Statistical Machine Learning

https://cvml.ist.ac.at/courses/SML_W18

Christoph Lampert

IST AUSTRIA

Institute of Science and Technology

Spring Semester 2018/2019

Lecture 1

Overview (tentative)

Date		no.	Topic
Oct 08	Mon	1	A Hands-On Introduction
Oct 10	Wed	–	self-study (Christoph traveling)
Oct 15	Mon	2	Bayesian Decision Theory
			Generative Probabilistic Models
Oct 17	Wed	3	Discriminative Probabilistic Models
			Maximum Margin Classifiers
Oct 22	Mon	4	Generalized Linear Classifiers, Optimization
Oct 24	Wed	5	Evaluating Predictors; Model Selection
Oct 29	Mon	–	self-study (Christoph traveling)
Oct 31	Wed	6	Overfitting/Underfitting, Regularization
Nov 05	Mon	7	Learning Theory I: classical/Rademacher bounds
Nov 07	Wed	8	Learning Theory II: miscellaneous
Nov 12	Mon	9	Probabilistic Graphical Models I
Nov 14	Wed	10	Probabilistic Graphical Models II
Nov 19	Mon	11	Probabilistic Graphical Models III
Nov 21	Wed	12	Probabilistic Graphical Models IV
until Nov 25			final project

Definition (Mitchell, 1997)

A computer program is said to *learn* from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.

Definition (Mitchell, 1997)

A computer program is said to *learn* from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.

Example: Backgammon

- T)ask: Play backgammon.
- E)xperience: Games playes against itself
- P)erformance Measure: Games won against human players.

Definition (Mitchell, 1997)

A computer program is said to *learn* from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.

Example: Spam classification

- T)ask: determine if emails are Spam or non-Spam.
- E)xperience: Incoming emails with human classification
- P)erformance Measure: percentage of correct decisions

Definition (Mitchell, 1997)

A computer program is said to *learn* from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.

Example: Stock market predictions

- T)ask: predict the price of some shares
- E)xperience: past prices
- P)erformance Measure: money you win or lose

Notation

Task:

- $\mathcal{X}$: input set, set of all possible inputs
- $\mathcal{Y}$: output set, set of all possible outputs
- $f : \mathcal{X} \to \mathcal{Y}$: prediction function,
  - e.g. $\mathcal{X} = \{\texttt{all possible emails}\}, \mathcal{Y} = \{\texttt{spam}, \texttt{ham}\}$
    f spam filter: for new email $x \in \mathcal{X}$: $f(x) = \texttt{spam}$ or $f(x) = \texttt{ham}$.

Performance:

- $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$: loss function
  - e.g. $\ell(y, y')$ is *cost* of predicting y' if y is correct.
  - $\ell(y, y')$ can be asymmetric: $\texttt{spam} \to \texttt{ham}$ is annoying, but no big deal.
  - $\texttt{ham} \to \texttt{spam}$ can cause serious problems.

Experience: task-dependent, many different scenarios

- **Supervised learning**: a labeled **training set**
  examples from $\mathcal{X}$ with outputs provided by an expert,
  $\mathcal{D} = \{(x^1, y^1), \dots, (x^n, y^n)\} \subset \mathcal{X} \times \mathcal{Y}$
  - A person goes through his/her n emails and marks each one whether it is spam or not.

Many other variants on how to formalize *experience* exist:

- **Unsupervised Learning:** $\mathcal{D} = \{x^1, \dots, x^n\}$, only observing, no input from an expert/teacher
- **Semi-supervised Learning:** $\mathcal{D} = \{(x^1, y^1), \dots, (x^l, y^l)\} \cup \{x^{l+1}, \dots, x^n\}$: only a subset of examples has labels (common for spam filters)
- **Reinforcement Learning:** $\mathcal{D} = \{(x^1, r^1), \dots, (x^n, r^n)\}$ with $r^i \in \mathbb{R}$: actions and feedback how good the action was (backgammon: nobody tells you the best move, but eventually you observe the outcome of winning or losing)
- **Multiple Instance Learning:** $\mathcal{D} = \{(X^1, y^1), \dots, (X^n, y^n)\}$ where $X^i = \{x^{i,1}, \dots, x^{i,n_i}\}$. Labels are given not for individual samples, but for groups (e.g. parmacy: a drug cocktail has a certain effect, but it could have been any of the active substances inside)
- **Active Learning:** $\mathcal{D} = \{x^1, \dots, x^n\}$, but the algorithms may *ask* for labels (spam: email program can ask the user, if its not too often)

Definition

- A *supervised learning system* (or *learner*), L, is a (computable) function from the set of (finite) training sets to the set of prediction functions:

$$L : \mathbb{P}^{<\infty}(\mathcal{X} \times \mathcal{Y}) \to \mathcal{Y}^{\mathcal{X}}$$
$$\text{i.e.} \quad L : \mathcal{D} \mapsto f$$

If presented with a training set $\mathcal{D} \subset \mathcal{X} \times \mathcal{Y}$, it provides a decision rule/function $f : \mathcal{X} \to \mathcal{Y}$.

Definition

Let L be a learning system.

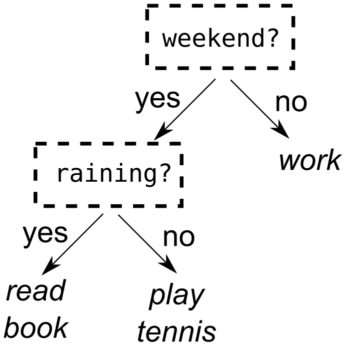
- The process of *computing* $f = L(\mathcal{D})$ is called *training (phase)*.
- Applying f to new data is called *prediction*, or *testing* (phase).

We will look at examples of classical learning algorithms, to get a feeling what *problems* a learning system faces.

- Decision Trees
- Nearest Neighbor Classifiers
- Perceptron
- Boosting
- Artificial Neural Networks

Caveat: for each of these are there more advanced, often better, variants. Here, we look at the only as prototypes, not as guideline what to actually use in real life.

Task: decide what to do today



Classifier has a tree structure:

- each *interior node* makes a decision: it picks an attribute within x, branches for each possible value
- each *leaf* has one output label
- to classify a new example, we
  - put it into the root node,
  - follow the decisions until we reach a leaf.
  - use the leaf value as the prediction

Decisions trees ('expert systems') are popular especially for non-experts:

- **easy to use**, and **interpretable.**

How to automatically build a decision tree

Given: training set $\mathcal{D} = \{(x^1, y^1), \dots, (x^n, y^n)\}$.

Convention:

- each node contains a subset of examples,
- its label is the majority label of the examples in this node (any of the majority labels, if there's a tie)

Decision Tree – Training

initialize: put all examples in root node
mark root as *active*
repeat
  pick active node with largest number of misclassified examples
  mark the node as *inactive*
  for each attributes, check error rate of splitting along this attribute
  keep the split with smallest error, if any, and mark children as *active*
until no more active nodes.

Decision Tree – Classification

input decision tree, example x
  assign x to root node
  while x not in leaf node **do**
    move x to child according to the test in node
  end while
output label of the leaf that x is in

Decision Trees Example - Training

- We have a personalized dating agency, our only customer is *Zoe*.
- Task: For new customers registering, predict if *Zoe* should date them.
- Performance Measure: If *Zoe* is happy with the decision.
- Experience: We show *Zoe* a catalog of previous custemers and she tells us whether she would have like to date them or not.
- Let $x \in \mathcal{X}$ be a collection of values or properties, $x = (x_1, \ldots, x_d)$.

property	possible values
`eye color`	blue/brown/green
`handsome`	yes/no
`height`	short/tall
`sex`	male (M)/female (F)
`soccer fan`	yes/no

Preparation: you give Zoe a set of profiles to see whom she would like to date (none of these people really have to exists...)

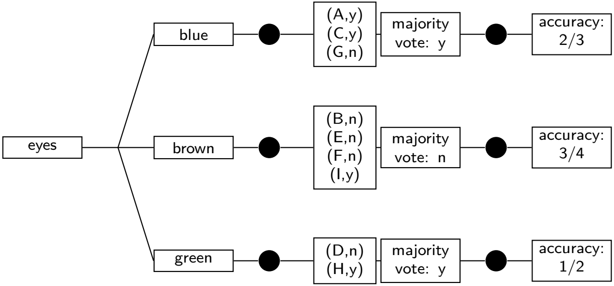
Here's her answers, which we'll use as **training data:**

		$\mathcal{X}$				$\mathcal{Y}$
person	eyes	handsome	height	sex	soccer	date?
Apu	blue	yes	tall	M	no	yes
Bernice	brown	yes	short	F	no	no
Carl	blue	no	tall	M	no	yes
Doris	green	yes	short	F	no	no
Edna	brown	no	short	F	yes	no
Prof. Frink	brown	yes	tall	M	yes	no
Gil	blue	no	tall	M	yes	no
Homer	green	yes	short	M	no	yes
Itchy	brown	no	short	M	yes	yes

Step 1: put all all training examples into the root node

$$root = \{ \text{(A,y),(B,n),(C,y),(D,n),(E,n),(F,n),(G,n),(H,y),(I,y)} \}$$

For each feature, check the classification accuracy of this single feature:



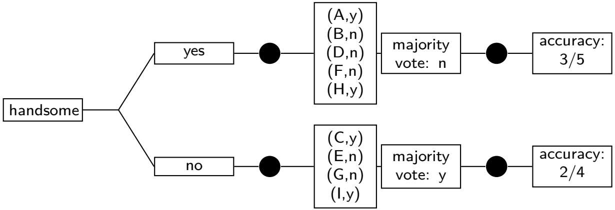
Total accuracy eyes: 6/9

Decision Trees Example - Training phase

Step 1: put all all training examples into the root node

$$root = \{\ (\text{A,y}),(\text{B,n}),(\text{C,y}),(\text{D,n}),(\text{E,n}),(\text{F,n}),(\text{G,n}),(\text{H,y}),(\text{I,y})\ \}$$

For each feature, check the classification accuracy of this single feature:



Total accuracy `handsome`: 5/9

Decision Trees Example - Training phase

Step 1: put all all training examples into the root node

$$root = \{\ (\text{A,y}),(\text{B,n}),(\text{C,y}),(\text{D,n}),(\text{E,n}),(\text{F,n}),(\text{G,n}),(\text{H,y}),(\text{I,y})\ \}$$

For each feature, check the classification accuracy of this single feature:

feature	accuracies	$\rightarrow$	total
`eyes`	blue: (2/3), brown: (3/4), green: (1/2)	$\rightarrow$	total: (6/9)
`handsome`	yes: (3/5), no: (2/4)	$\rightarrow$	total: (5/9)
`height`	tall: (2/4), short: (3/5)	$\rightarrow$	total: (5/9)
`sex`	male: (4/6), female: (3/3)	$\rightarrow$	**total: (7/9)**
`soccer`	yes: (3/4), no: (3/6)	$\rightarrow$	total: (6/9)

Best feature: `sex`.

Decision Trees Example - Training phase

Step 1 result: first split ist along `sex` feature

```
                       sex
            m/                          \f
(A,y),(C,y),(F,n),(G,n),(H,y),(I,y)    (B,n), (D,n), (E,n)
```

Right node: no mistakes, no more splits

Left node: run checks again for remaining data

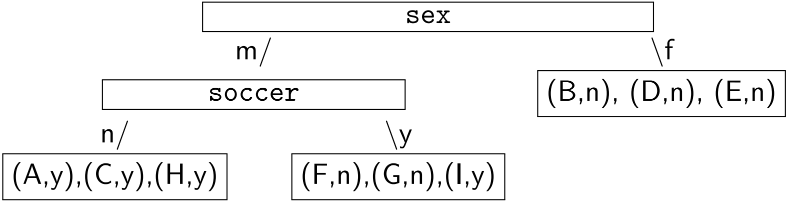
Step 2:

person	eyes	handsome	height	sex	soccer	date?
Apu	blue	yes	tall	male	no	yes
Carl	blue	no	tall	male	no	yes
Frink	brown	yes	tall	male	yes	no
Gil	blue	no	tall	male	yes	no
Homer	green	yes	short	male	no	yes
Itchy	brown	no	short	male	yes	yes

feature	accuracies	$\rightarrow$	total
eyes	blue: (2/3), brown: (1/2), green: (1/1)	$\rightarrow$	total: (4/6)
handsome	yes: (2/3), no: (2/3)	$\rightarrow$	total: (4/6)
height	tall: (2/4), short: (2/2)	$\rightarrow$	total: (4/6)
sex	male: (4/6)	$\rightarrow$	total: (4/6)
soccer	yes: (2/3), no: (3/3)	$\rightarrow$	**total: (5/6)**

Best feature: `soccer`.

Step 2 result: second split ist along `soccer` feature



Left node: no mistakes, no more splits
Right node: run checks again for remaining data

Step 3:

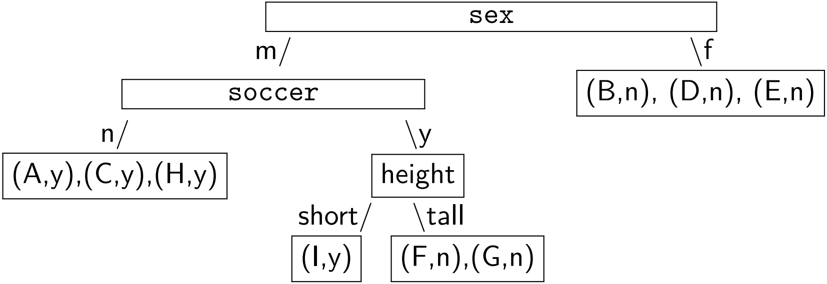
person	eyes	handsome	height	sex	soccer	date?
Frink	brown	yes	tall	male	yes	no
Gil	blue	no	tall	male	yes	no
Itchy	brown	no	short	male	yes	yes

feature	accuracies	$\rightarrow$	total
eyes	blue: $(1/1)$, brown: $(1/2)$, green: $(0/0)$	$\rightarrow$	total: $(2/3)$
handsome	yes: $(1/1)$, no: $(1/2)$	$\rightarrow$	total: $(2/3)$
height	tall: $(2/2)$, short: $(1/1)$	$\rightarrow$	**total: (3/3)**
sex	male: $(2/3)$	$\rightarrow$	total: $(2/3)$
soccer	yes: $(2/3)$	$\rightarrow$	total: $(2/3)$

Best feature: `height`.

Decision Trees Example - Training phase

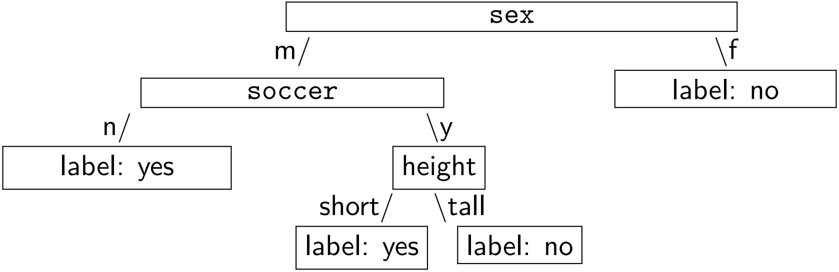
Step 3 result: third split ist along `height` feature



Left node: no mistakes, no more splits
Right node: no mistakes, no more splits

Decision Trees Example - Training phase

Step 3 result: third split ist along `height` feature



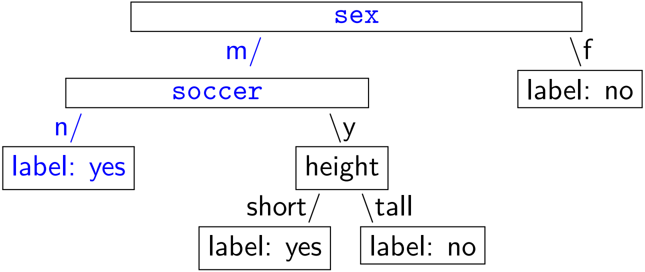
Left node: no mistakes, no more splits
Right node: no mistakes, no more splits

→ Decision tree learning complete.

Decision Trees Example - How good is this classifier?

Training example 1: correct

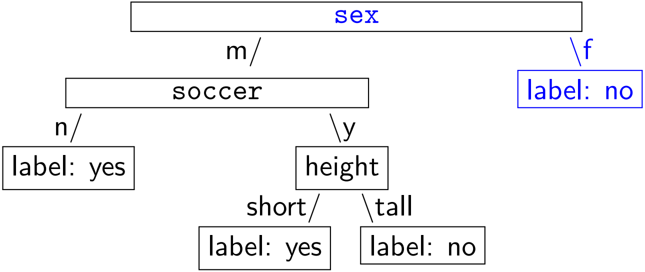
person	eyes	handsome	height	sex	soccer	date?
Apu	blue	yes	tall	male	no	yes



Decision Trees Example - How good is this classifier?

Training example 2: correct

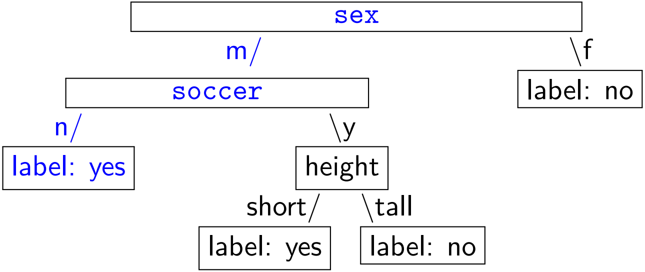
person	eyes	handsome	height	sex	soccer	date?
Bernice	brown	yes	short	F	no	no



Decision Trees Example - How good is this classifier?

Training example 3: correct

person	eyes	handsome	height	sex	soccer	date?
Carl	blue	no	tall	M	no	yes



Decision Trees Example - How good is this classifier?

- All training examples are classified correctly!

- All training examples are classified correctly!

Not overly surprising... that's how we constructed the tree.

Decision Trees Example - How good is this classifier?

What if we check on new data of the same kind?

person	eyes	handsome	height	sex	soccer	date?	
Jimbo	blue	no	tall	M	no	yes	
Krusty	green	yes	short	M	yes	no	
Lisa	blue	yes	tall	F	no	no	
Moe	brown	no	short	M	no	no	
Ned	brown	yes	short	M	no	yes	
Quimby	blue	no	tall	M	no	yes	

Decision Trees Example - How good is this classifier?

What if we check on new data of the same kind?

person	eyes	handsome	height	sex	soccer	date?	tree
Jimbo	blue	no	tall	M	no	yes	**yes**
Krusty	green	yes	short	M	yes	no	**yes**
Lisa	blue	yes	tall	F	no	no	**no**
Moe	brown	no	short	M	no	no	**yes**
Ned	brown	yes	short	M	no	yes	**yes**
Quimby	blue	no	tall	M	no	yes	**yes**

2 mistakes in 6, hm...

Observation

Zoe won't care if our tree classifier worked perfectly on the training data. What really matters is how it works on future data: **ability to generalize**

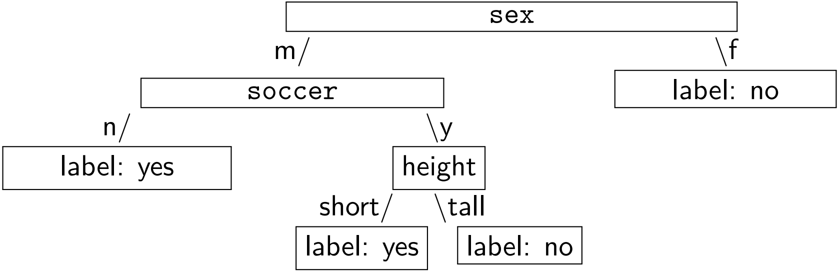
Observation

There is a relation between accuracy during training and accuracy at test time, but it isn't a simple one. **Perfect performance on the training set does not guarantee perfect performance on future data!**

Why did the tree make a mistake?
Maybe it took the training data too seriously?
Would Zoe really decide that male soccer fans are only datable, if they are *short*, but not if they are *tall*?
Let's see what happens in we simplify the tree?

Decision Trees Example - How good is this classifier?

Original four-level tree: 2 mistakes in 6.



person	eyes	handsome	height	sex	soccer	date?	tree
Jimbo	blue	no	tall	M	no	yes	yes
Krusty	green	yes	short	M	yes	no	yes
Lisa	blue	yes	tall	F	no	no	no
Moe	brown	no	short	M	no	no	yes
Ned	brown	yes	short	M	no	yes	yes
Quimby	blue	no	tall	M	no	yes	yes

Decision Trees Example - How good is this classifier?

Tree with three levels: 1 mistake in 6.

```
                 sex
            m/          \f
       soccer          label: no
     n/       \y
label: yes   label: no
```

person	eyes	handsome	height	sex	soccer	date?	tree
Jimbo	blue	no	tall	M	no	yes	**yes**
Krusty	green	yes	short	M	yes	no	**no**
Lisa	blue	yes	tall	F	no	no	**no**
Moe	brown	no	short	M	no	no	**yes**
Ned	brown	yes	short	M	no	yes	**yes**
Quimby	blue	no	tall	M	no	yes	**yes**

Decision Trees Example - How good is this classifier?

Tree with two levels: 2 mistakes in 6.

sex

m/ \f

label: yes | label: no

person	eyes	handsome	height	sex	soccer	date?	tree
Jimbo	blue	no	tall	M	no	yes	yes
Krusty	green	yes	short	M	yes	no	yes
Lisa	blue	yes	tall	F	no	no	no
Moe	brown	no	short	M	no	no	yes
Ned	brown	yes	short	M	no	yes	yes
Quimby	blue	no	tall	M	no	yes	yes

Decision Trees Example - How good is this classifier?

Tree with one level: 3 mistakes in 6.

label: no

person	eyes	handsome	height	sex	soccer	date?	tree
Jimbo	blue	no	tall	M	no	yes	**no**
Krusty	green	yes	short	M	yes	no	**no**
Lisa	blue	yes	tall	F	no	no	**no**
Moe	brown	no	short	M	no	no	**no**
Ned	brown	yes	short	M	no	yes	**no**
Quimby	blue	no	tall	M	no	yes	**no**

Decision Trees Example - How good is this classifier?

Error analysis:

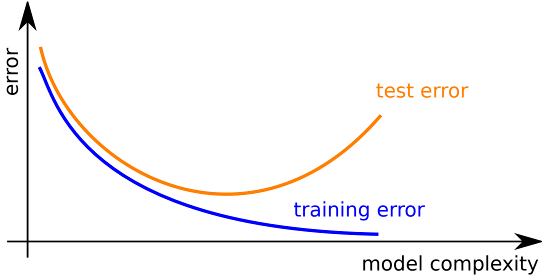
size	training error	test error
height 1	$4/9$	$3/6$
height 2	$2/9$	$2/6$
height 3	$1/9$	$1/6$
height 4 (full)	$0/9$	$2/6$

Decision Trees Example - How good is this classifier?

Error analysis:

size	training error	test error
height 1	4/9	3/6
height 2	2/9	2/6
height 3	1/9	1/6
height 4 (full)	0/9	2/6

Very typical behaviour of machine learning systems:



Decision Tree Example - Lessons learned

Classifiers can have different **complexity**:

- **Complexity** has impact on both: training error and testing error.
- Training error: usually decreases with increasing complexity
- Test error: first decreases, then might go up again.

Test error behavior is so common that it has its own name:

- too simple models: high test error due to **underfitting**
  - the model cannot absorb the information from the training data
- too complex models: high test error due to **overfitting**
  - the model tries to reproduce idiosyncracies of the training data that future data will not have

Optimal classifier has a complexity somewhere inbetween, but:

- we cannot tell from either training error or test error alone if we underfit, overfit or neither
- seeing the complete *curve* will tell us!

- Categorial data can often be handled nicely by a tree.
- For continuous data, $\mathcal{X} = \mathbb{R}^d$, one typically uses splits by comparing any coordinate by a threshold: $[\![x_i \geq \theta]\!]$?
- Finding a split consists of checking all $i = 1, \ldots, d$ and all (reasonable) thresholds, e.g. all $x_i^1, \ldots, x_i^n$
- If d is large, and all dimension are roughly of equal importance (e.g. time series), this is tedious, and the resulting tree might not be good.

Nearest Neighbor – analysis: Cover&Thomas, 1967

Nearest Neighbor – Training

input dataset $\mathcal{D} = \{(x^1, y^1), \dots, (x^n, y^n)\} \subset \mathbb{R}^d \times \mathcal{Y}$
store all examples $(x^1, y^1), \dots, (x^n, y^n)$.

Nearest Neighbor – Prediction

input new example $x \in \mathbb{R}^d$
for each training example (x^i, y^i)
compute $\mathit{dist}_i(x) = \|x - x^i\|$ (Euclidean distance)
output y^j for $j = \mathbf{argmin}_j\, \mathit{dist}_i(x)$

(if $\mathbf{argmin}$ is not unique, pick between possible examples)

Definition (Decision Boundary)

Let $f : \mathcal{X} \to \mathcal{Y}$ be a classifier with discrete $\mathcal{Y} = \{1, \ldots, M\}$. The points where f is *discontinuous* are called *decision boundary*.

Blackboard illustration

Nearest Neighbor

Nearest Neighbor prediction in the real world:

- very natural and intuitive
- we apply it without even considering it "learning" or "prediction"
- very popular in industry under the name 'case based reasoning', for example helpdesk: **"Similar problems have similar solutions".**

From a machine learning point of view:

- consider data as points in a (potentially high-dim.) vector space
- distance between two points tells us their *similarity*
- **Similar points tend to have the same label**.

We can also use NN for categorical labels: embed values into $\mathbb{R}^d$, e.g.

$$x_{Apu} = (\underbrace{1}_{blue}, \underbrace{0}_{brown}, \underbrace{0}_{green}, \underbrace{1}_{handsome}, \underbrace{0}_{not\ handsome}, \underbrace{1}_{tall}, \underbrace{0}_{short}, \underbrace{1}_{male}, \underbrace{0}_{female}, \underbrace{1}_{soccer}, \underbrace{0}_{notsoccer})$$

k-Nearest Neighbor – Training

input dataset $\mathcal{D} = \{(x^1, y^1), \ldots, (x^n, y^n)\} \subset \mathbb{R}^d \times \mathcal{Y}$

store all examples $(x^1, y^1), \ldots, (x^n, y^n)$.

k-Nearest Neighbor – Classification

input new example x

for each training example (x^i, y^i) compute $d_i(x) = \|x - x^i\|$ (Euclidean distance)

sort d_i in increasing order

output majority vote among y^is within the k smallest d^i

Observation: Previous "nearest neighbor" is 1-nearest neighbour.
For $k > 2$, k-NN can ignore training example, if the neighbors don't support their label.

k controls the complexity of the model:

- $k = n$, we always may the majority decision (underfitting).
- $k = 1$, decisions based on a single (most similar) example at a time, this might have an unreliable label (overfitting).
- as before: there's a sweet spot inbetween.

Perceptron – Rosenblatt, 1957

So far we've seen two classifiers:

- decision tree: picks a few important features to base decision on
- k-NN: all features contribute equally (to Euclidean distance)

Often, neither is optimal:

- we have many features, we want to make use of them.
- but some features are more useful or reliable than others.

Idea: learn how important each feature, x_j, is by a weight, w_j

Perceptron algorightm: inspired by (early) neuroscience:

- neurons form a weighted sum of their inputs $x = (x_1, \dots, x_d)$
- they output a spike if the result exceeds a threshold, θ

$$h(x) = \begin{cases} +1 & \text{if } \sum_j w_j x_j \geq \theta \\ -1 & \text{otherwise} \end{cases} \qquad = \operatorname{sign}\left(\langle w, x\rangle - \theta\right).$$

Perceptron – Training (for $\theta = 0$)

input training set $\mathcal{D} \subset \mathbb{R}^d \times \{-1, +1\}$

initialize $w = (0, \dots, 0) \in \mathbb{R}^d$.

repeat

for all $(x, y) \in \mathcal{D}$: **do**

compute $a := \langle w, x \rangle$ ('activation')

if $ya \leq 0$ **then**

$w \leftarrow w + yx$

end if

end for

until w wasn't updated for a complete pass over $\mathcal{D}$

Perceptron – Classification (for $\theta = 0$)

input new example x

output $f(x) = \operatorname{sign}\langle w, x \rangle$ by convention, $\operatorname{sign}(0) = -1$

Perceptron – Example

$\mathcal{D}$: $(x^1, y^1) = (\begin{pmatrix}3\\1\end{pmatrix}, +1)$, $(x^2, y^2) = (\begin{pmatrix}1\\1\end{pmatrix}, +1)$, $(x^3, y^3) = (\begin{pmatrix}1\\4\end{pmatrix}, -1)$.

Round 1:

- $w = \begin{pmatrix}0\\0\end{pmatrix}$, $\quad i = 1$: $\langle w, x^1\rangle = 0$, $\quad 1 \cdot 0 = 0 \leq 0 \quad \rightarrow$ update

  $$w_{new} = w_{old} + 1 \cdot \begin{pmatrix}3\\1\end{pmatrix} = \begin{pmatrix}3\\1\end{pmatrix}$$

- $w = \begin{pmatrix}3\\1\end{pmatrix}$, $\quad i = 2$: $\langle w, x^2\rangle = 4 \quad 1 \cdot 4 = 4 \not\leq 0 \quad \rightarrow$ no change
- $w = \begin{pmatrix}3\\1\end{pmatrix}$, $\quad i = 3$: $\langle w, x^3\rangle = 7$, $\quad (-1) \cdot 7 = -7 \leq 0 \quad \rightarrow$ update

  $$w_{new} = w_{old} + (-1)\begin{pmatrix}1\\4\end{pmatrix} = \begin{pmatrix}2\\-3\end{pmatrix}$$

Perceptron – Example

$\mathcal{D}$: $(x^1, y^1) = (\begin{pmatrix} 3 \\ 1 \end{pmatrix}, +1)$, $(x^2, y^2) = (\begin{pmatrix} 1 \\ 1 \end{pmatrix}, +1)$, $(x^3, y^3) = (\begin{pmatrix} 1 \\ 4 \end{pmatrix}, -1)$.

Round 2:

- $w = \begin{pmatrix} 2 \\ -3 \end{pmatrix}$, $\quad i = 1$: $\langle w, x^1 \rangle = 3, \quad 1 \cdot 3 = 3 \not\leq 0 \quad \rightarrow$ no change
- $w = \begin{pmatrix} 2 \\ -3 \end{pmatrix}$, $\quad i = 2$: $\langle w, x^2 \rangle = -1, \quad 1 \cdot (-1) = -1 \leq 0$

  $\rightarrow \quad w_{new} = w_{old} + 1 \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 3 \\ -2 \end{pmatrix}$
- $w = \begin{pmatrix} 3 \\ -2 \end{pmatrix}$, $\quad i = 3$: $\langle w, x^3 \rangle = -5, \quad (-1) \cdot (-5) = 5 \not\leq 0$

  $\rightarrow$ no change

Perceptron – Example

$\mathcal{D}$: $(x^1, y^1) = (\begin{pmatrix} 3 \\ 1 \end{pmatrix}, +1)$, $(x^2, y^2) = (\begin{pmatrix} 1 \\ 1 \end{pmatrix}, +1)$, $(x^3, y^3) = (\begin{pmatrix} 1 \\ 4 \end{pmatrix}, -1)$.

Round 3:

- $w = \begin{pmatrix} 3 \\ -2 \end{pmatrix}$, $\quad i = 1$: $\langle w, x^1 \rangle = 7, \quad 1 \cdot 7 = 7 \not\leq 0$
- $w = \begin{pmatrix} 3 \\ -2 \end{pmatrix}$, $\quad i = 2$: $\langle w, x^2 \rangle = 1, \quad 1 \cdot 1 = 1 \not\leq 0$
- $w = \begin{pmatrix} 3 \\ -2 \end{pmatrix}$, $\quad i = 3$: $\langle w, x^3 \rangle = -5, \quad (-5) \cdot (-1) = 5 \not\leq 0$

nothing changed for 1 complete round: converged

Perceptron – Example

$\mathcal{D}$: $(x^1, y^1) = (\begin{pmatrix} 3 \\ 1 \end{pmatrix}, +1)$, $(x^2, y^2) = (\begin{pmatrix} 1 \\ 1 \end{pmatrix}, +1)$, $(x^3, y^3) = (\begin{pmatrix} 1 \\ 4 \end{pmatrix}, -1)$.

Round 3:

- $w = \begin{pmatrix} 3 \\ -2 \end{pmatrix}$, $\quad i = 1$: $\langle w, x^1 \rangle = 7, \quad 1 \cdot 7 = 7 \not\leq 0$
- $w = \begin{pmatrix} 3 \\ -2 \end{pmatrix}$, $\quad i = 2$: $\langle w, x^2 \rangle = 1, \quad 1 \cdot 1 = 1 \not\leq 0$
- $w = \begin{pmatrix} 3 \\ -2 \end{pmatrix}$, $\quad i = 3$: $\langle w, x^3 \rangle = -5, \quad (-5) \cdot (-1) = 5 \not\leq 0$

  nothing changed for 1 complete round: converged

Final classifier: $f(x) = \mathrm{sign}\left(3 \cdot x_1 - 2 \cdot x_2\right)$

Limitation: always has a *linear* decision boundary, might not converge

Boosting

Given: training examples $\mathcal{D} = \{(x^1, y^1), \ldots, (x^n, y^n)\} \subset \mathcal{X} \times \mathcal{Y}$.
For simplicity: $\mathcal{Y} = \{\pm 1\}$.

Main insight of Boosting:

- It's hard to guess a strong (=good) classifier.
- It's easy to guess *weak* classifiers.

Boosting takes a large set of weak classifiers and combines them into a single strong classifier.

Boosting – Weak Classifiers

For example: if our features are

property	possible values
`eye color`	blue/brown/green
`handsome`	yes/no
`height`	short/tall
`sex`	male (M)/female (F)
`soccer fan`	yes/no

define (weak) classifiers:

$$h_1(x) = \begin{cases} +1 & \text{if } \texttt{eye color} = \texttt{brown} \\ -1 & \text{otherwise.} \end{cases} \qquad h_2(x) = \begin{cases} +1 & \text{if } \texttt{eye color} = \texttt{blue} \\ -1 & \text{otherwise.} \end{cases}$$

$$h_3(x) = \begin{cases} +1 & \text{if } \texttt{eye color} = \texttt{green} \\ -1 & \text{otherwise.} \end{cases} \qquad h_4(x) = \begin{cases} -1 & \text{if } \texttt{eye color} = \texttt{brown} \\ +1 & \text{otherwise.} \end{cases}, \ldots$$

$$h_5(x) = \begin{cases} +1 & \text{if } \texttt{handsome} = \texttt{yes} \\ -1 & \text{otherwise.} \end{cases} \qquad h_6(x) = \begin{cases} -1 & \text{if } \texttt{handsome} = \texttt{yes} \\ +1 & \text{otherwise.} \end{cases}, \ldots$$

Set of all possible combinations: $\mathcal{H} = \{h_1, \ldots, h_J\}$.

AdaBoost – Training

input training set $\mathcal{D}$, set of weak classifiers $\mathcal{H}$, number of iterations T.

$d_1 = d_2 = \cdots = d_n = 1/n$ (weight for each example)

for t=1,...,T **do**

for $h \in \mathcal{H}$ **do** $e^t(h) = \sum_{i=1}^{n} d_i \, [\![h(x^i) \neq y^i]\!]$ (weighted training error)

$h_t = \mathbf{argmin}_{h \in \mathcal{H}} \, e^t(h)$ ("best" of the weak classifiers)

$\alpha_t = \frac{1}{2} \log(\frac{1 - e_t(h_t)}{e_t(h_t)})$ (classifier importance, $\alpha_t = 0$ if $e_t(h_t) = \frac{1}{2}$)

for $i = 1, \ldots, n$ **do** $\tilde{d}_i \leftarrow d_i \times \begin{cases} e^{\alpha_t} & \text{if } h_t(x^i) \neq y^i, \\ e^{-\alpha_t} & \text{otherwise.} \end{cases}$

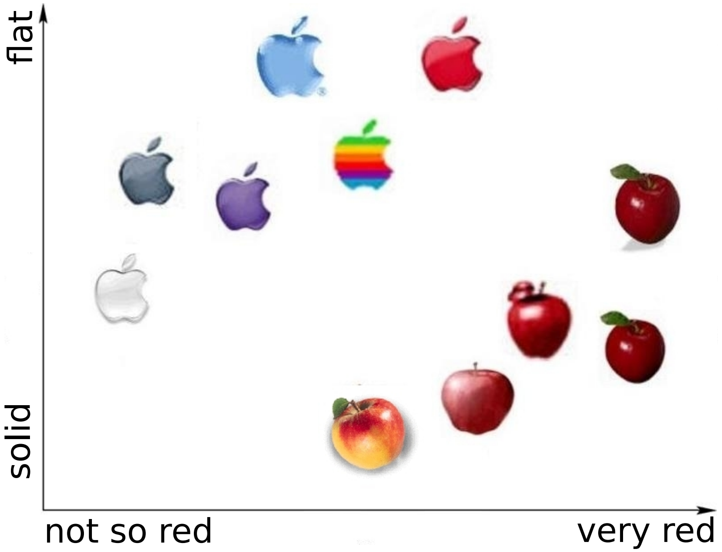
for $i = 1, \ldots, n$ **do** $d_i \leftarrow \tilde{d}_i / \sum_i \tilde{d}_i$

end for

output classifier: $f(x) = \operatorname{sign} \sum_{t=1}^{T} \alpha_t h_t(x)$

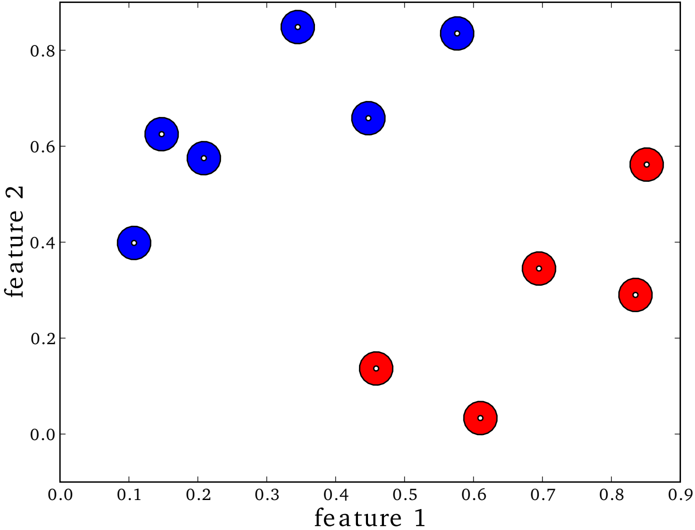
AdaBoost – Example

Task: $\mathcal{X} = \mathbb{R}^2$, weak classifiers look at each dimension separately

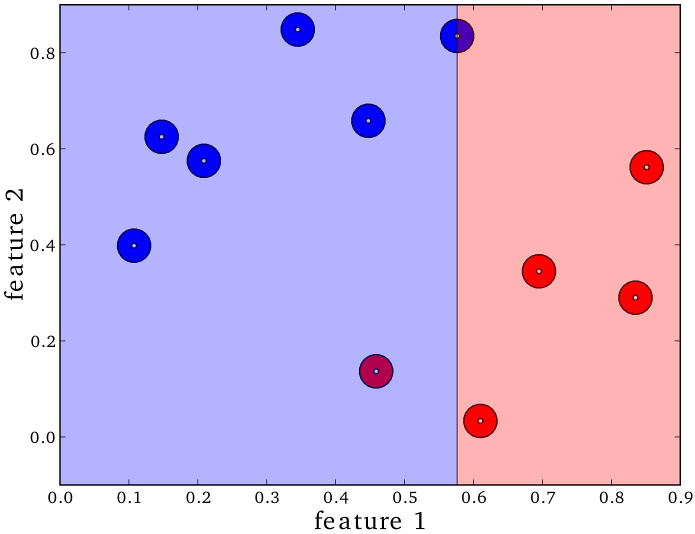


AdaBoost – Example

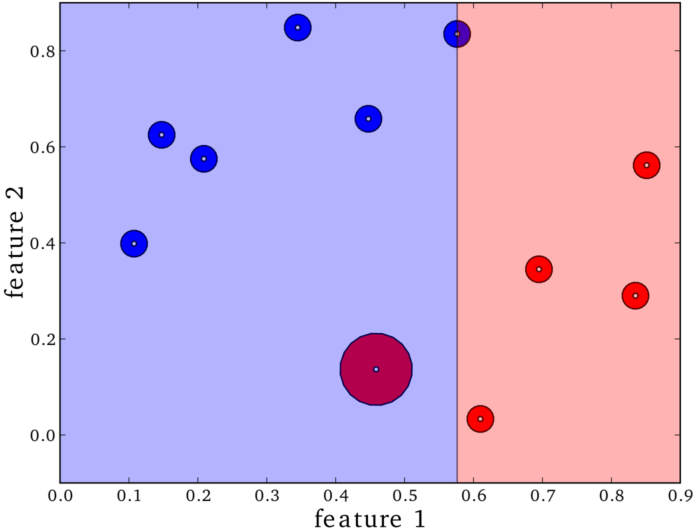
Iteration $t = 1, \quad d_1, \ldots, d_n = (\frac{1}{11}, \frac{1}{11}, \frac{1}{11}, \frac{1}{11}, \frac{1}{11}, \frac{1}{11}, \frac{1}{11}, \frac{1}{11}, \frac{1}{11}, \frac{1}{11}, \frac{1}{11})$



Iteration $t = 1$, best weak classifier, $e_1(h_1) = \frac{1}{11}$, $\alpha_1 = 1.15$

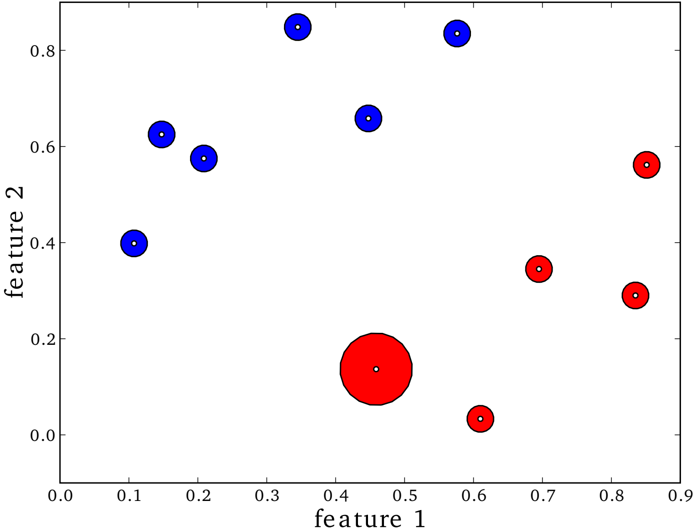


Iteration $t = 1$, best weak classifier, $e_1(h_1) = \frac{1}{11}$, $\alpha_1 = 1.15$



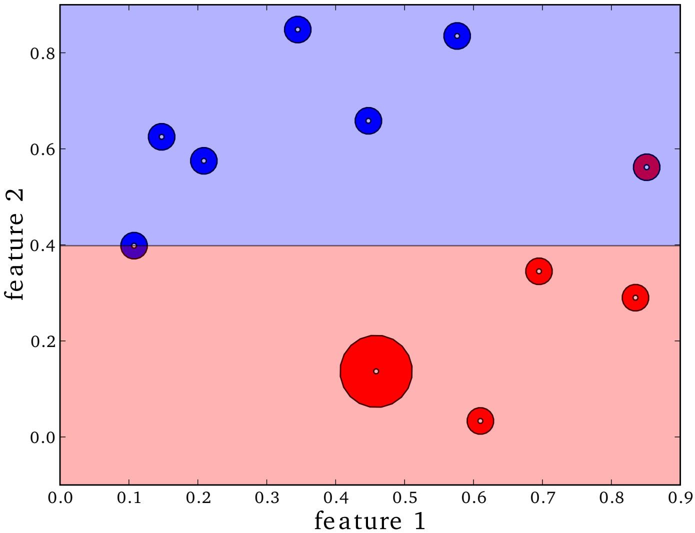
AdaBoost – Example

Iteration $t = 2, \quad d_1, \ldots, d_n \approx (\frac{1}{20}, \frac{1}{20}, \frac{1}{20}, \frac{1}{20}, \frac{1}{20}, \frac{1}{20}, \frac{1}{20}, \ \frac{1}{2} \ , \frac{1}{20}, \frac{1}{20}, \frac{1}{20}, \frac{1}{20})$



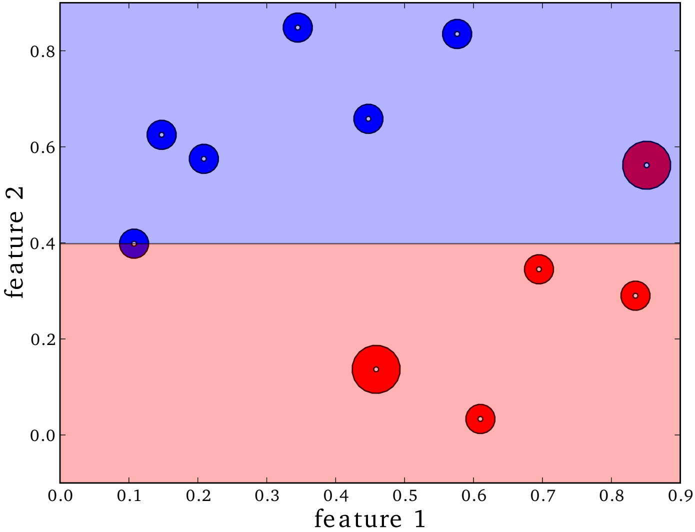
AdaBoost – Example

Iteration $t = 2$, best weak classifier, $e_2(h_2) = \frac{1}{20}$, $\alpha_2 = 1.47$



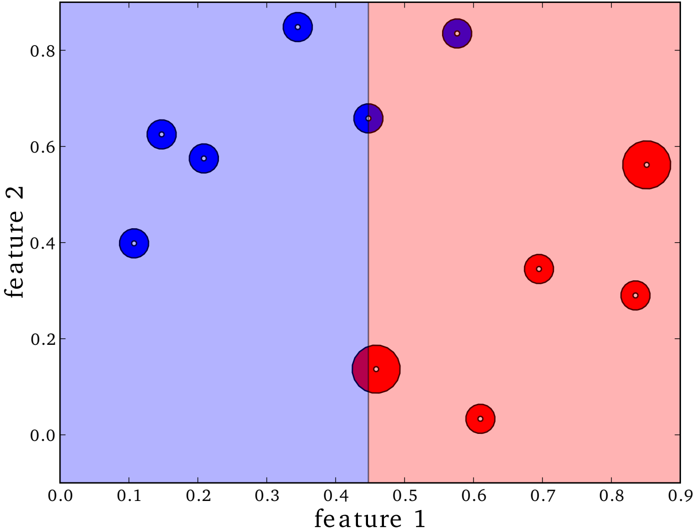
AdaBoost – Example

Iteration $t = 2$, best weak classifier, $e_2(h_2) = \frac{1}{20}$, $\alpha_2 = 1.47$

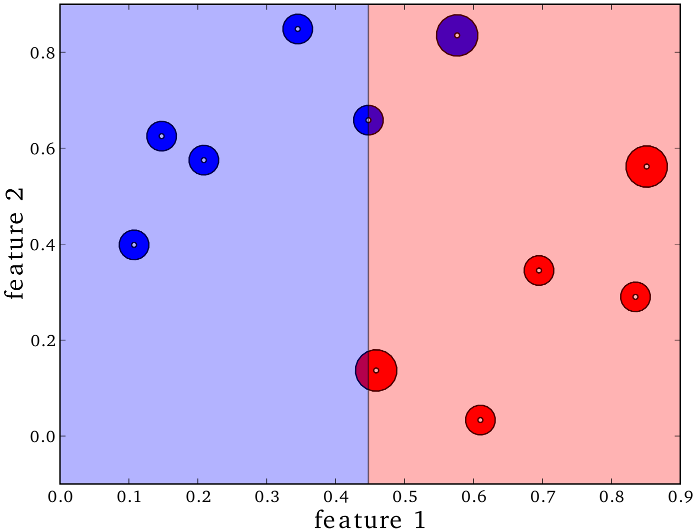


AdaBoost – Example

Iteration $t = 3$

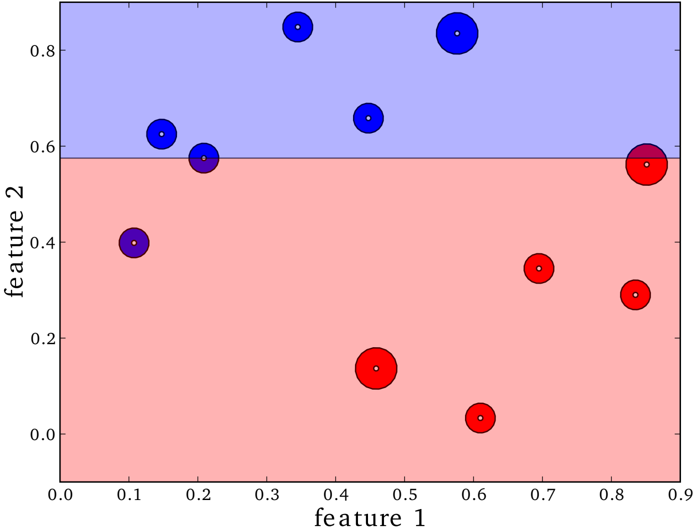


Iteration $t = 3$

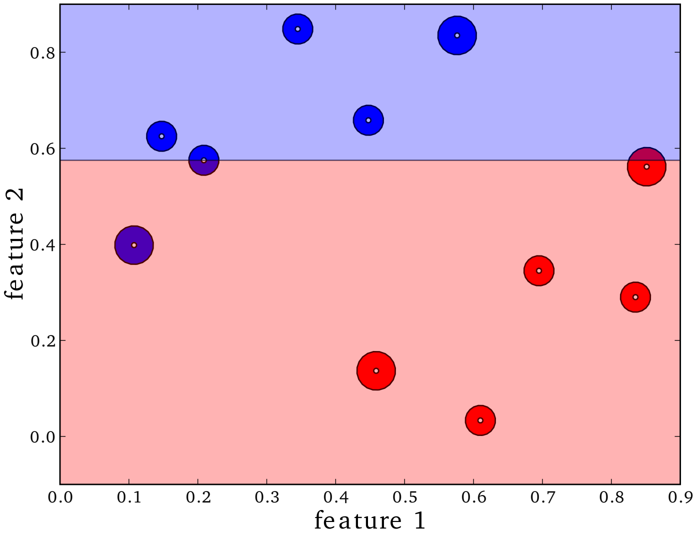


AdaBoost – Example

Iteration $t = 4$

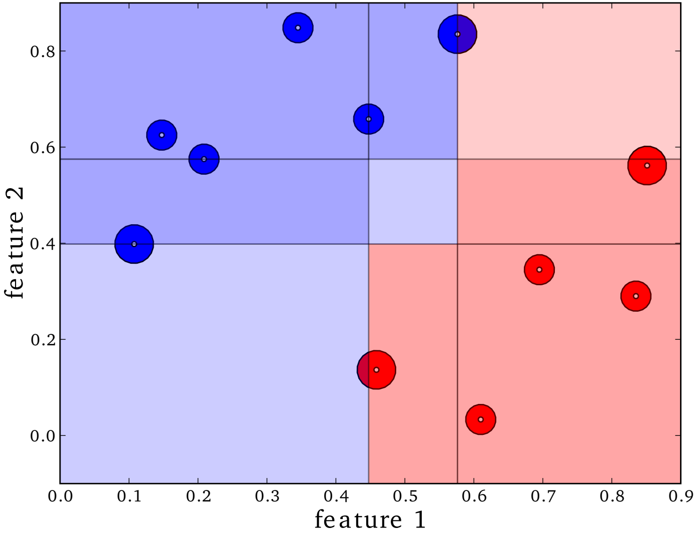


Iteration $t = 5$



AdaBoost – Example

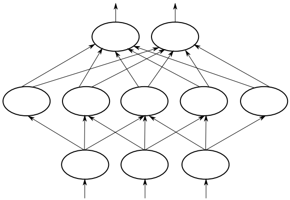
Final classifier: $f(x) = \text{sign}\left(\ 1.15h_1(x) + 1.5h_2(x) + \cdots + 0.9h_5(x)\ \right)$



Artificial Neural Network have been proposed as promising models to achieve artificial intelligence since the 1950s.

Main idea:

- stack layers of simple elements ("neurons")
- one layer's outputs are next layer's input.



Network parametrizes a function:

- each neuron N_i computes a linear/affine function

$$a_i = \langle w_i, x_{\text{input}} \rangle + b_i \qquad \text{"activation"}$$

followed by a componentwise non-linear transformation, $\sigma : \mathbb{R} \to \mathbb{R}$,

$$o_i = \sigma(a_i) \qquad \text{e.g.} \quad \sigma(t) = \mathbf{max}\{0, t\}$$

Training is a bit involved (later...)

Summary

Learning algorithms come in all kind of forms and flavors:

- tree structured, "expert systems"
- similarity-based, geometric
- linear thresholding function
- weighted combinations of simpler units
- iterated/stacked combinations of simpler units

Summary

Learning algorithms come in all kind of forms and flavors:

- tree structured, "expert systems"
- similarity-based, geometric
- linear thresholding function
- weighted combinations of simpler units
- iterated/stacked combinations of simpler units

Machine learning research

- explains their properties
- provides tools to choose between different methods
- allows constructing new ones (with better properties)