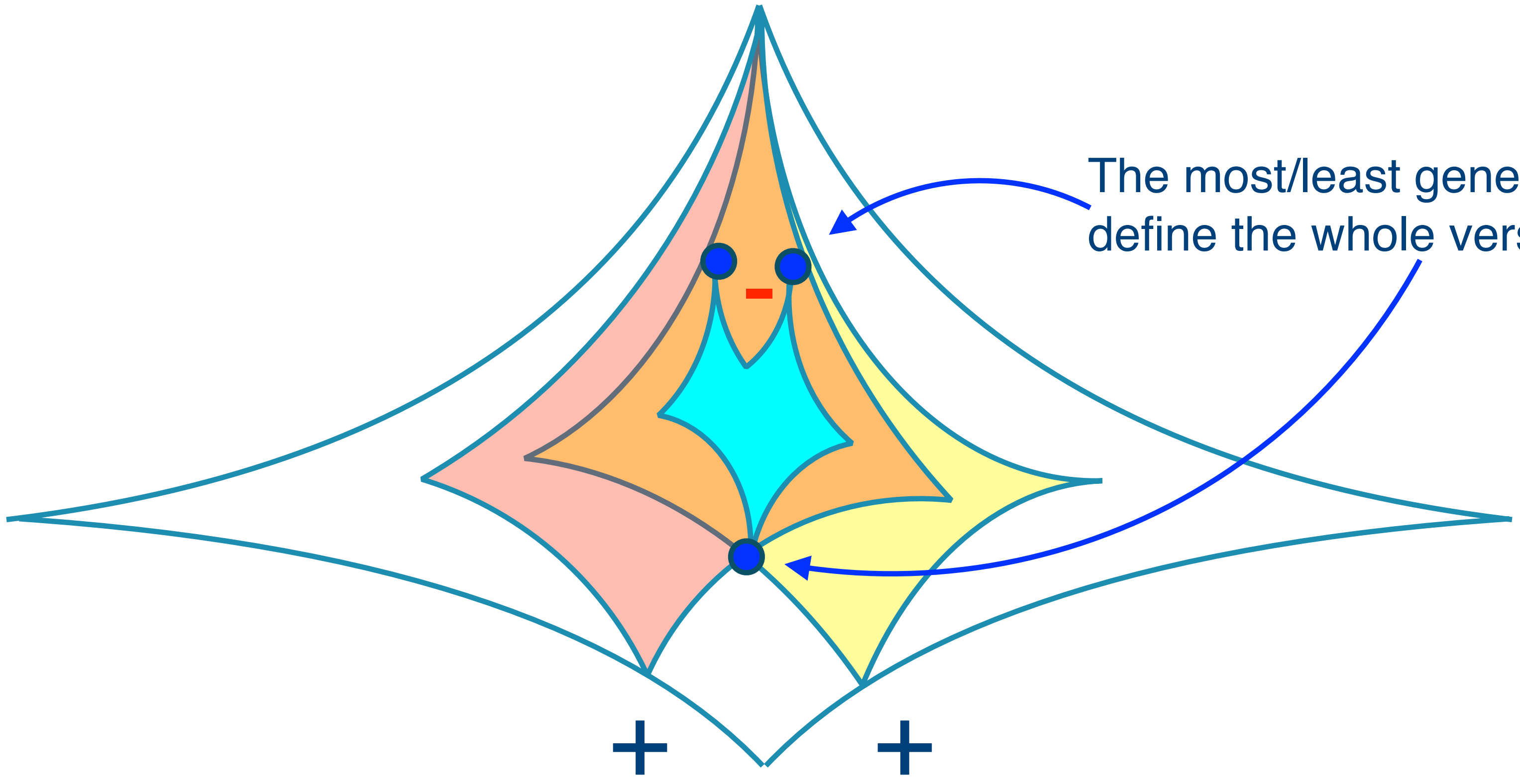




Candidate Elimination



Candidate Elimination



The most/least general solutions define the whole version space

Candidate Elimination

- The candidate elimination algorithm illustrates
  - Search in a discrete hypothesis space (with *lattice* structure)
  - Search for all solutions, rather than just one, in an efficient manner
  - Importance of generality ordering

- Some obvious disadvantages:
  - Not robust to *noise*: result = set of hypotheses consistent with *all* data; 1 erroneous data point → set may be empty!
  - Only *conjunctive* concepts : strong limitation

1 solution vs. *set of solutions*

- Predictive learning usually finds 1 model
- Candidate Elimination is an example of a method that returns a *set* of solutions
- Other examples of this setting:
  - Find all patterns (in some given pattern space) that satisfy some given criteria
    - association rules
    - constraints formulated in first order logic

Association rules

- Prototypical example from "market basket analysis": "people who buy A,B,C also buy D,E"
- General format: if $a_1, a_2, \ldots, a_n$ then $a_{n+1}, a_{n+2}, \ldots, a_{n+m}$
- Rule "If <this> then <that>" is characterized by
  - **Support**: % of all clients that buy <this>
    - Low support means: of marginal importance
  - **Confidence**: % of buyers of <this> that also buy <that>
    - Confidence need not be close to 100%
    - Any increase over normal level indicates an association
- Task: find the **set** of all association rules with support & confidence above a chosen threshold

Client	cheese	bread	butter	wine	jam	ham
1	yes	yes	yes	yes	no	yes
2	yes	no	yes	no	no	no
3	no	yes	yes	no	no	yes
...	...	...	...	...	...	...

```
IF bread & butter THEN cheese
  confidence: 50%
  support: 5%
```

Illustration: Learning constraints in first order logic

Prolog dataset / knowledge base:

```
male(homer).
male(bart).
female(marge).
female(lisa).
father(homer,lisa).
father(homer,bart).
mother(marge,lisa).
mother(marge,bart).
parent(homer,lisa).
...
```

Knowledge discovered:

```
false :- male(X), female(X).
female(X) :- mother(X,Y).
male(X) :- father(X,Y).
parent(X,Y) :- father(X,Y).
parent(X,Y) :- mother(X,Y).
...
```

KU LEUVEN

Imposing semantic constraints on learned models

- Consider AI systems that make use of learned models
- We may want to impose (symbolic) constraints on their behavior (fairness, safety, …)
  - E.g. autonomous driving: observe speed limits
  - E.g. bank loans: monotonicity constraints
- Different ways of doing this:
  - Fix model predictions when they violate a constraint
  - Use a model for which you can verify the constraints (**model verification)**
  - Use a learner that cannot return a model that violates the constraints - **model certification**

$\boxed{(x, f(x)) \models C}$

$\boxed{f \models C}$

$\boxed{\forall D : f = Learner(D) \implies f \models C}$

$\boxed{\forall x : (x, f(x)) \models C}$

Formulation as a MaxSMT(NRA) problem

- Given a parametrized class of functions f_θ , a set of hard semantic constraints $\mathscr{C} = \{C_1, \ldots C_k\}$, and a set of soft "goodness of fit" constraints $\mathscr{F} = \{F_1, \ldots F_n\}$, find a θ such that
  (1) $f_\theta \models C_i$ for all i and
  (2) $f_\theta \models F_i$ for a maximal number of i
- Typically, one C_i is, e.g., of the form $\forall \boldsymbol{x} \in \mathscr{X} : (\boldsymbol{x}, f_\theta(\boldsymbol{x})) \models \ldots$: "universal constraint" (quantified over **domain**)
- "Goodness of fit" constraints depend on the type of learning problem, e..g,. For regression: $||f(\boldsymbol{x}_i) - y_i||_2 < \varepsilon_j$ for all $(\boldsymbol{x}_i, y_i) \in T$ with $\varepsilon_1 < \varepsilon_2 < \ldots < \varepsilon_k$ given and T the training set
- Note: soft constraints express goodness of fit on T, hard constraints express domain constraints over $\mathscr{X}$
- MaxSMT(NRA) reasons about the real-valued parameters to ensure f_θ meets all hard constraints
- Practically feasible? Not without changes — but some approaches developed for linear models and ReLU-based neural networks

End of this introduction

- Questions? Feel free to come talk to me when I’m around