Computer Vision

Pattern Recognition Concepts

Luis F. Teixeira
MAP-i | 2014/15

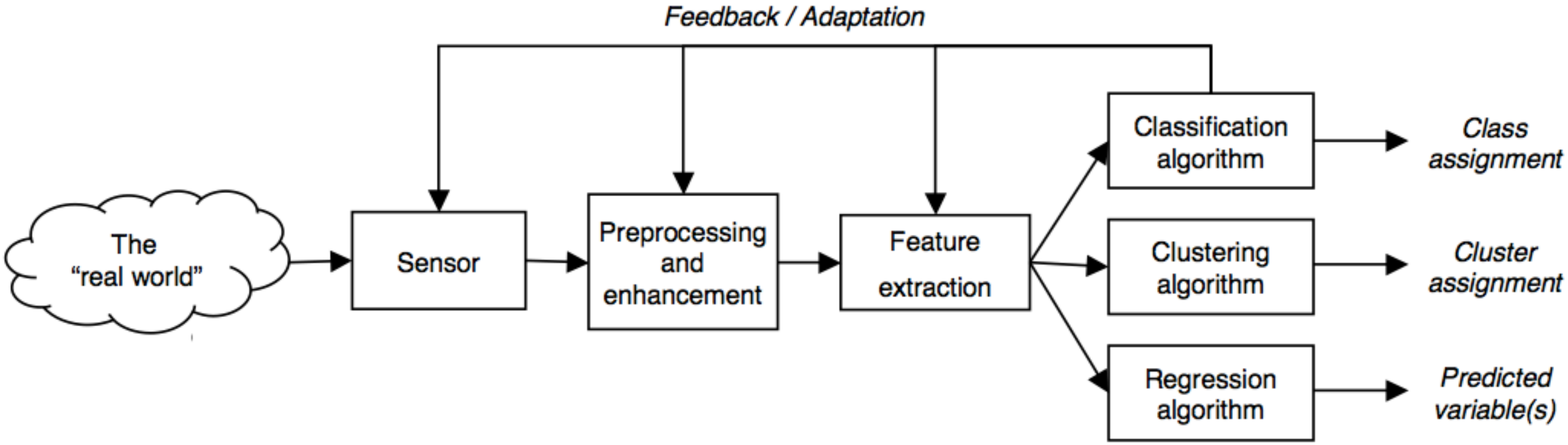
Outline

- General pattern recognition concepts
- Classification
- Classifiers
  - Decision Trees
  - Instance-Based Learning
  - Bayesian Learning
  - Neural Networks
  - Support Vector Machines
  - Model Ensembles

CONCEPTS

Pattern Recognition System

- **A typical pattern recognition system contains**
  - A sensor
  - A preprocessing mechanism
  - A feature extraction mechanism (manual or automated)
  - A classification or description algorithm
  - A set of examples (training set) already classified or described



Pattern Recognition

- Tens of thousands of pattern recognition / machine learning algorithms
- Hundreds new every year
- Every algorithm has three components:
  - **Representation**
  - **Evaluation**
  - **Optimization**

Representation

- Decision trees
- Sets of rules / Logic programs
- Instances
- Graphical models (Bayes/Markov nets)
- Neural networks
- Support vector machines
- Model ensembles
- Etc.

Evaluation

- Accuracy
- Precision and recall
- Squared error
- Likelihood
- Posterior probability
- Cost / Utility
- Margin
- Entropy
- K-L divergence
- Etc.

Optimization

- Combinatorial optimization
  - E.g.: Greedy search
- Convex optimization
  - E.g.: Gradient descent
- Constrained optimization
  - E.g.: Linear programming

Pattern Recognition

- Understanding domain, prior knowledge, and goals
- Data integration, selection, cleaning, pre-processing, etc.
- Learning models
- Interpreting results
- Consolidating and deploying discovered knowledge
- Loop

Tools

- OpenCV
  - http://opencv.org/
- WEKA
  - http://www.cs.waikato.ac.nz/ml/weka/
- RapidMiner
  - http://rapid-i.com/content/view/181/190/

Algorithms

- Classification
  - **Supervised**, **categorical** labels
  - Bayesian classifier, KNN, SVM, Decision Tree, Neural Network, etc.
- Clustering
  - **Unsupervised**, **categorical** labels
  - Mixture models, K-means clustering, Hierarchical clustering, etc.
- Regression
  - **Supervised or Unsupervised**, **real**-valued labels

Algorithms

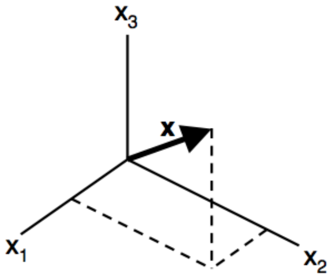
- Classification
  - **Supervised**, **categorical** labels
  - Bayesian classifier, KNN, SVM, Decision Tree, Neural Network, etc.
- Clustering
  - Unsupervised, categorical labels
  - Mixture models, K-means clustering, Hierarchical clustering, etc.
- Regression
  - -Supervised or Unsupervised, real-valued labels

Concepts

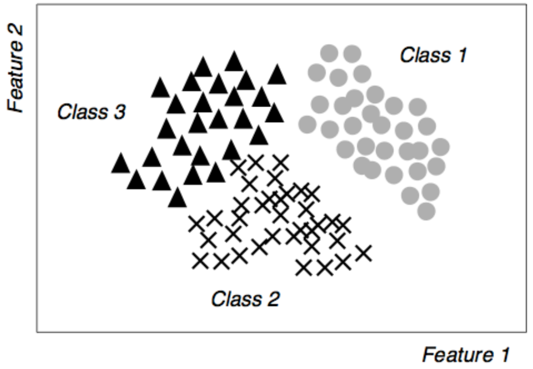
- **Feature**
  - A feature is any distinctive aspect, quality or characteristic. Features may be symbolic (i.e., color) or numeric (i.e., height)
  - The combination of *d* features is represented as a *d*-dimensional column vector called a **feature vector**
    - The d-dimensional space defined by the feature vector is called **feature space**
    - Objects are represented as points in a feature space. This representation is called a **scatter plot**

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \\ x_d \end{bmatrix}$$

Feature vector



Feature space (3D)



Scatter plot (2D)

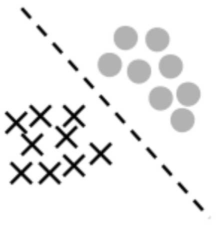
Concepts

- **Pattern**
  - Pattern is a composite of traits or features characteristic of an individual
  - In **classification**, a pattern is a pair of variables {**x**,ω} where
    - **x** is a collection of observations or features (feature vector)
    - ω is the concept behind the observation (label)

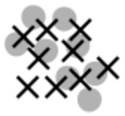
- **What makes a "good" feature vector?**
  - The quality of a feature vector is related to its ability to discriminate examples from different classes
    - Examples from the same class should have similar feature values
    - Examples from different classes have different feature values

Concepts

- “Good” features?

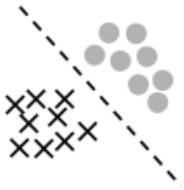


“Good” features

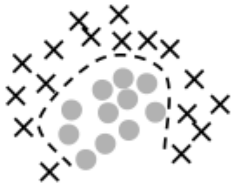


“Bad” features

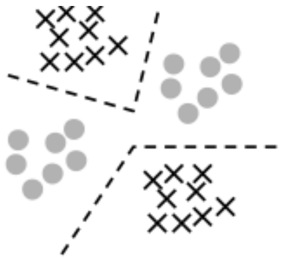
- Feature properties



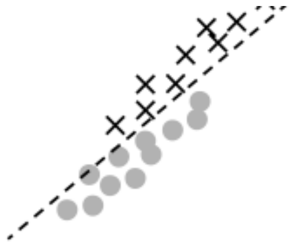
Linear separability



Non-linear separability



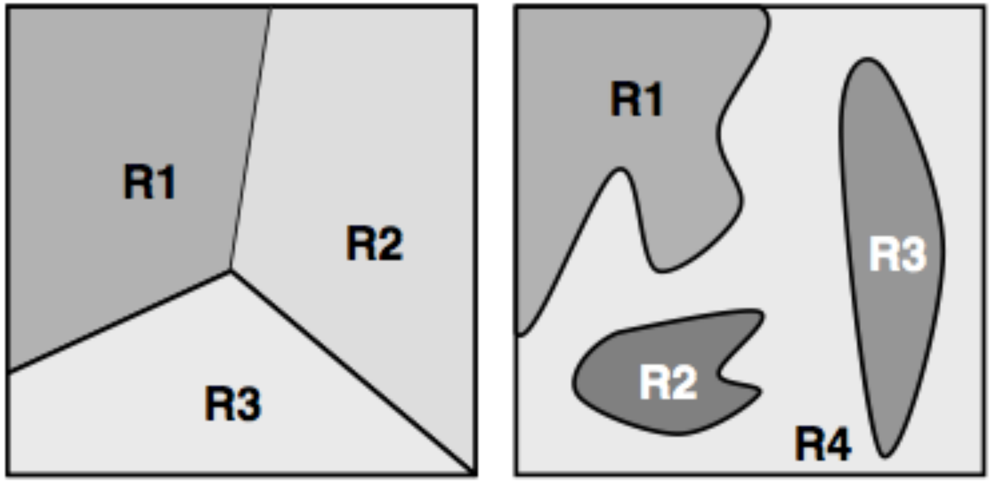
Multi-modal



Highly correlated features

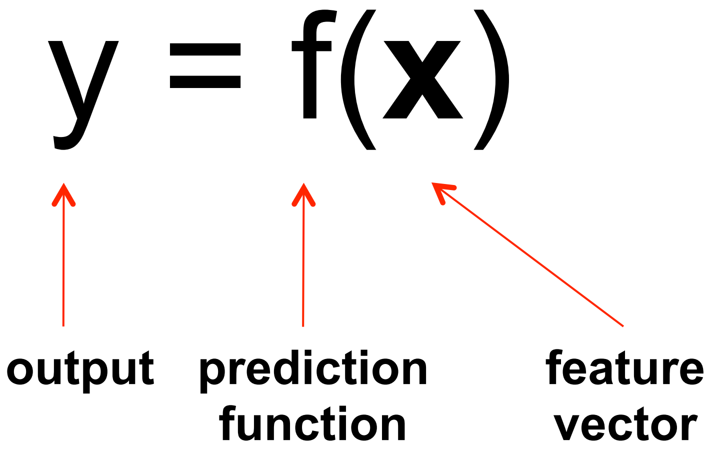
Concepts

- **Classifiers**
  - The goal of a classifier is to partition the feature space into class-labeled **decision regions**
  - Borders between decision regions are called **decision boundaries**



CLASSIFICATION

Classification



$$y = f(\mathbf{x})$$

output

prediction function

feature vector

- **Training:** given a *training set* of labeled examples $\{(\mathbf{x}_1,y_1), \ldots, (\mathbf{x}_N,y_N)\}$, estimate the prediction function f by minimizing the prediction error on the training set
- **Testing:** apply f to a never before seen *test example* **x** and output the predicted value $y = f(\mathbf{x})$

Classification

- Given a collection of *labeled* examples, come up with a function that will predict the labels of new examples.

Training examples

Novel input

- How good is some function we come up with to do the classification?
- Depends on
  - Mistakes made
  - Cost associated with the mistakes

An example*

- **Problem**: sorting incoming fish on a conveyor belt according to species

- Assume that we have only two kinds of fish:
  - Salmon
  - Sea bass

Picture taken with a camera

Adapted from Duda, Hart and Stork, Pattern Classification, 2nd Ed.

An example: the problem

What ***humans*** see

0	3	2	5	4	7	6	9	8
3	0	1	2	3	4	5	6	7
2	1	0	3	2	5	4	7	6
5	2	3	0	1	2	3	4	5
4	3	2	1	0	3	2	5	4
7	4	5	2	3	0	1	2	3
6	5	4	3	2	1	0	3	2
9	6	7	4	5	2	3	0	1
8	7	6	5	4	3	2	1	0

What ***computers*** see

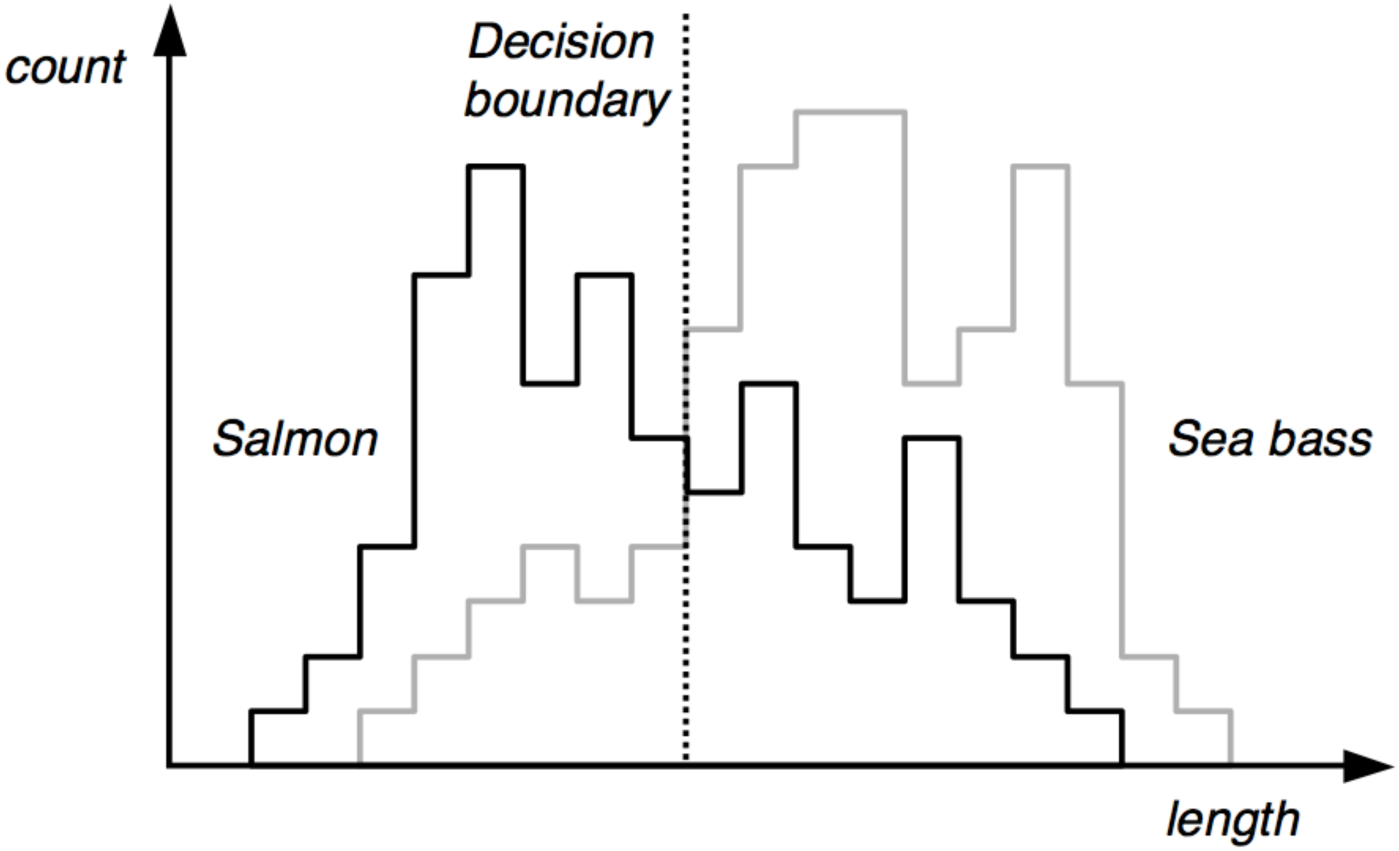
An example: decision process

- What kind of information can distinguish one species from the other?
  - Length, width, weight, number and shape of fins, tail shape, etc.
- What can cause problems during sensing?
  - Lighting conditions, position of fish on the conveyor belt, camera noise, etc.
- What are the steps in the process?
  - Capture image -> isolate fish -> take measurements -> make decision

An example: our system

- **Sensor**
  - The camera captures an image as a new fish enters the sorting area
- **Preprocessing**
  - Adjustments for average intensity levels
  - Segmentation to separate fish from background
- **Feature Extraction**
  - Assume a fisherman told us that a sea bass is generally longer than a salmon. We can use **length** as a feature and decide between sea bass and salmon according to a threshold on length.
- **Classification**
  - Collect a set of examples from both species
    - Plot a distribution of lengths for both classes
  - Determine a decision boundary (threshold) that minimizes the classification error

An example: features

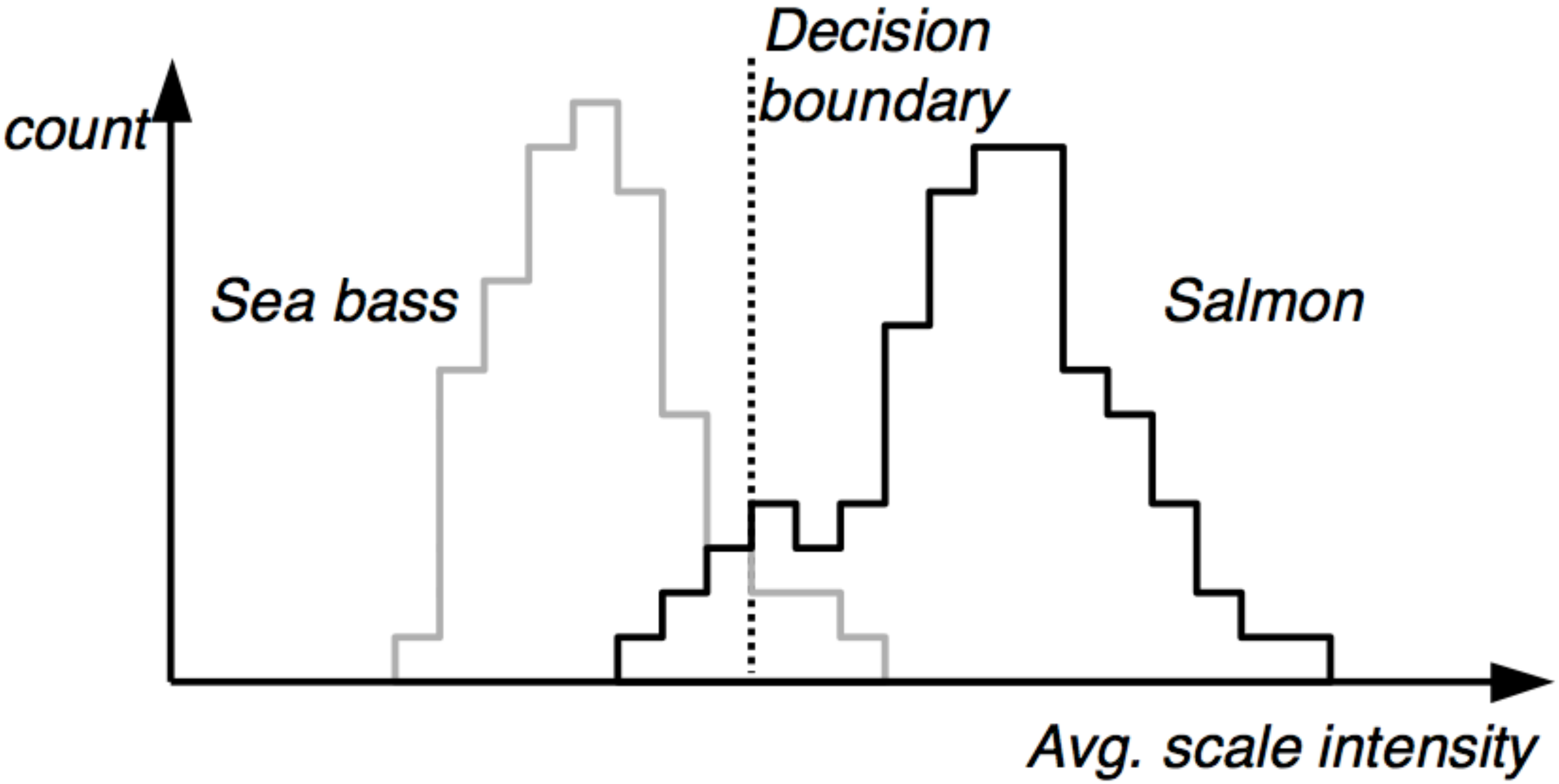


We estimate the system's probability of error and obtain a discouraging result of 40%. Can we improve this result?

An example: features

- Even though sea bass is longer than salmon on the average, there are many examples of fish where this observation does not hold
- Committed to achieve a higher recognition rate, we try a number of features
  - Width, Area, Position of the eyes w.r.t. mouth...
  - only to find out that these features contain no discriminatory information
- Finally we find a “good” feature: **average intensity of the scales**

An example: features



Histogram of the lightness feature for two types of fish in **training samples**. It looks easier to choose the threshold but we still can not make a perfect decision.

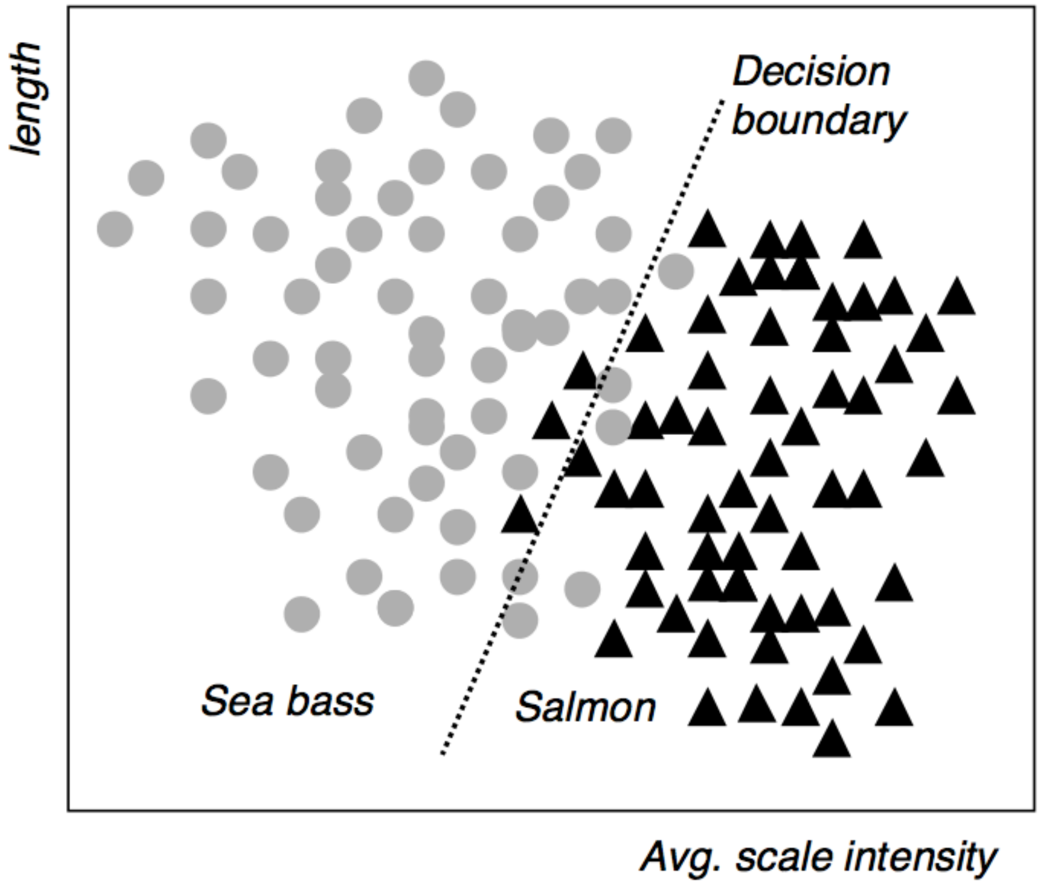
An example: multiple features

- We can use two features in our decision:
  - lightness: $\boldsymbol{x}_1$
  - length: $\boldsymbol{x}_2$
- Each fish image is now represented as a point (feature vector)

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

in a two-dimensional **feature space**.

An example: multiple features

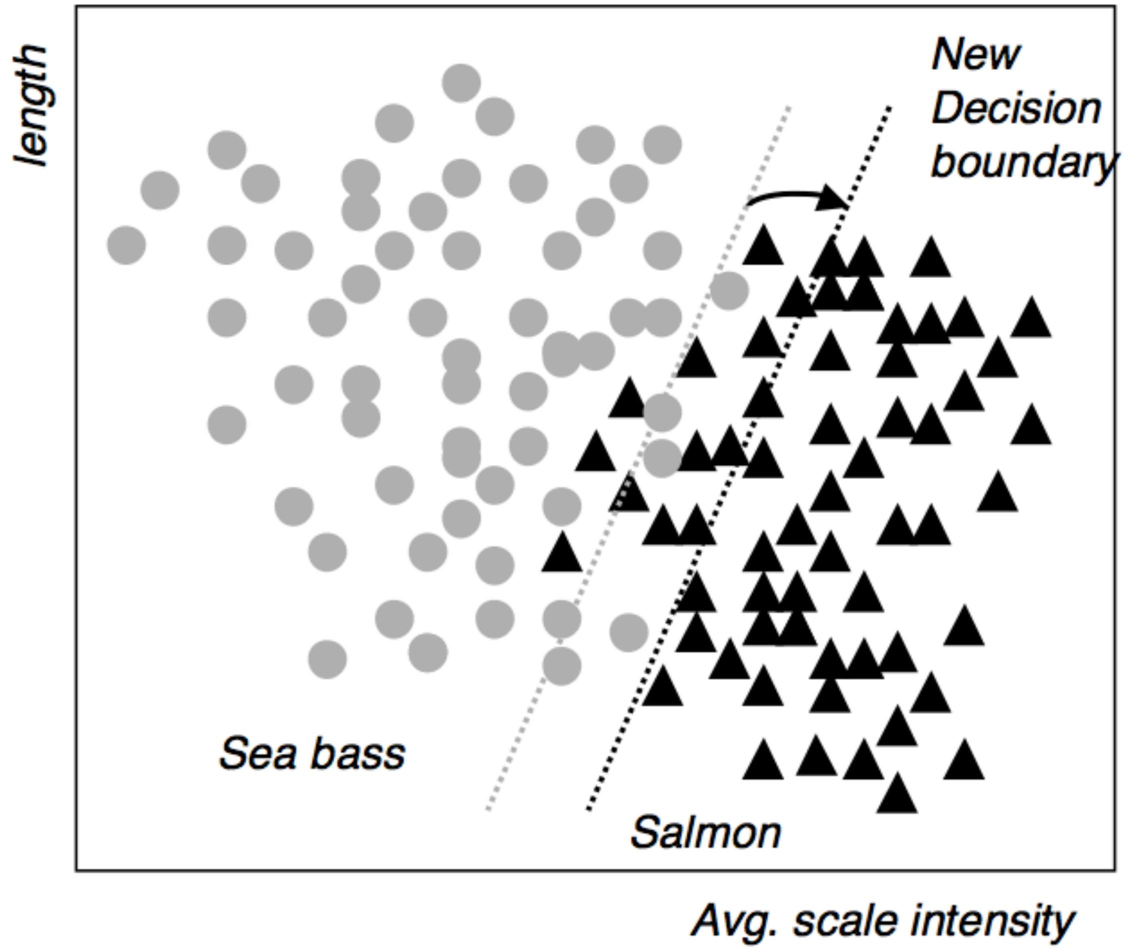


Scatter plot of lightness and length features for training samples. We can compute a **decision boundary** to divide the feature space into two regions with a classification rate of 95.7%.

An example: cost of error

- We should also consider **costs of different errors** we make in our decisions.
- For example, if the fish packing company knows that:
  - Customers who buy salmon will object vigorously if they see sea bass in their cans.
  - Customers who buy sea bass will not be unhappy if they occasionally see some expensive salmon in their cans.
- How does this knowledge affect our decision?

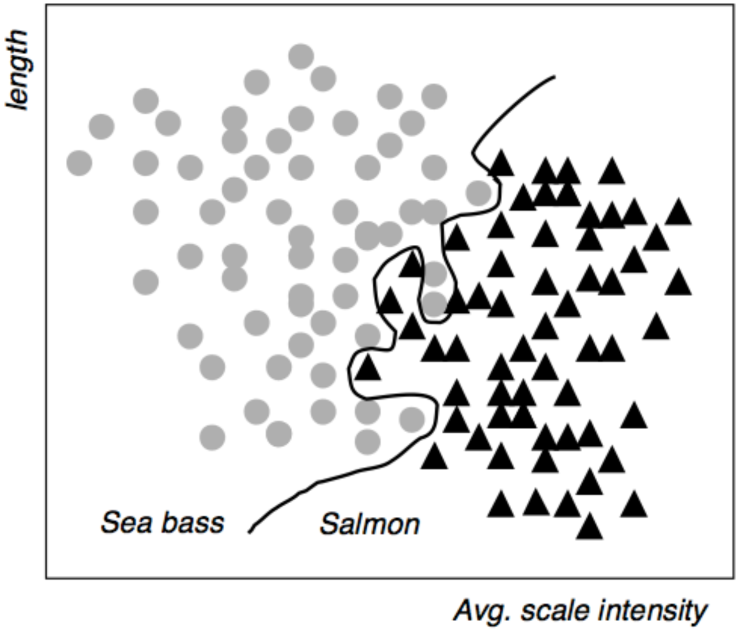
An example: cost of error



We could intuitively shift the decision boundary to minimize an alternative cost function

An example: generalization

- **The issue of generalization**
  - The recognition rate of our linear classifier (95.7%) met the design specifications, but we still think we can improve the performance of the system
  - We then design a über-classifier that obtains an impressive classification rate of 99.9975% with the following decision boundary



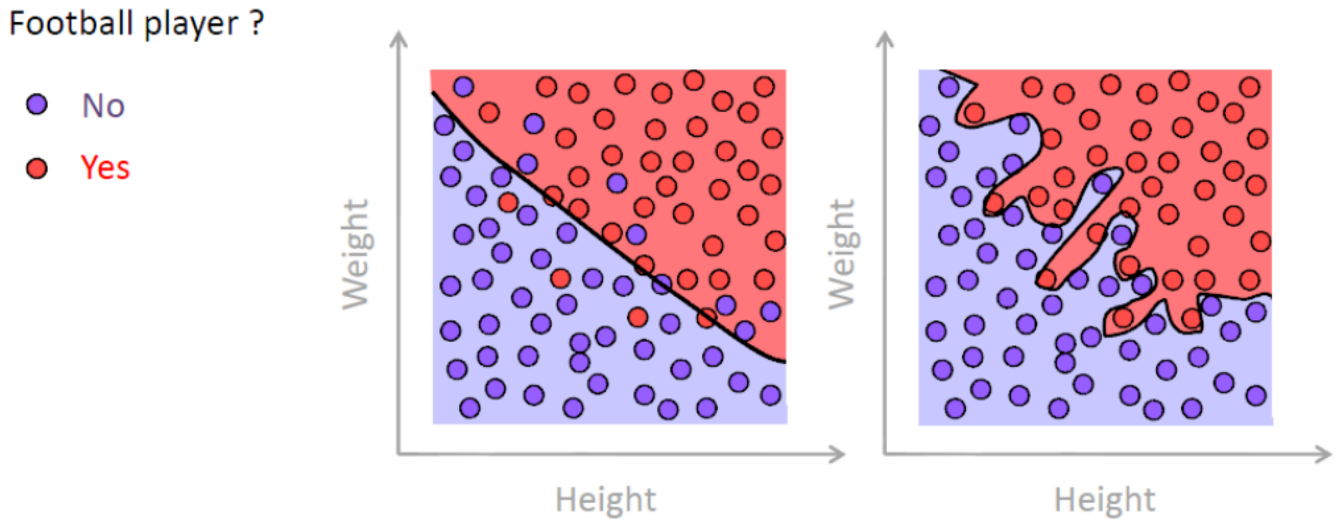
An example: generalization

- **The issue of generalization**
  - Satisfied with our classifier, we integrate the system and deploy it to the fish processing plant
  - A few days later the plant manager calls to complain that the system is misclassifying an average of 25% of the fish

- **What went wrong?**

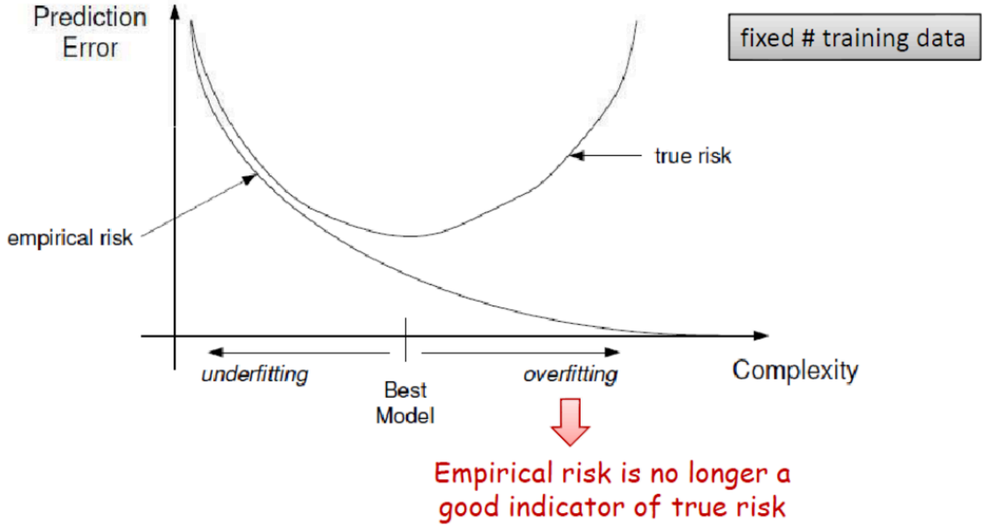
Overfitting

- If we allow very complicated classifiers, we could overfit the training data



Overfitting

- If we allow very complicated classifiers, we could overfit the training data



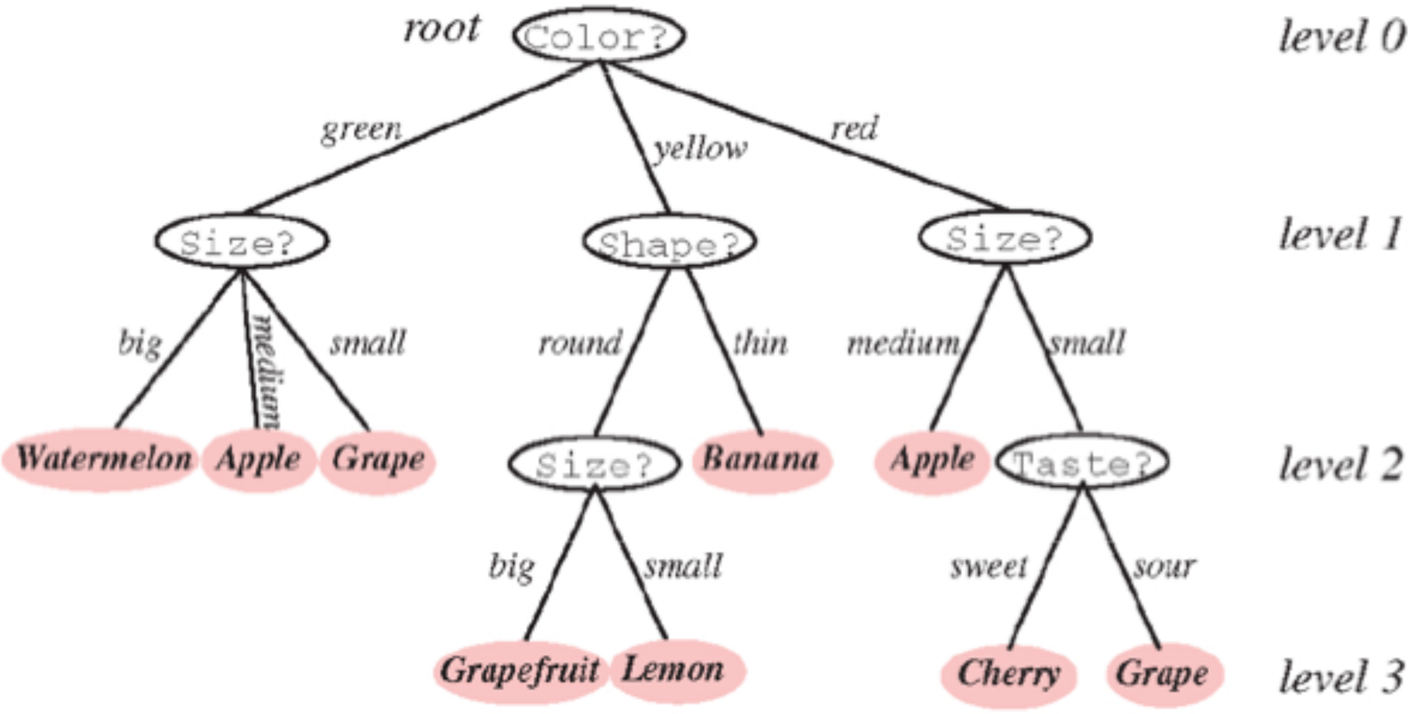
- **Empirical risk** is the performance on the training data – proportion of misclassified examples
- **True risk** is the performance in a random test point – proportion of misclassification

DECISION TREES

Decision trees

- Decision trees are **hierarchical** decision systems in which conditions are sequentially tested until a class is accepted
- The feature space is **split** into unique regions corresponding to the classes, in a **sequential** manner
- The searching of the region to which the feature vector will be assigned to is achieved via a **sequence of decisions** along a path of nodes

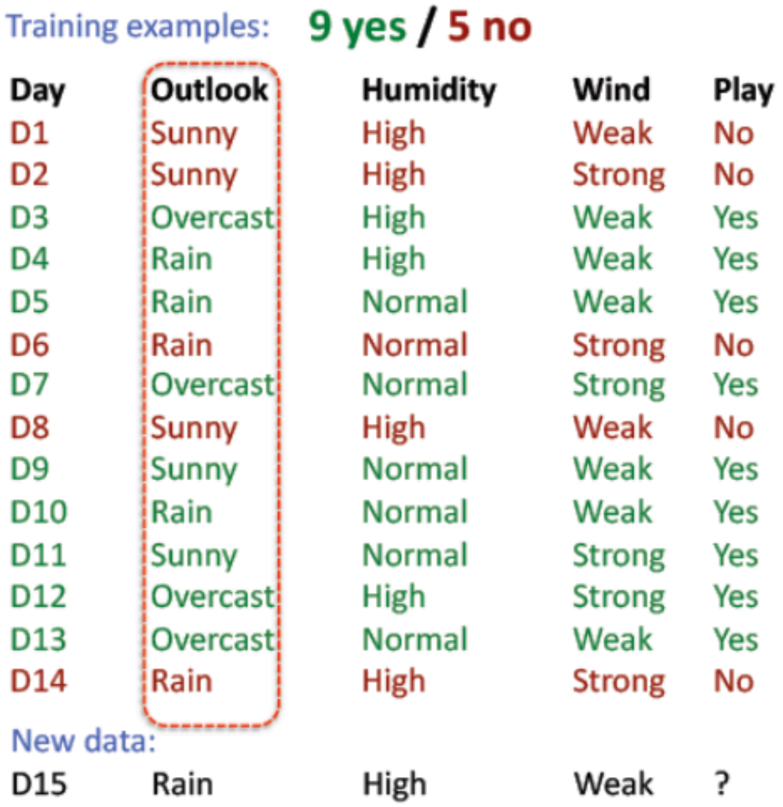
Decision trees



Decision trees classify a pattern through a **sequence** of questions, in which the next question depends on the answer to the current question

Decision trees

- Example: predict if John will play tennis
  - – Divide & conquer:
    - Split into subsets
    - Are they pure? (all yes or all no)
    - If yes: stop
    - If not: repeat
  - – See which subset new data falls into



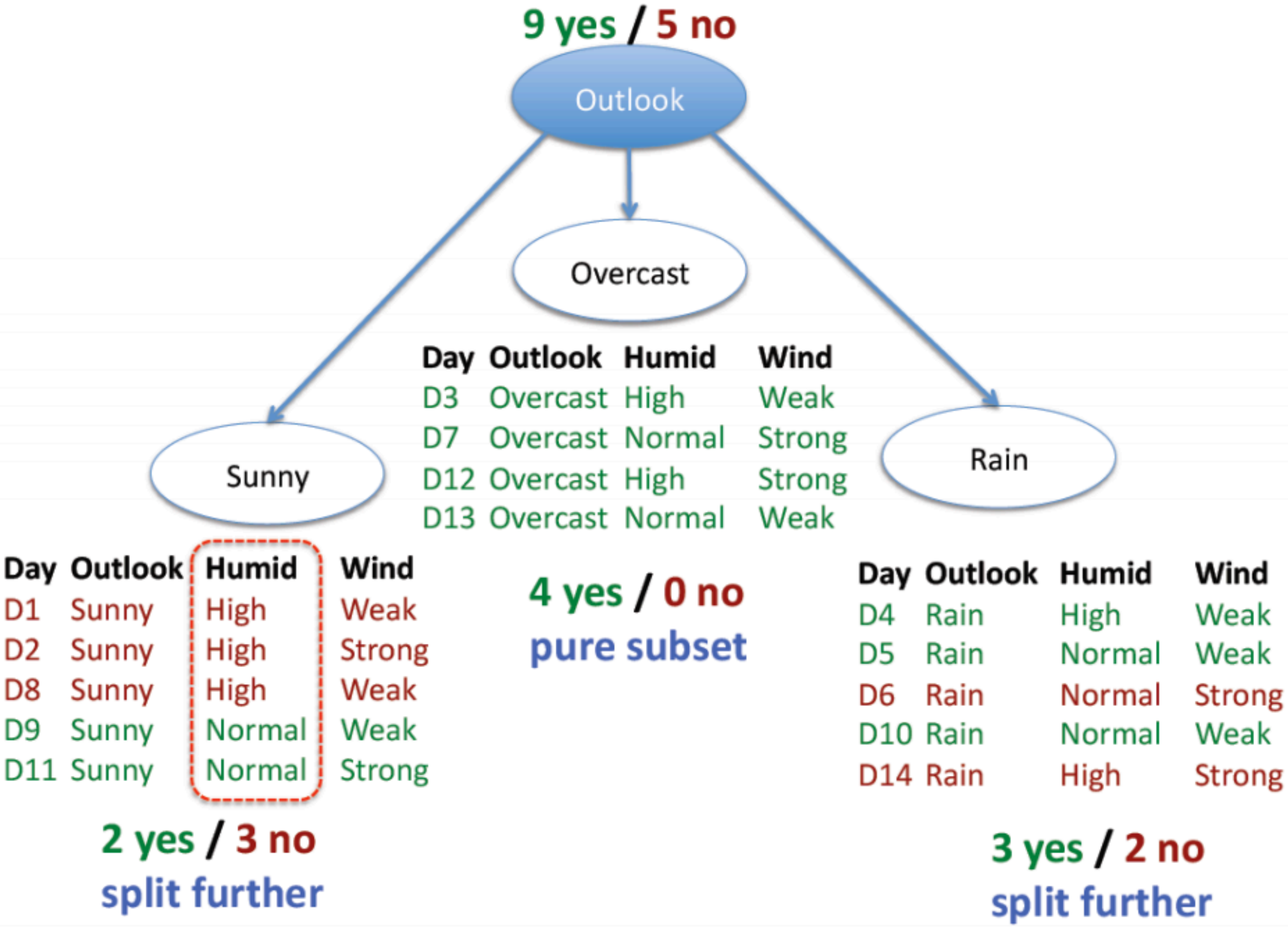
Training examples: **9 yes / 5 no**

Day	Outlook	Humidity	Wind	Play
D1	Sunny	High	Weak	No
D2	Sunny	High	Strong	No
D3	Overcast	High	Weak	Yes
D4	Rain	High	Weak	Yes
D5	Rain	Normal	Weak	Yes
D6	Rain	Normal	Strong	No
D7	Overcast	Normal	Strong	Yes
D8	Sunny	High	Weak	No
D9	Sunny	Normal	Weak	Yes
D10	Rain	Normal	Weak	Yes
D11	Sunny	Normal	Strong	Yes
D12	Overcast	High	Strong	Yes
D13	Overcast	Normal	Weak	Yes
D14	Rain	High	Strong	No

New data:

D15	Rain	High	Weak	?

Decision trees



9 yes / 5 no

Day	Outlook	Humid	Wind
D3	Overcast	High	Weak
D7	Overcast	Normal	Strong
D12	Overcast	High	Strong
D13	Overcast	Normal	Weak

4 yes / 0 no
pure subset

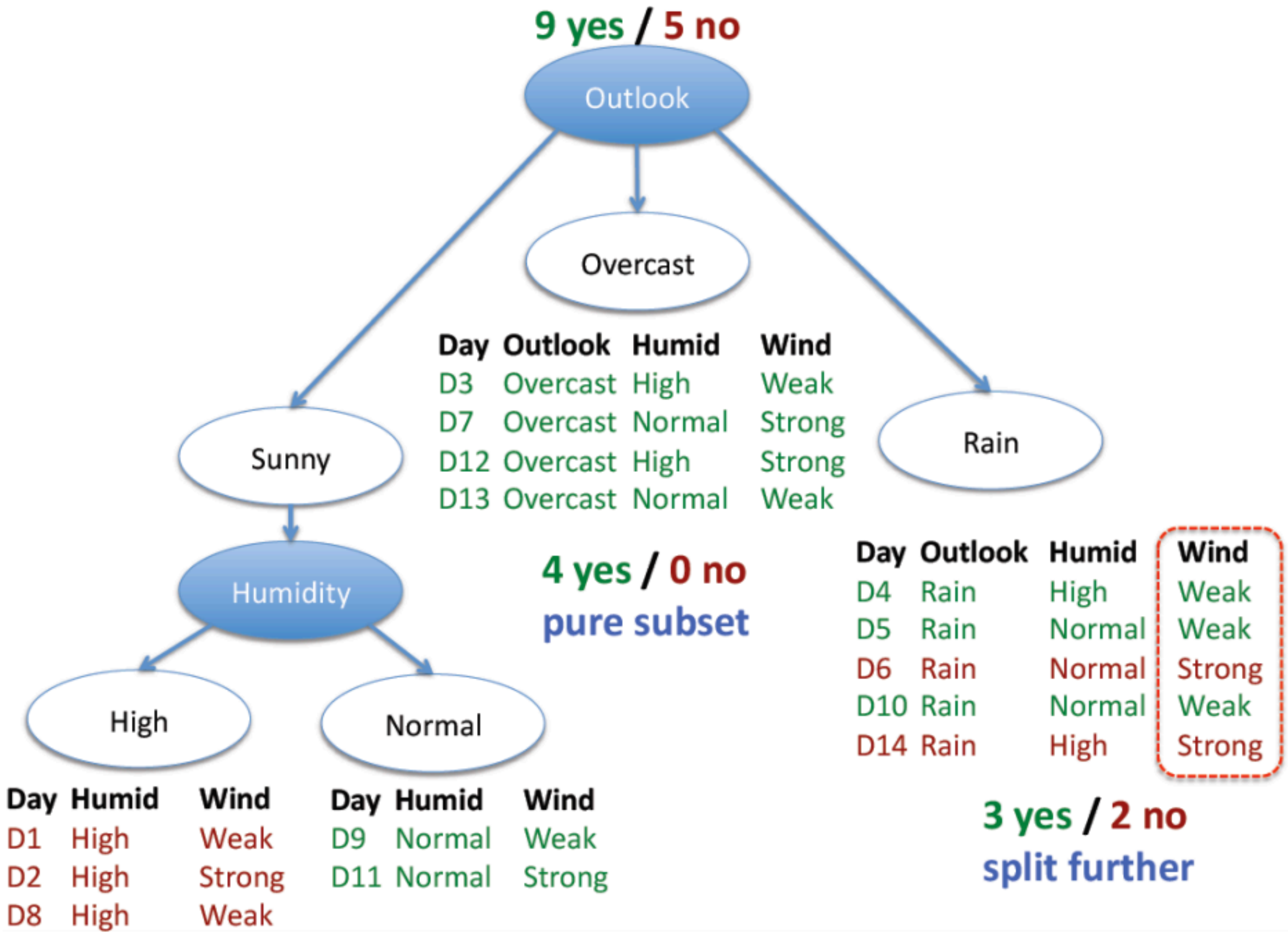
Day	Outlook	Humid	Wind
D1	Sunny	High	Weak
D2	Sunny	High	Strong
D8	Sunny	High	Weak
D9	Sunny	Normal	Weak
D11	Sunny	Normal	Strong

2 yes / 3 no
split further

Day	Outlook	Humid	Wind
D4	Rain	High	Weak
D5	Rain	Normal	Weak
D6	Rain	Normal	Strong
D10	Rain	Normal	Weak
D14	Rain	High	Strong

3 yes / 2 no
split further

Decision trees



9 yes / 5 no

Outlook

Overcast

Day	Outlook	Humid	Wind
D3	Overcast	High	Weak
D7	Overcast	Normal	Strong
D12	Overcast	High	Strong
D13	Overcast	Normal	Weak

4 yes / 0 no

pure subset

Sunny

Humidity

High

Day	Humid	Wind
D1	High	Weak
D2	High	Strong
D8	High	Weak

Normal

Day	Humid	Wind
D9	Normal	Weak
D11	Normal	Strong

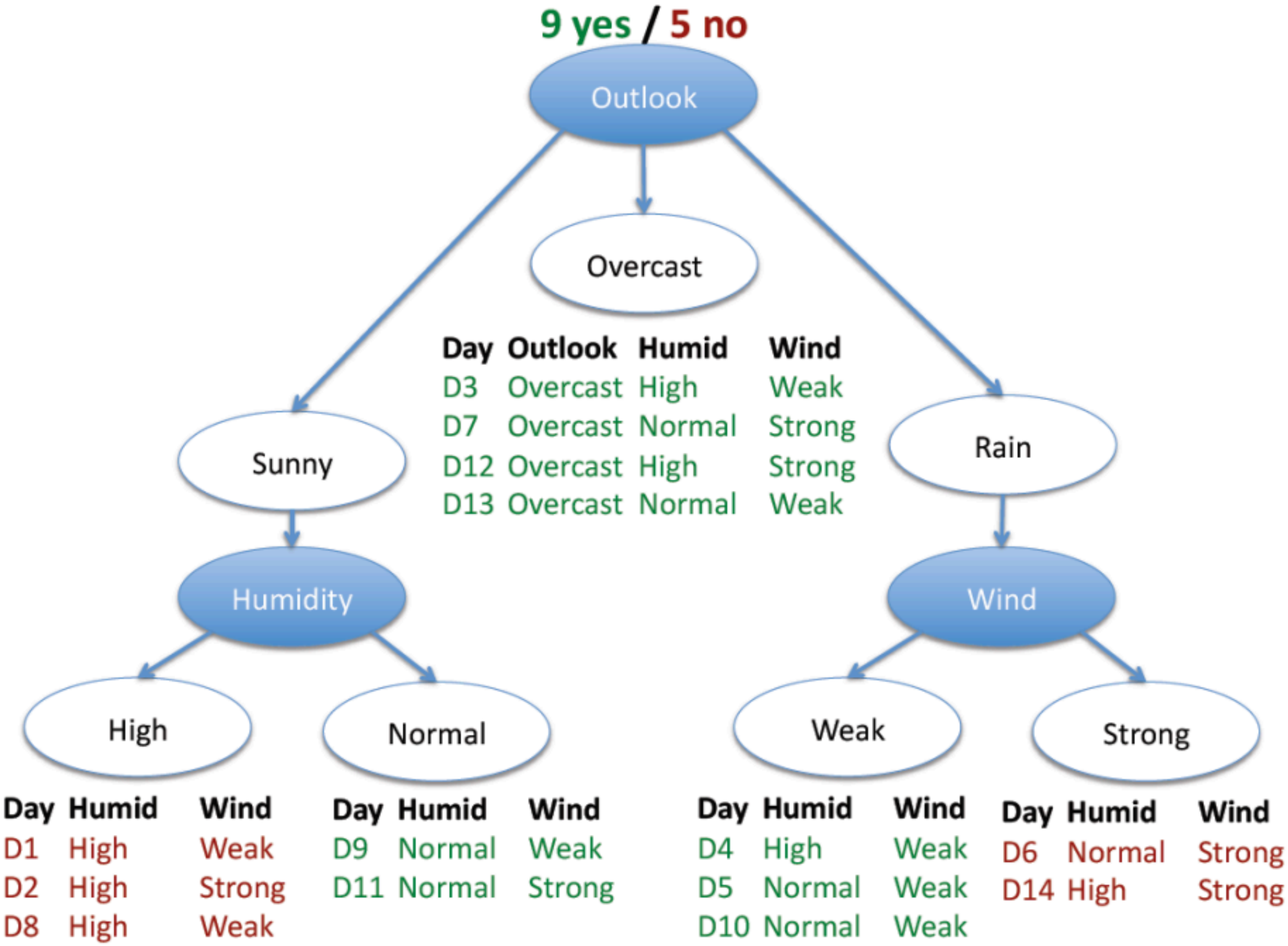
Rain

Day	Outlook	Humid	Wind
D4	Rain	High	Weak
D5	Rain	Normal	Weak
D6	Rain	Normal	Strong
D10	Rain	Normal	Weak
D14	Rain	High	Strong

3 yes / 2 no

split further

Decision trees



Day	Outlook	Humid	Wind
D3	Overcast	High	Weak
D7	Overcast	Normal	Strong
D12	Overcast	High	Strong
D13	Overcast	Normal	Weak

Day	Humid	Wind
D1	High	Weak
D2	High	Strong
D8	High	Weak

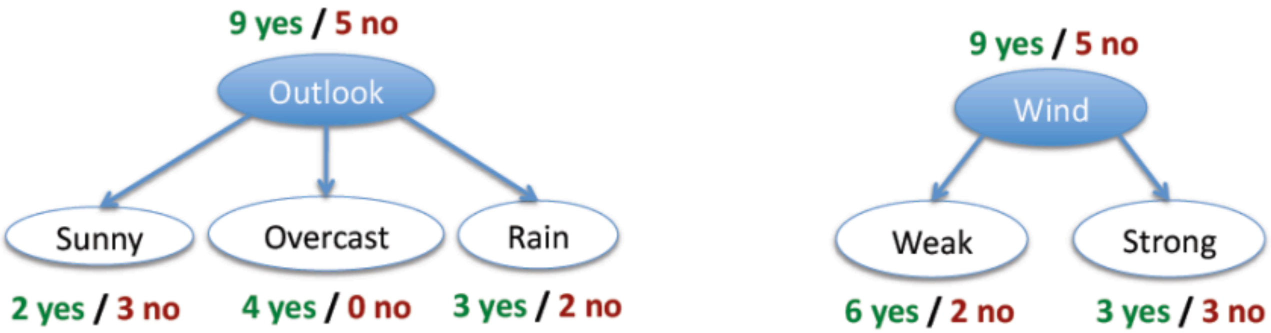
Day	Humid	Wind
D9	Normal	Weak
D11	Normal	Strong

Day	Humid	Wind
D4	High	Weak
D5	Normal	Weak
D10	Normal	Weak

Day	Humid	Wind
D6	Normal	Strong
D14	High	Strong

Decision trees

- Which attribute to split on?



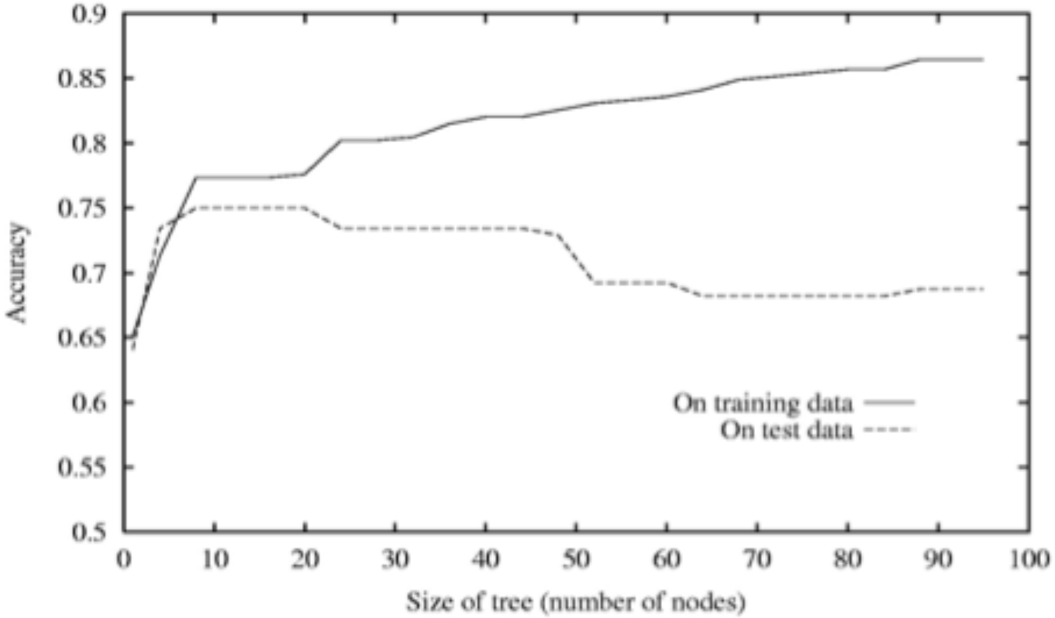
- We want to measure "purity" of the split
  - More certain about Yes/No after the split
    - Pure set (4 yes / 0 no) => completely certain (100%)
    - Impure set (3 yes / 3 no) => completely uncertain (50%)
  - Entropy and Mutual Information measures can be used

Decision trees

- Non-boolean features
  - **Features with multiple discrete values**
    - Construct a multiway split
    - Test for one value versus all others
    - Group values in two disjoint subsets
  - **Real-valued features**
    - Consider a threshold split using each observed value of the feature
    - Mutual information can be used to choose the best split

Decision trees

- Overfitting



- How to avoid?
  - Stop growing when data split not statistically significant
  - Grow full tree, the post-prune (e.g. C4.5, using rule post-pruning)

Decision trees

- Advantages
  - Interpretable: humans can understand decisions
  - Easily handles irrelevant attributes
  - Very compact: #nodes << D after pruning
  - Very fast at testing: O(#nodes)
- Disadvantages
  - Only axis-aligned splits of data
  - Greedy: may not find best tree
    - exponentially many possible trees

INSTANCE-BASED LEARNING

k-Nearest neighbour classifier

- Given the training data D = $\{x_1,...,x_n\}$ as a set of n labeled examples, the **nearest neighbour classifier** assigns a test point x the label associated with its closest neighbour (or k neighbours) in D.

- Closeness is defined using a **distance function**.

Distance functions

- A general class of metrics for *d*-dimensional patterns is the **Minkowski metric**, also known as the L_p norm

$$L_p(\mathbf{x},\mathbf{y}) = \left(\sum_{i=1}^{d} \left| x_i - y_i \right|^p \right)^{1/p}$$

- The **Euclidean distance** is the L_2 norm

$$L_2(\mathbf{x},\mathbf{y}) = \left(\sum_{i=1}^{d} \left| x_i - y_i \right|^2 \right)^{1/2}$$

- The **Manhattan** or **city block distance** is the L1 norm

$$L_1(\mathbf{x},\mathbf{y}) = \sum_{i=1}^{d} \left| x_i - y_i \right|$$

Distance functions

- The **Mahalanobis distance** is based on the covariance of each feature with the class examples.

$$D_M(\mathbf{x}) = \sqrt{\left(\mathbf{x} - \mu\right)^T \Sigma^{-1} \left(\mathbf{x} - \mu\right)}$$

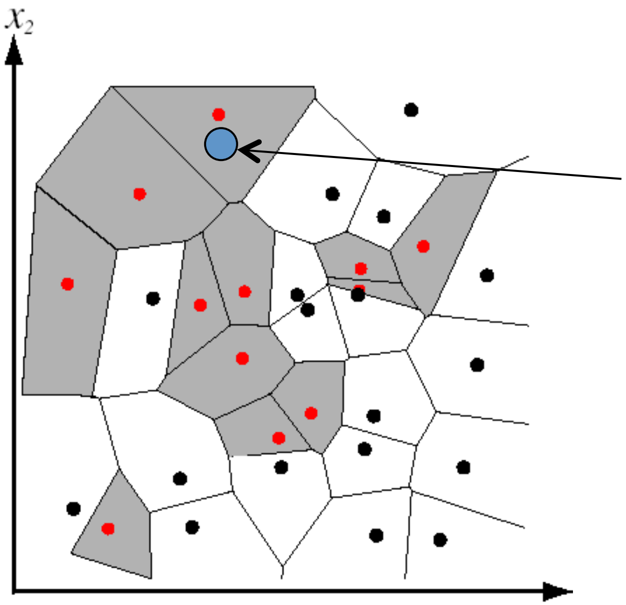
  - Based on the assumption that distances in the direction of high variance are less important
  - Highly dependent on a good estimate of covariance

1-Nearest neighbour classifier

Assign label of nearest training data point to each test data point

Black = negative

Red = positive



from Duda *et al.*

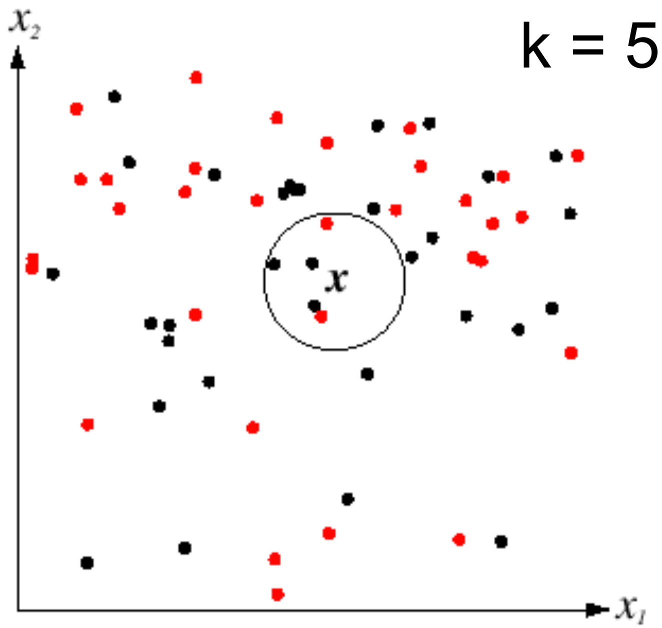
Novel test example

Closest to a positive example from the training set, so classify it as positive.

Voronoi partitioning of feature space
for 2-category 2D data

k-Nearest neighbour classifier

- For a new point, find the k closest points from training data
- Labels of the k points “vote” to classify



k = 5

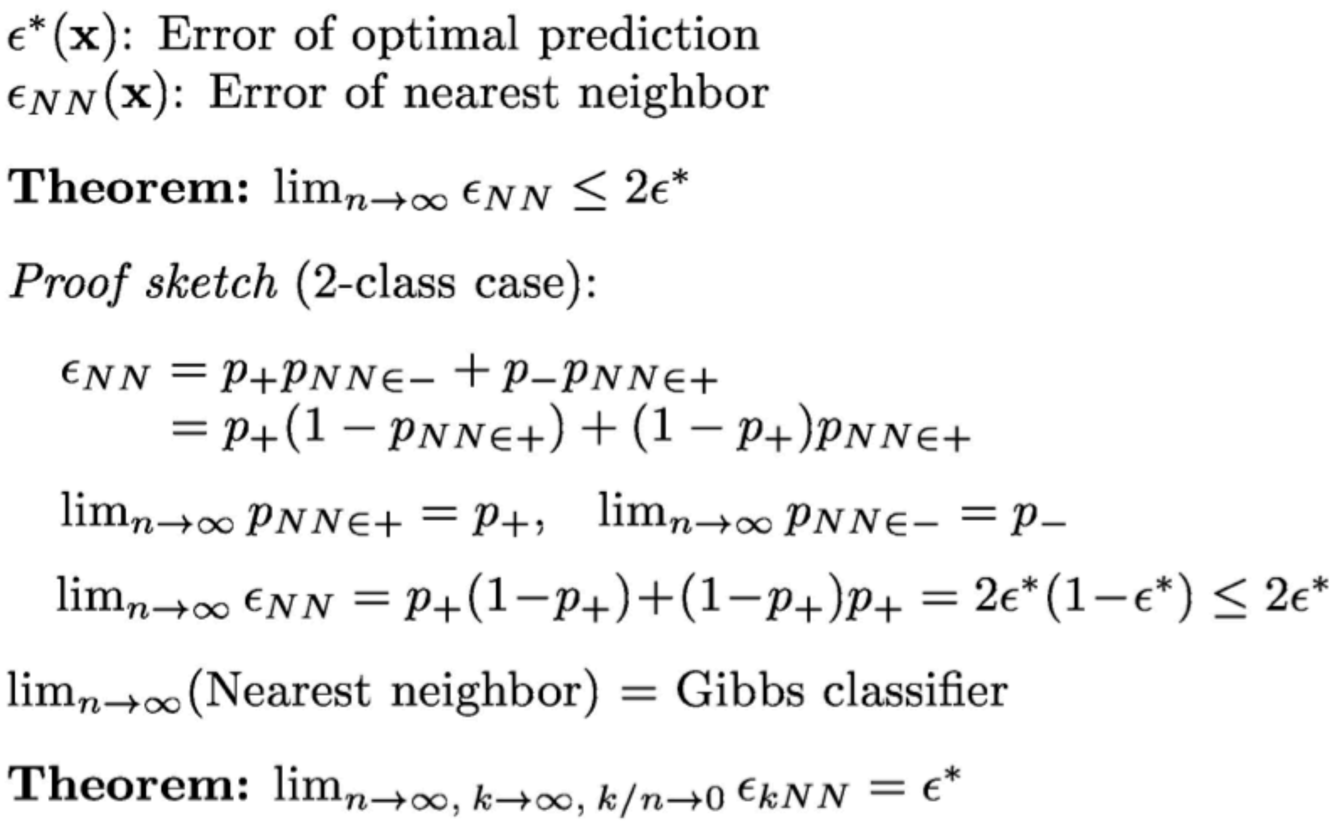
Black = negative

Red = positive

If the query lands here, the 5 NN consist of 3 negatives and 2 positives, so we classify it as negative.

How good is KNN

- In the limit KNN gives the optimal decision



$\epsilon^*(\mathbf{x})$: Error of optimal prediction
$\epsilon_{NN}(\mathbf{x})$: Error of nearest neighbor

Theorem: $\lim_{n\to\infty} \epsilon_{NN} \le 2\epsilon^*$

Proof sketch (2-class case):

$$\begin{aligned}\epsilon_{NN} &= p_+ p_{NN\in -} + p_- p_{NN\in +} \\ &= p_+(1 - p_{NN\in +}) + (1 - p_+) p_{NN\in +}\end{aligned}$$

$$\lim_{n\to\infty} p_{NN\in +} = p_+, \quad \lim_{n\to\infty} p_{NN\in -} = p_-$$

$$\lim_{n\to\infty} \epsilon_{NN} = p_+(1-p_+)+(1-p_+)p_+ = 2\epsilon^*(1-\epsilon^*) \le 2\epsilon^*$$

$\lim_{n\to\infty}$(Nearest neighbor) = Gibbs classifier

Theorem: $\lim_{n\to\infty,\, k\to\infty,\, k/n\to 0} \epsilon_{kNN} = \epsilon^*$

kNN as a classifier

- **Advantages**:
  - Simple to implement
  - Flexible to feature / distance choices
  - Naturally handles multi-class cases
  - Can do well in practice with enough representative data
- **Disadvantages:**
  - Large search problem to find nearest neighbors → Highly susceptible to the **curse of dimensionality**
  - Storage of data
  - Must have a meaningful distance function

Curse of dimensionality

- KNN is easily misled in a high-dimension space
- Why?
  - Easy problems in low-dim are hard in hi-dim
  - Low-dim intuitions do not apply in hi-dim
- Examples
  - Normal distribution
  - Uniform distribution on hypercube
  - Points on hypergrid
  - Approximation of hypersphere by a hypercube
  - Volume of hypersphere

Feature selection

- **Filter approach**
  - Pre-select features individually (e.g. by information gain)
  - Find best transformation that reduces dimensionality (e.g. PCA – Principal Component Analysis)
- **Wrapper approach**
  - Run learner with different combinations of features
    - Forward selection
    - Backward selection
    - Etc.

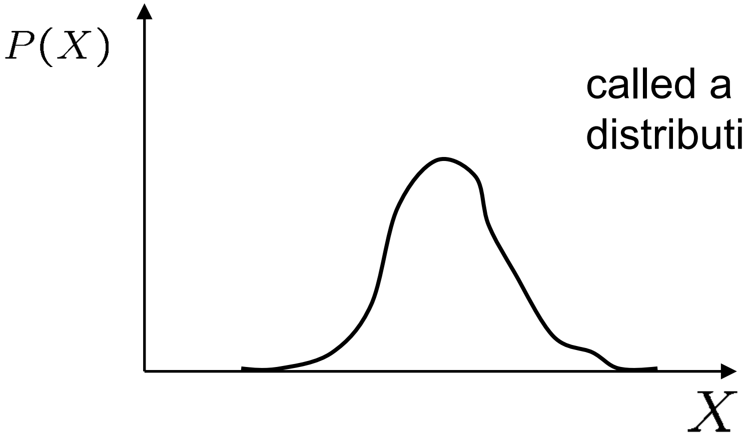
Overfitting in KNN

- How to avoid?
  - Set *k* by cross-validation
  - Form prototypes
  - Remove noisy instances
    - e.g., remove **x** if all **x**'s *k* nearest neighbours are of another class

BAYESIAN LEARNING

Review of probability theory

- Basic probability
  - X is a random variable
  - P(X) is the probability that X achieves a certain value



called a probability distribution/density function (PDF)

$$0 \leq P(X) \leq 1$$

$$\int_{-\infty}^{\infty} P(X) dX = 1$$

continuous X

$$\sum P(X) = 1$$

discrete X

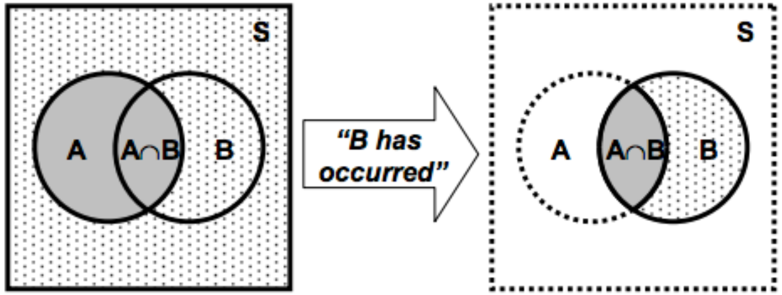
Conditional probability

- If A and B are two events, the probability of event A when we already know that event B has occurred P[A|B] is defined by the relation

$$P[A \mid B] = \frac{P[A \cap B]}{P[B]} \text{ for } P[B] > 0$$

- P[A|B] is read as the "conditional probability of A conditioned on B", or simply the "probability of A given B

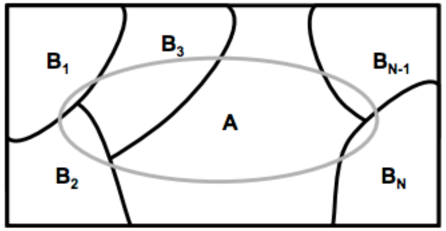
- Graphical interpretation



Conditional probability

- Theorem of Total Probability
  - Let B_1, B_2, …, B_N be mutually exclusive events, then

$$P[A] = P[A \mid B_1]P[B_1] + \ldots + P[A \mid B_N]P[B_N] = \sum_{k=1}^{N} P[A \mid B_k]P[B_k]$$



- Bayes Theorem
  - Given B_1, B_2, …, B_N, a partition of the sample space S. Suppose that event A occurs; what is the probability of event B_j?
  - Using the definition of conditional probability and the Theorem of total probability we obtain

$$P[B_j \mid A] = \frac{P[A \cap B_j]}{P[A]} = \frac{P[A \mid B_j] \cdot P[B_j]}{\sum_{k=1}^{N} P[A \mid B_k] \cdot P[B_k]}$$

Bayes theorem

- For pattern recognition, Bayes Theorem can be expressed as

$$P(\omega_j \mid \mathrm{x}) = \frac{P(\mathrm{x} \mid \omega_j) \cdot P(\omega_j)}{\sum_{k=1}^{N} P(\mathrm{x} \mid \omega_k) \cdot P(\omega_k)} = \frac{P(\mathrm{x} \mid \omega_j) \cdot P(\omega_j)}{P(\mathrm{x})}$$

where ω_j is the j^th class and **x** is the feature vector

- Each term in the Bayes Theorem has a special name
  - P(ω_j) **Prior** probability (of class ω_j)
  - P(ω_j|x) **Posterior** probability (of class ω_j given the observation x)
  - P(x|ω_j) **Likelihood** (conditional prob. of x given class ω_j)
  - P(x) **Evidence** (normalization constant that does not affect the decision)
- Two commonly used decision rules are
  - Maximum A Posteriori (**MAP**): choose the class ω_j with highest P(ω_j|x)
  - Maximum Likelihood (**ML**): choose the class ω_j with highest P(x|ω_j)
  - ML and MAP are equivalent for non-informative priors (P(ω_j) constant)

Bayesian decision theory

- Bayesian Decision Theory is a statistical approach that quantifies the **tradeoffs** between various decisions using **probabilities and costs** that accompany such decisions.

- Fish sorting example:
  - define *C*, the type of fish we observe (state of nature), as a random variable where
    - $C = C_1$ for sea bass
    - $C = C_2$ for salmon
  - $P(C_1)$ is the **a priori probability** that the next fish is a sea bass
  - $P(C_2)$ is the **a priori probability** that the next fish is a salmon

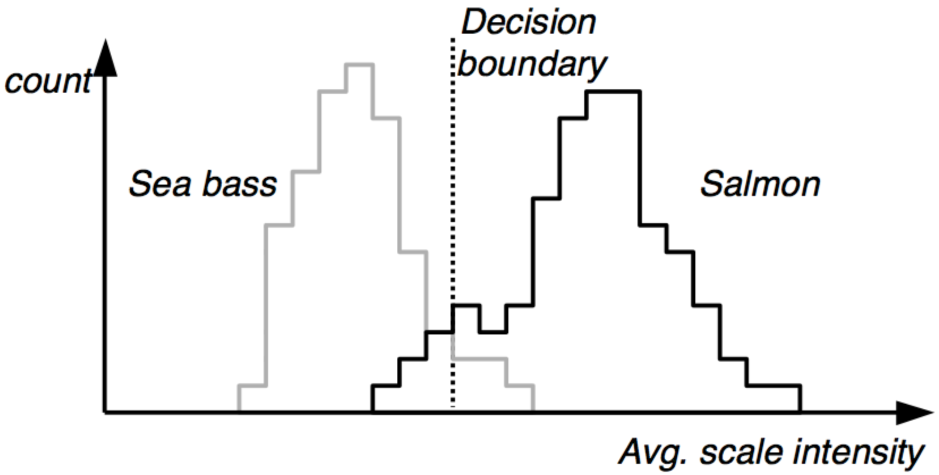
Prior probabilities

- Prior probabilities reflect our knowledge of how likely each type of fish will appear **before** we actually see it.
- How can we choose $P(C_1)$ and $P(C_2)$?
  - Set $P(C_1) = P(C_2)$ if they are equiprobable (uniform priors).
  - May use different values depending on the fishing area, time of the year, etc.
- Assume there are no other types of fish
  - $P(C_1) + P(C_2) = 1$
- In a general classification problem with K classes, **prior probabilities reflect prior expectations** of observing each class and

$$\sum_{i=1}^{K} P(C_i) = 1$$

Class-conditional probabilities

- Let x be a continuous random variable, representing the lightness measurement
- Define p(x|C_j) as the class-conditional probability density or likelihood (probability of x given that the state of nature is C_j for j = 1, 2).
- p(x|C_1) and p(x|C_2) describe the difference in lightness between populations of sea bass and salmon.



Posterior probabilities

- Suppose we know P(C_j) and P(x|C_j) for j = 1, 2, and measure the lightness of a fish as the value x.
- Define P(C_j|x) as the **a posteriori probability** (probability of the type being C_j, given the measurement of feature value x).
- We can use the **Bayes formula** to convert the prior probability to the posterior probability

$$P(C_j \mid x) = \frac{P(x \mid C_j)P(C_j)}{P(x)}$$

$$\text{where } P(x) = \sum_{j=1}^{2} P(x \mid C_j)P(C_j)$$

Making a decision

- How can we make a decision after observing the value of x?

$$\text{Decide}\begin{cases} C_1 & \text{if } P(C_1 \mid x) > P(C_2 \mid x) \\ C_2 & \textit{otherwise} \end{cases}$$

- Rewriting the rule gives

$$\text{Decide}\begin{cases} C_1 & \text{if } \dfrac{P(x \mid C_1)}{P(x \mid C_2)} > \dfrac{P(C_2)}{P(C_1)} \\ C_2 & \textit{otherwise} \end{cases}$$

- Bayes decision rule **minimizes** the error of this decision

Making a decision

- Confusion matrix
  - For C_1 we have:

		Assigned	
		C_1	C_2
True	C_1	correct detection	mis-detection
	C_2	false alarm	correct rejection

  - The two types of errors (false alarm and mis-detection) can have **distinct costs**

Minimum-error-rate classification

- Let {$C_{1,\dots,}$ C_K} be the finite set of *K* states of nature (**classes**, **categories**).
- Let **x** be the D-component vector-valued random variable (**feature vector)**.
- If all errors are equally costly, the minimum-error decision rule is defined as

$$\text{Decide } C_i \text{ if } P(C_i \mid x) > P(C_j \mid x) \quad \forall j \neq i$$

- The resulting error is called the **Bayes error** and is the best performance that can be achieved.

Bayesian decision theory

- Bayesian decision theory gives the **optimal decision** rule under the assumption that the "true" values of the probabilities are **known**.
- But, how can we estimate (learn) the unknown $p(\boldsymbol{x}|C_j)$, $j = 1, \ldots, K$?

- **Parametric models**: assume that the form of the density functions is known
- **Non-parametric models**: no assumption about the form

Bayesian decision theory

- Parametric models
  - Density models (e.g., Gaussian)
  - Mixture models (e.g., mixture of Gaussians)
  - Hidden Markov Models
  - Bayesian Belief Networks

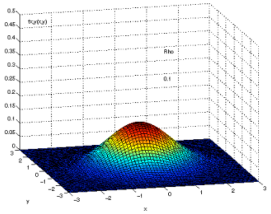
- Non-parametric models
  - Nearest neighbour estimation
  - Histogram-based estimation
  - Parzen window estimation

Gaussian density

- **Gaussian** can be considered as a model where the feature vectors for a given class are continuous-valued, randomly corrupted versions of a single typical or prototype vector.

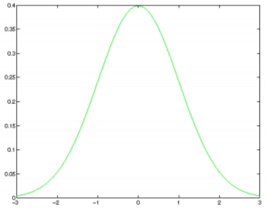
For $\mathbf{x} \in \boldsymbol{R}^D$

$$p(\mathbf{x}) = N(\boldsymbol{\mu}, \Sigma) = \frac{1}{(2\pi)^{n/2} |\Sigma|^{1/2}} e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T \Sigma^{-1} (\mathbf{x}-\boldsymbol{\mu})}$$



For $x \in \boldsymbol{R}$

$$p(\mathbf{x}) = N(\boldsymbol{\mu}, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$



- Some **properties** of the Gaussian:
  - Analytically tractable
  - Completely specified by the 1st and 2nd moments
  - Has the maximum entropy of all distributions with a given mean and variance
  - Many processes are asymptotically Gaussian (Central Limit Theorem)
  - “Uncorrelatedness” implies independence

Bayes linear classifier

- Let us assume that the **class-conditional densities** are **Gaussian** and then explore the resulting form for the posterior probabilities.
- Assume that all classes share the same covariance matrix, thus the density for class C_k is given by

$$p(\mathbf{x} \mid C_k) = \frac{1}{(2\pi)^{D/2} \mid \Sigma \mid^{1/2}} e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_k)^T \Sigma^{-1} (\mathbf{x}-\boldsymbol{\mu}_k)^T}$$

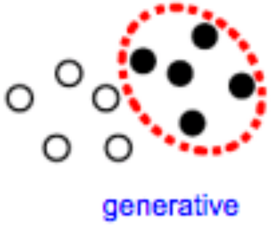
- We then model the class-conditional densities p(**x**|C_k) and class priors p(C_k) and use these to compute **posterior probabilities** p(C_k|**x**) through Bayes' theorem
- The **maximum likelihood** estimates of a Gaussian are

$$\hat{\boldsymbol{\mu}} = \frac{1}{n}\sum_{i=1}^{n} \mathbf{x}_i \text{ and } \hat{\Sigma} = \frac{1}{n}\sum_{i=1}^{n} (\mathbf{x}_i - \hat{\boldsymbol{\mu}})(\mathbf{x}_i - \hat{\boldsymbol{\mu}})^T$$

- Assuming only **2 classes** the **decision boundary is linear**

Naïve Bayes

- A complete probability distribution for each class
  - defines likelihood for any point x
  - can "generate" synthetic observations



$$P(C_i \mid x) \propto P(x \mid C_i)\ P(C_i)$$

Independence assumption

- Compute P ($x_1 \ldots x_n | y$) for every observation $x_1 \ldots x_n$
  - – class-conditional “counts”, based on training data
  - – problem: may not have seen every $x_1 \ldots x_n$ for every y
    - digits: 2400 possible black/white patterns (20x20)
    - spam: every possible combination of words: 2^{10000}
  - – often have observations for individual x_i for every class
- Assume $x_1 \ldots x_n$ conditionally independent given y

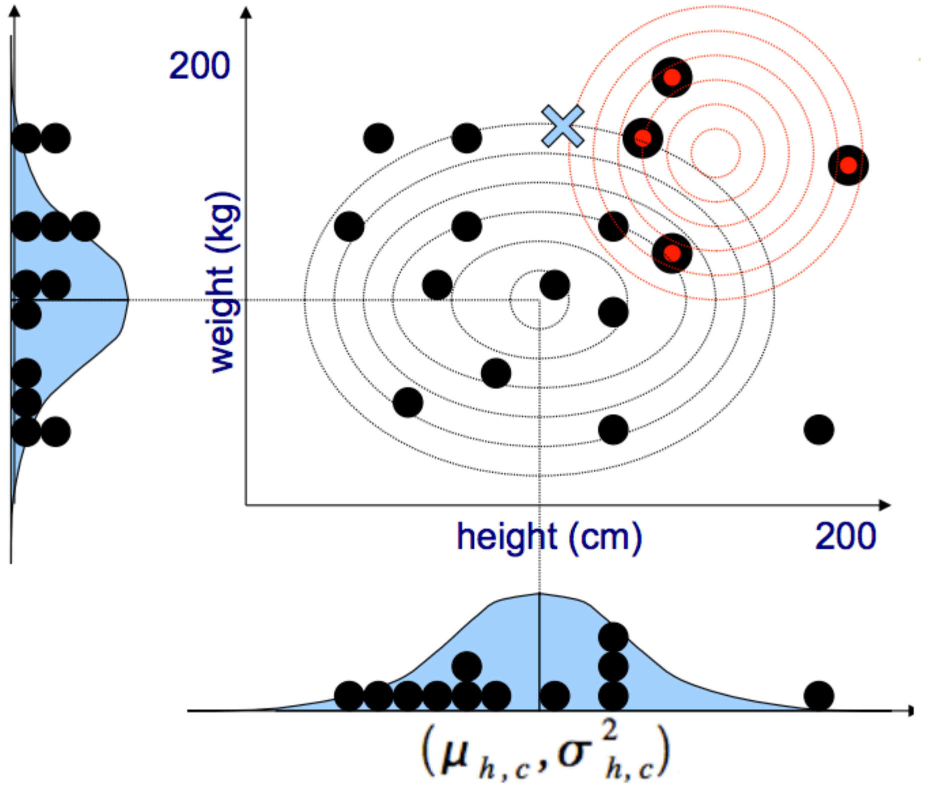
$$P(x_1 \ldots x_n | y) = \underbrace{\prod_{i=1}^{n} P(x_i | x_1 \ldots x_{i-1}, y)}_{\text{chain rule (exact)}} \underbrace{= \prod_{i=1}^{n} P(x_i | y)}_{\text{independence}}$$

Naïve Bayes

- Continuous example
  - Distinguish children from adults based on size
    - classes: $\{a,c\}$, attributes: height [cm], weight [kg]
    - training examples: $\{h_i, w_i, y_i\}$, 4 adults, 12 children
  - Class probabilities: $P(a) = \frac{4}{4+12} = 0.25; P(c) = 0.75$
  - Model for adults:
    - height ~ Gaussian with mean, variance $\begin{cases} \mu_{h,a} = \frac{1}{4} \sum_{i:y_i=a} h_i \\ \sigma^2_{w,a} = \frac{1}{4} \sum_{i:y_i=a} (h_i - \mu_{h,a})^2 \end{cases}$
    - weight ~ Gaussian $\left(\mu_{w,a}, \sigma^2_{w,a}\right)$
    - assume height and weight independent
  - Model for children: same, using $\left(\mu_{h,c}, \sigma^2_{h,c}\right), \left(\mu_{w,c}, \sigma^2_{w,c}\right)$

Naïve Bayes

- Continuous example



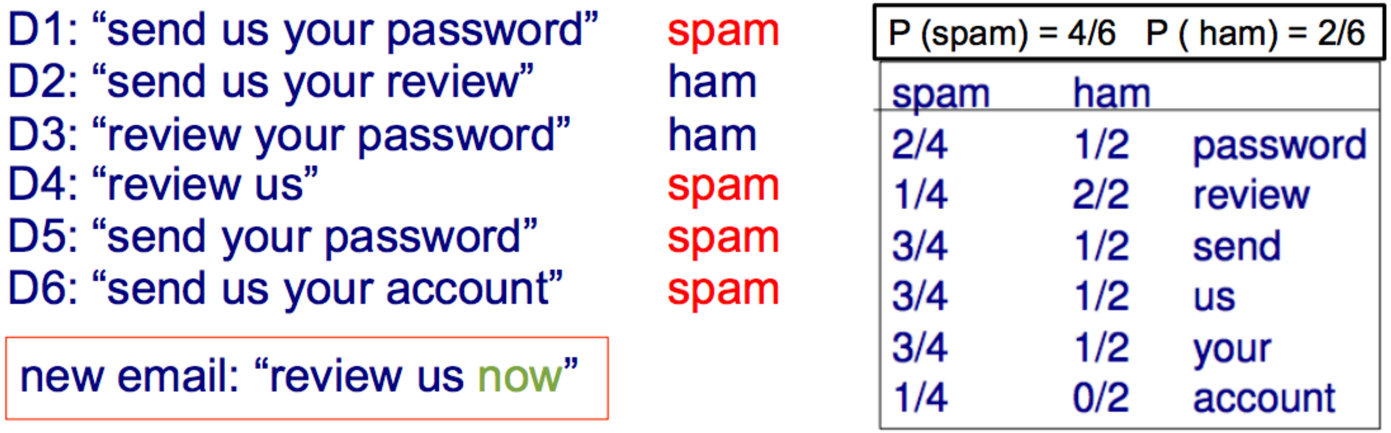
$$P(x|a) = p(h_x|a)p(w_x|a)$$

$$P(x|c) = p(h_x|c)p(w_x|c)$$

$$P(a|x) = \frac{P(x|a)P(a)}{P(x|a)P(a) + P(x|c)P(c)}$$

Naïve Bayes

- Discrete example
  - – Separate spam from valid email (ham)
    - attributes = words



D1: “send us your password”	spam
D2: “send us your review”	ham
D3: “review your password”	ham
D4: “review us”	spam
D5: “send your password”	spam
D6: “send us your account”	spam

new email: “review us now”

P (spam) = 4/6 P (ham) = 2/6

spam	ham	
2/4	1/2	password
1/4	2/2	review
3/4	1/2	send
3/4	1/2	us
3/4	1/2	your
1/4	0/2	account

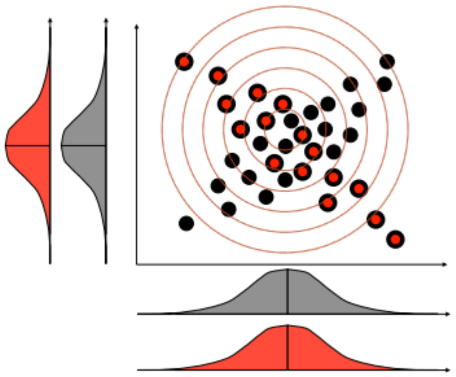
$$P(review\ us|spam) = P(0,1,0,1,0,0|spam) = \left(1-\frac{2}{4}\right)\left(\frac{1}{4}\right)\left(1-\frac{3}{4}\right)\left(\frac{3}{4}\right)\left(1-\frac{3}{4}\right)\left(1-\frac{1}{4}\right)$$

$$P(review\ us|ham) = P(0,1,0,1,0,0|ham) = \left(1-\frac{1}{2}\right)\left(\frac{2}{2}\right)\left(1-\frac{1}{2}\right)\left(\frac{1}{2}\right)\left(1-\frac{1}{2}\right)\left(1-\frac{0}{2}\right)$$

$$P(ham|review\ us) = \frac{0.0625 \times 2/6}{0.0625 \times 2/6 + 0.0044 \times 4/6} = 0.87$$

Naïve Bayes

- Advantages
  - Handles missing data
  - Good computational complexity
  - Incremental updates
- Disadvantages
  - Unable to handle correlated data
  - Problems with repetitions in the discrete case
  - Zero-frequency problem (the training examples may not include enough counts)



NEURAL NETWORKS

Artificial neural networks

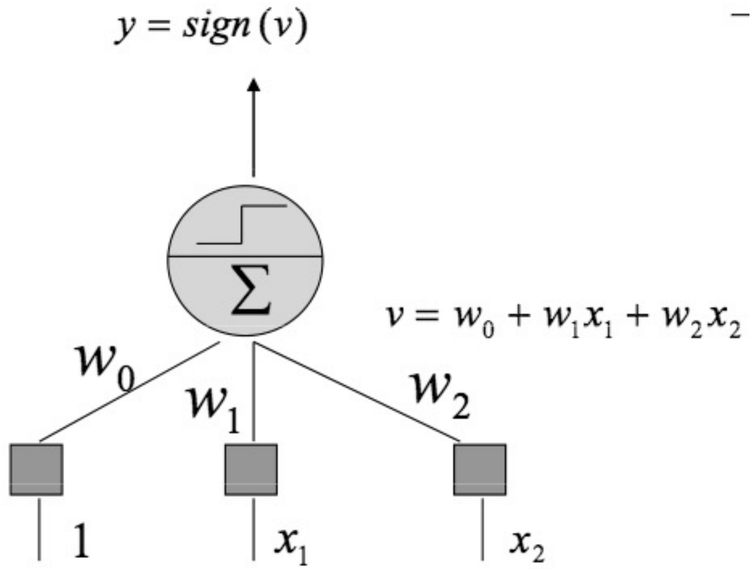
- A neural network is a set of connected input/output units where each connection has a **weight** associated with it
- During the learning phase, the **network learns by adjusting the weights** so as to be able to predict the correct class output of the input signals

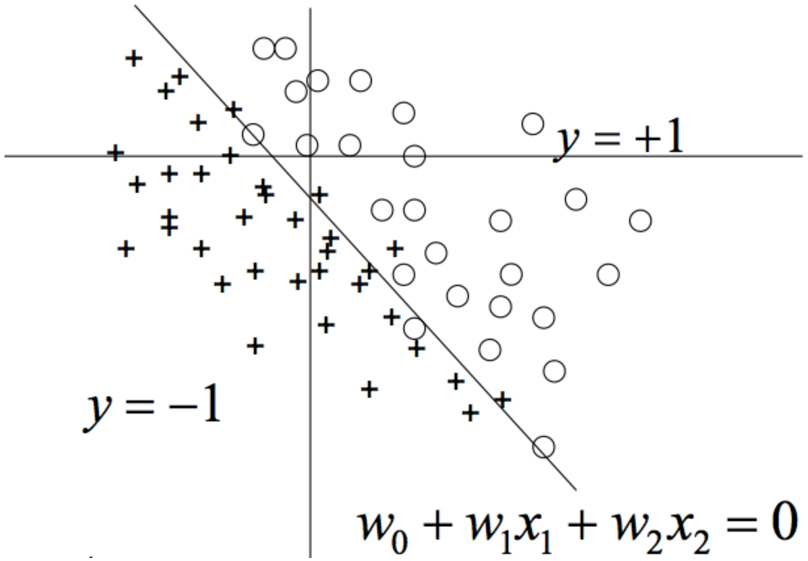
Artificial neural networks

- Examples of ANN:
  - Perceptron
  - Multilayer Perceptron (MLP)
  - Radial Basis Function (RBF)
  - Self-Organizing Map (SOM, or Kohonen map)
- Topologies:
  - Feed forward
  - Recurrent

Perceptron

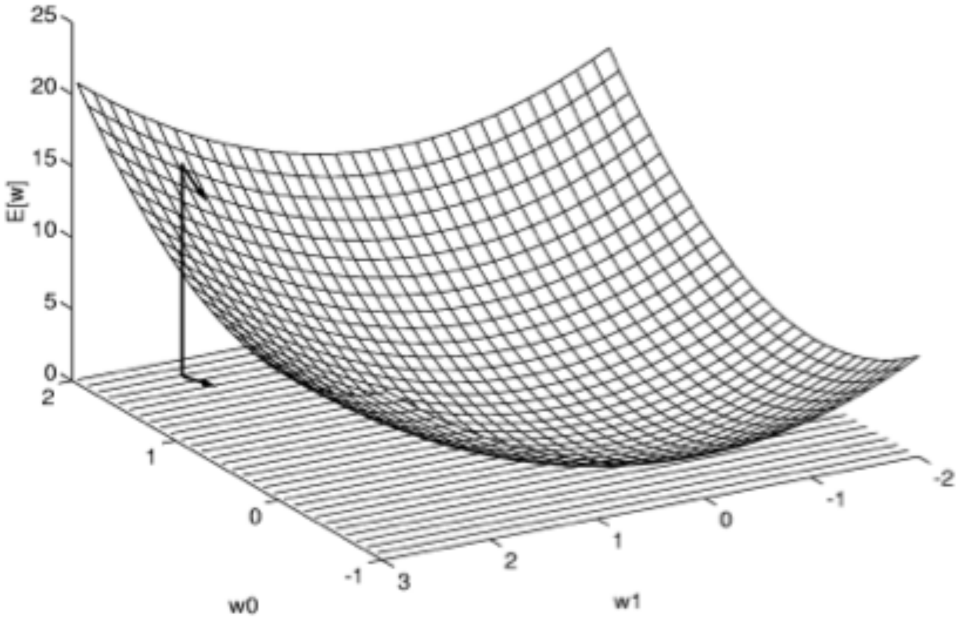
- Defines a (hyper)plane that linearly separates the feature space
- The inputs are real values and the output +1,-1
- Activation functions: step, linear, logistic sigmoid, Gaussian





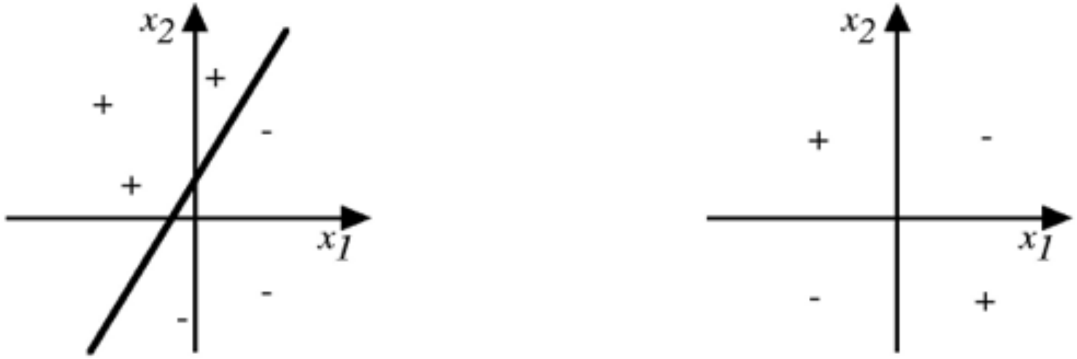
Training a Perceptron

- Considering the simpler linear unit, where the output *o* is given by $o = w_0 + w_1 x_1 + \cdots + w_n x_n$
- The weights can be learnt by minimizing the squared error $$E[\vec{w}] \equiv \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$
- Where D is the set of training examples



Perceptron

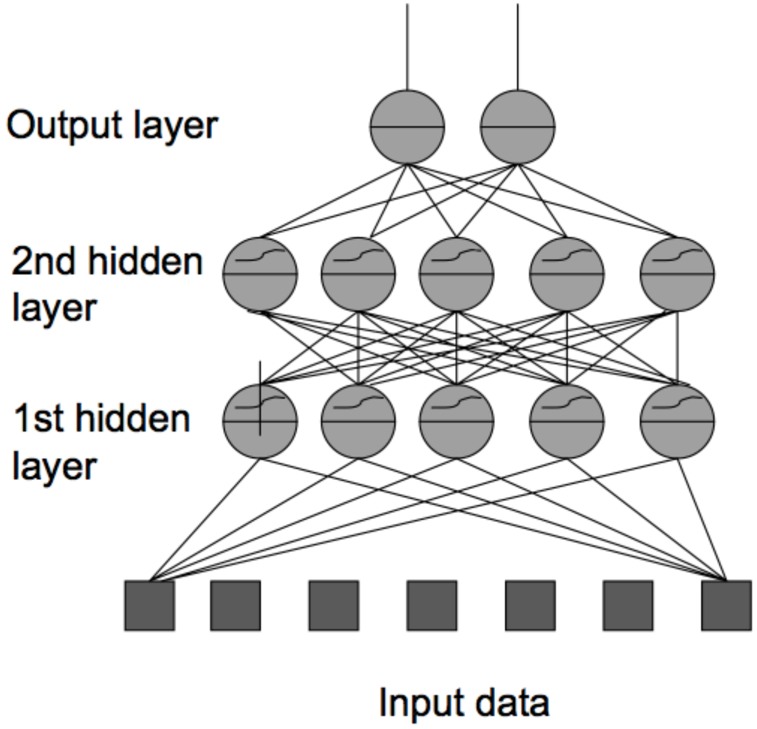
- Decision boundary



- Some functions not representable
  - All not linearly separable
  - Therefore we need a network of perceptrons

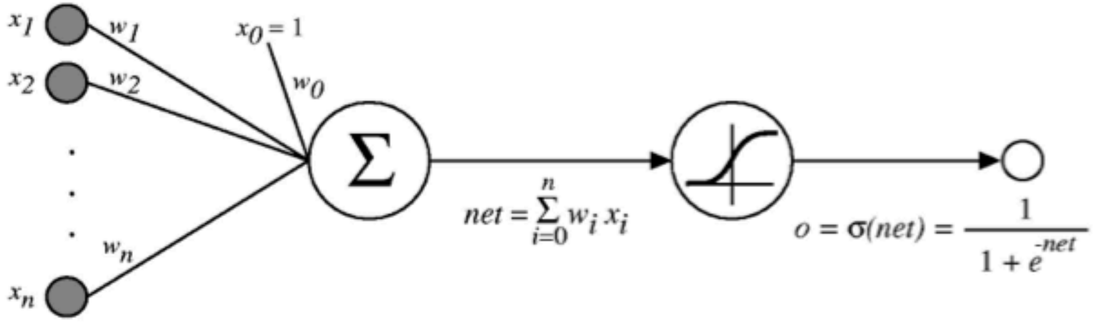
Multilayer perceptron

- To handle more **complex problems** (than linearly separable ones) we need multiple layers.
- Each layer receives its inputs from the previous layer and **forwards** its outputs to the next layer
- The result is the **combination of linear boundaries** which allow the separation of complex data
- Weights are obtained through the **back propagation algorithm**



Multilayer perceptron

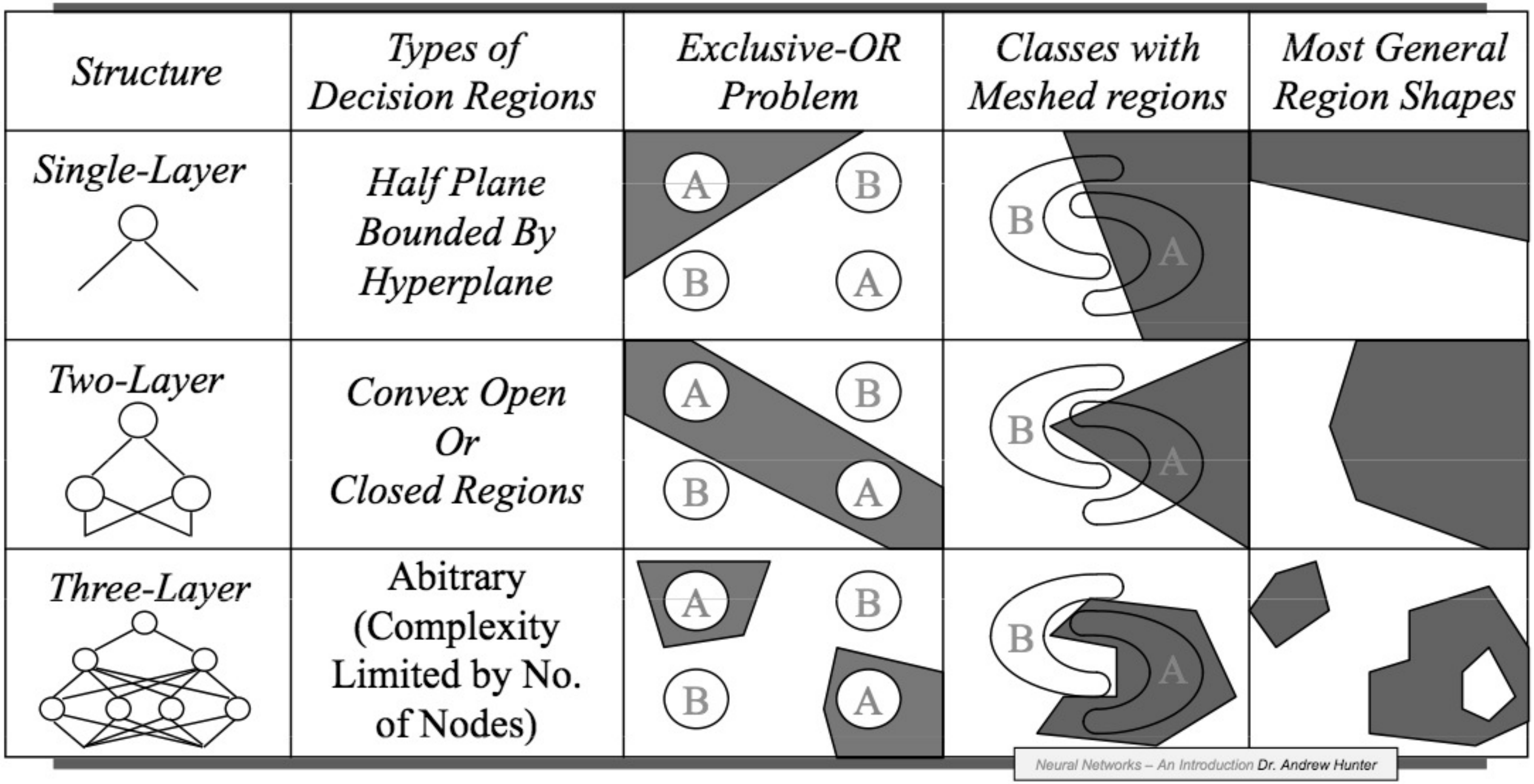
- It is possible to derive the gradient descent rules to train
  - One sigmoid unit
  - Multilayer networks of sigmoid units, using backpropagation


x_1
x_2
x_n
w_1
w_2
w_n
$x_0 = 1$
w_0
Σ
$net = \sum_{i=0}^{n} w_i x_i$
$o = \sigma(net) = \frac{1}{1 + e^{-net}}$

$\sigma(x)$ is the sigmoid function

$$\frac{1}{1+e^{-x}}$$

Non-linearly separable problems



Structure	Types of Decision Regions	Exclusive-OR Problem	Classes with Meshed regions	Most General Region Shapes
Single-Layer	Half Plane Bounded By Hyperplane			
Two-Layer	Convex Open Or Closed Regions			
Three-Layer	Abitrary (Complexity Limited by No. of Nodes)			

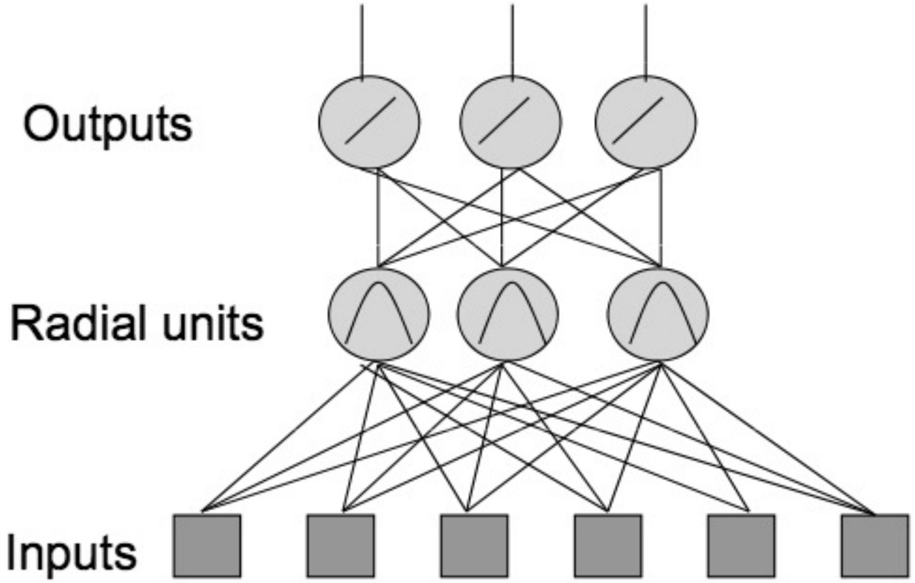
Neural Networks – An Introduction Dr. Andrew Hunter

ANN as a classifier

- Advantages
  - High tolerance to noisy data
  - Ability to classify untrained patterns
  - Well-suited for continuous-valued inputs and outputs
  - Successful on a wide array of real-world data
  - Algorithms are inherently parallel
- Disadvantages
  - Long training time
  - Requires a number of parameters typically best determined empirically, e.g., the network topology or ``structure.”
  - Poor interpretability: Difficult to interpret the symbolic meaning behind the learned weights and of ``hidden units" in the network

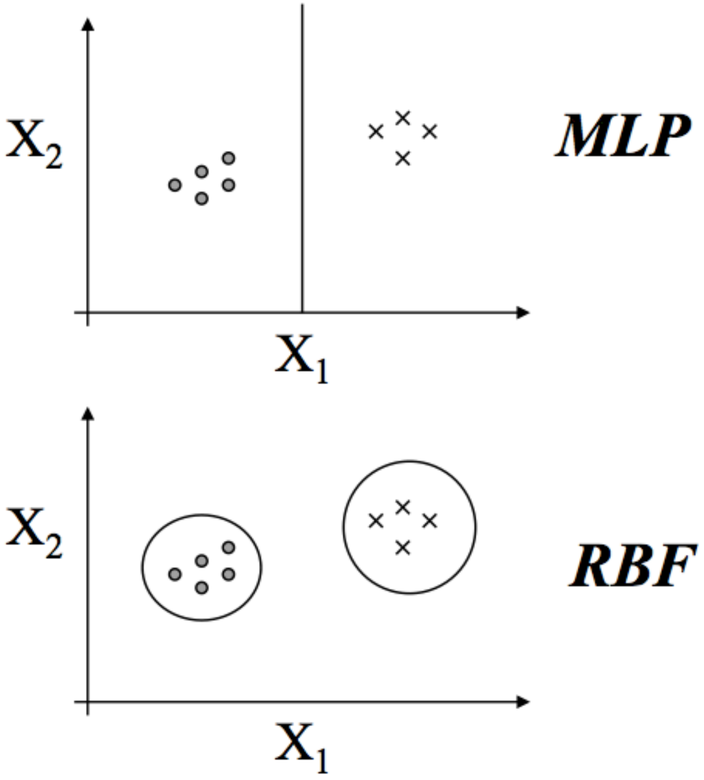
RBF networks

- RBF networks approximate functions using (radial) basis functions as the building blocks. Generally, the hidden unit function is Gaussian and the output Layer is linear



MLP vs RBF

- **Classification**
  - MLPs separate classes via hyperplanes
  - RBFs separate classes via hyperspheres
- **Learning**
  - MLPs use distributed learning
  - RBFs use localized learning
  - RBFs train faster
- **Structure**
  - MLPs have one or more hidden layers
  - RBFs have only one layer
  - RBFs require more hidden neurons => curse of dimensionality

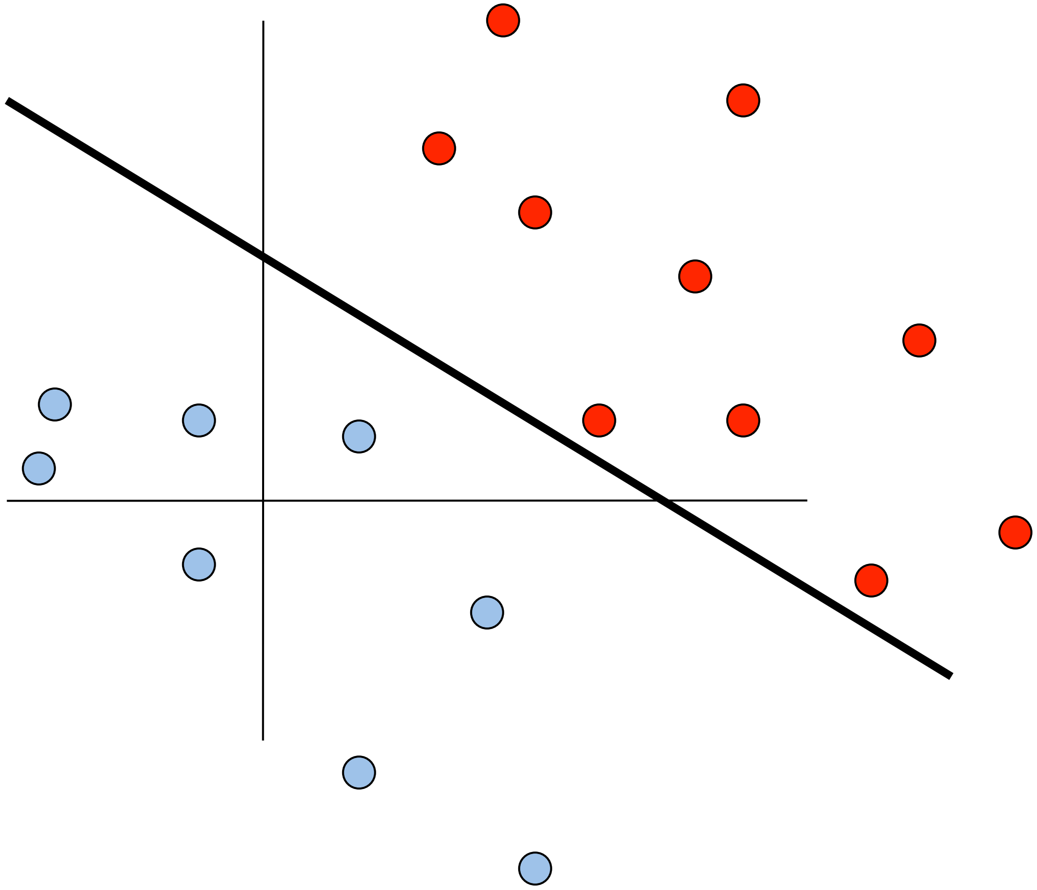


SUPPORT VECTOR MACHINES

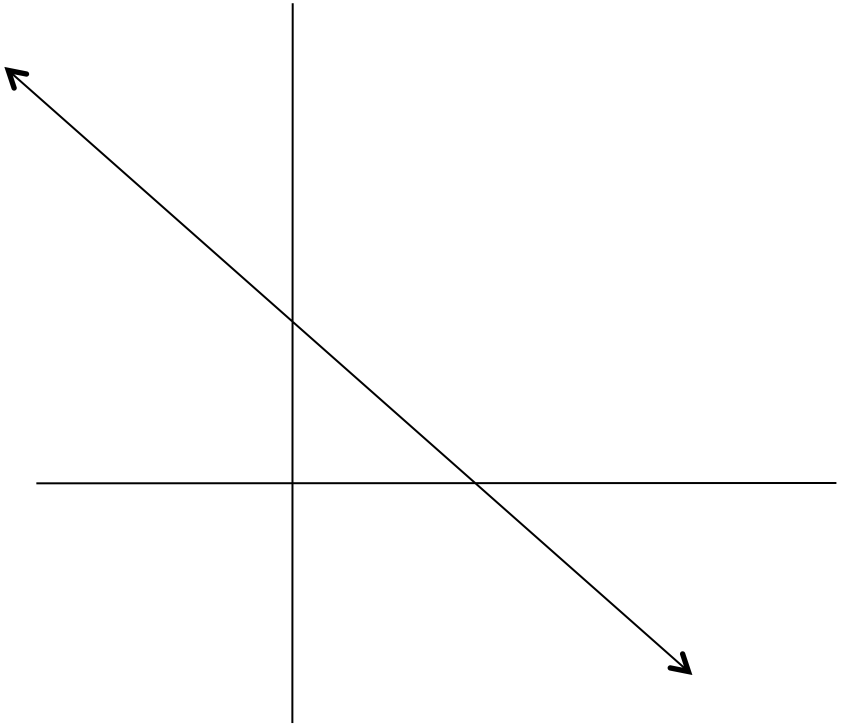
Support Vector Machine

- Discriminant function is a hyperplane (line in 2D) in feature space (similar to the Perceptron)
- In a nutshell:
  - Map the data to a predetermined very high-dimensional space via a kernel function
  - Find the hyperplane that **maximizes the margin** between the two classes
  - If data are not separable find the hyperplane that maximizes the margin and minimizes the (a weighted average of the) misclassifications

Linear classifiers



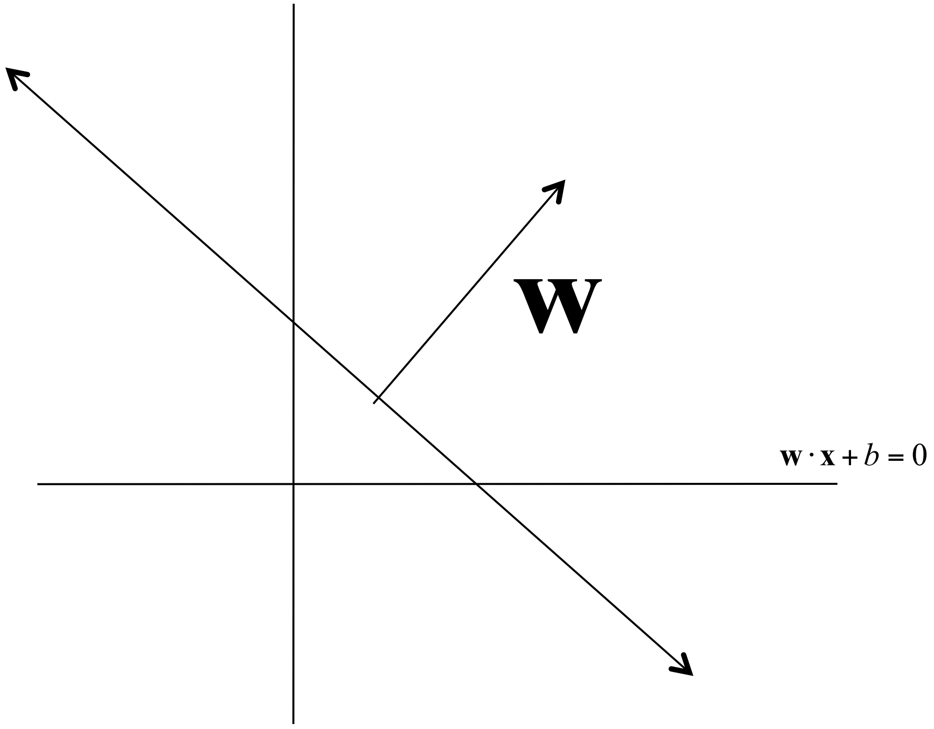
Linear functions in R^2



Let $\mathbf{w} = \begin{bmatrix} a \\ c \end{bmatrix}$ $\mathbf{x} = \begin{bmatrix} x \\ y \end{bmatrix}$

$$ax + cy + b = 0$$

Linear functions in R^2



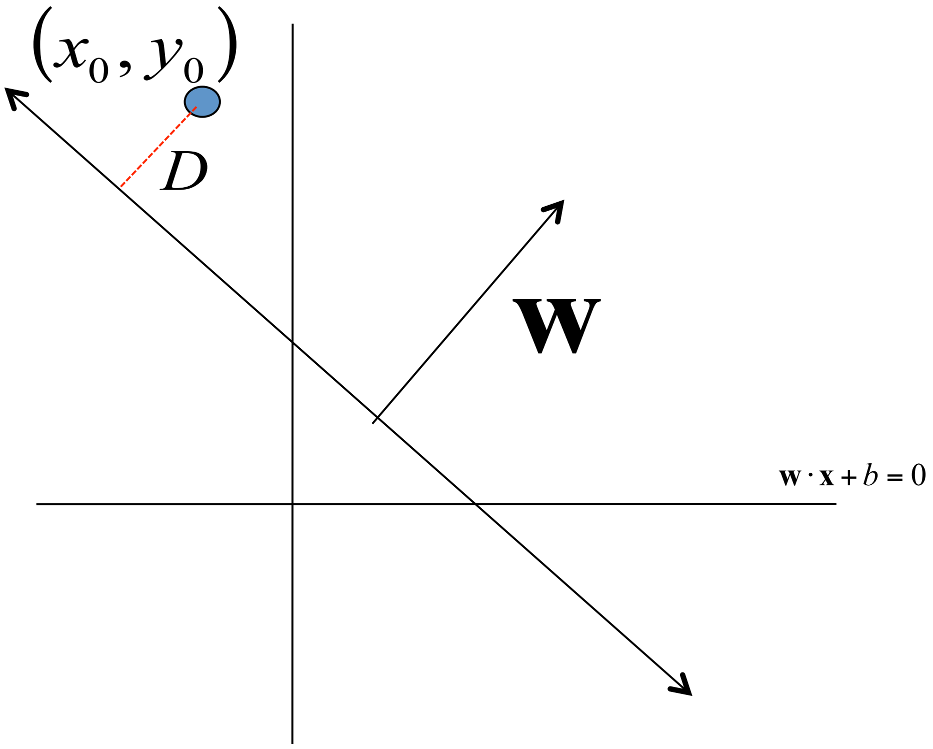
Let $\mathbf{w} = \begin{bmatrix} a \\ c \end{bmatrix}$ $\mathbf{x} = \begin{bmatrix} x \\ y \end{bmatrix}$

$$ax + cy + b = 0$$

$$\Updownarrow$$

$$\mathbf{w}^T \mathbf{x} + b = 0$$

Linear functions in R^2



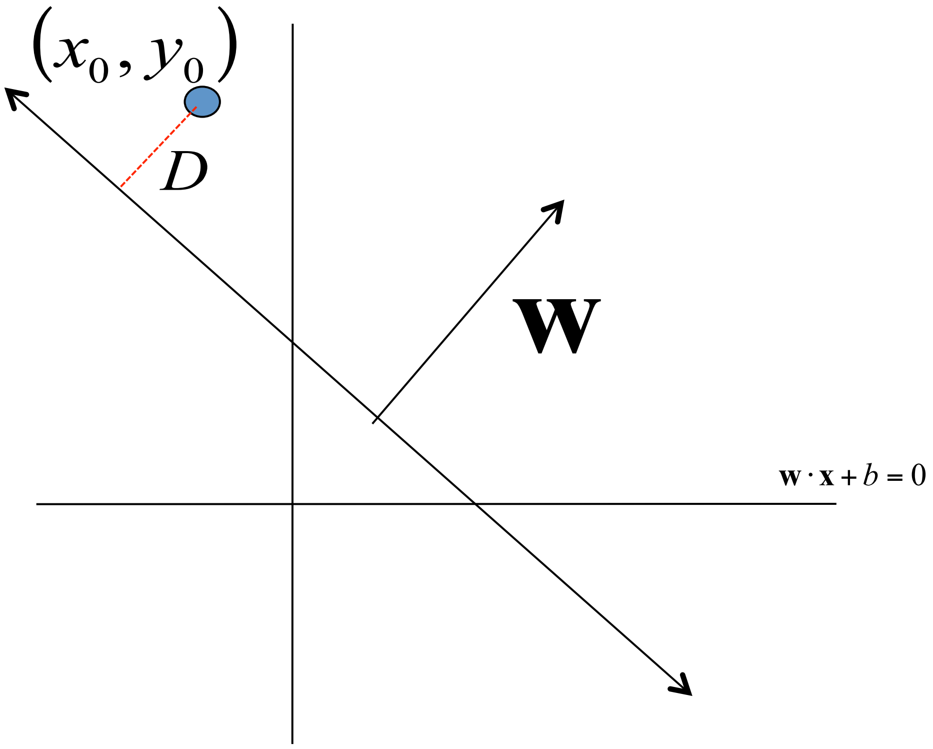
Let $\mathbf{w} = \begin{bmatrix} a \\ c \end{bmatrix}$ $\mathbf{x} = \begin{bmatrix} x \\ y \end{bmatrix}$

$$ax + cy + b = 0$$

$$\Updownarrow$$

$$\mathbf{w}^T \mathbf{x} + b = 0$$

Linear functions in R^2



Let $\mathbf{w} = \begin{bmatrix} a \\ c \end{bmatrix}$ $\mathbf{x} = \begin{bmatrix} x \\ y \end{bmatrix}$

$$ax + cy + b = 0$$

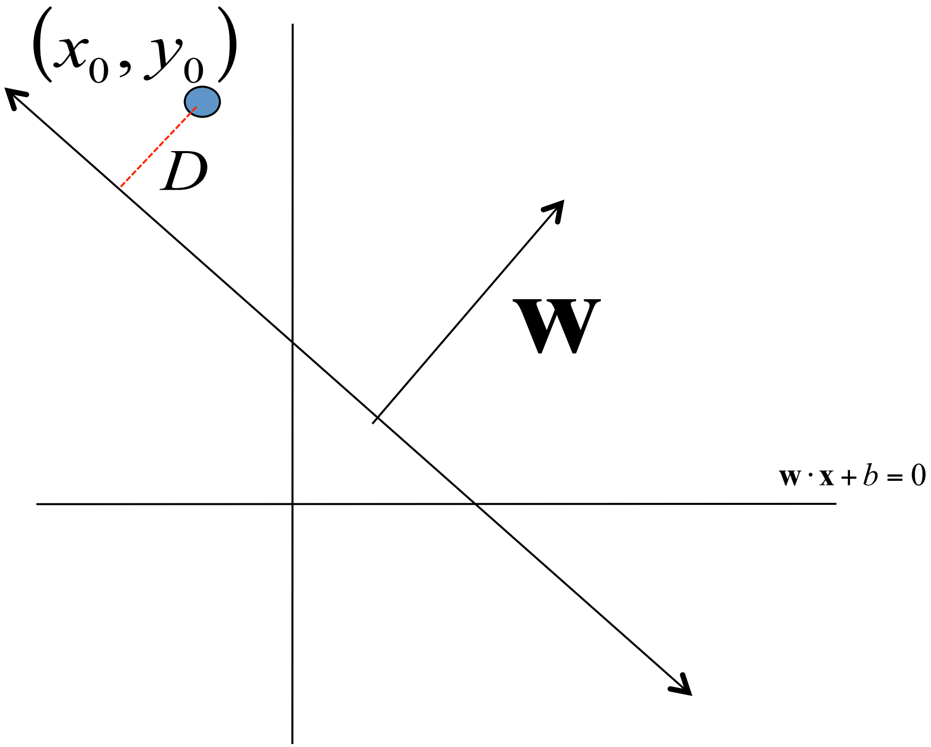
$$\Updownarrow$$

$$\mathbf{w}^T \mathbf{x} + b = 0$$

$$D = \frac{|ax_0 + cy_0 + b|}{\sqrt{a^2 + c^2}}$$

distance from point to line

Linear functions in R^2



Let $\mathbf{w} = \begin{bmatrix} a \\ c \end{bmatrix}$ $\quad \mathbf{x} = \begin{bmatrix} x \\ y \end{bmatrix}$

$$ax + cy + b = 0$$

$$\Updownarrow$$

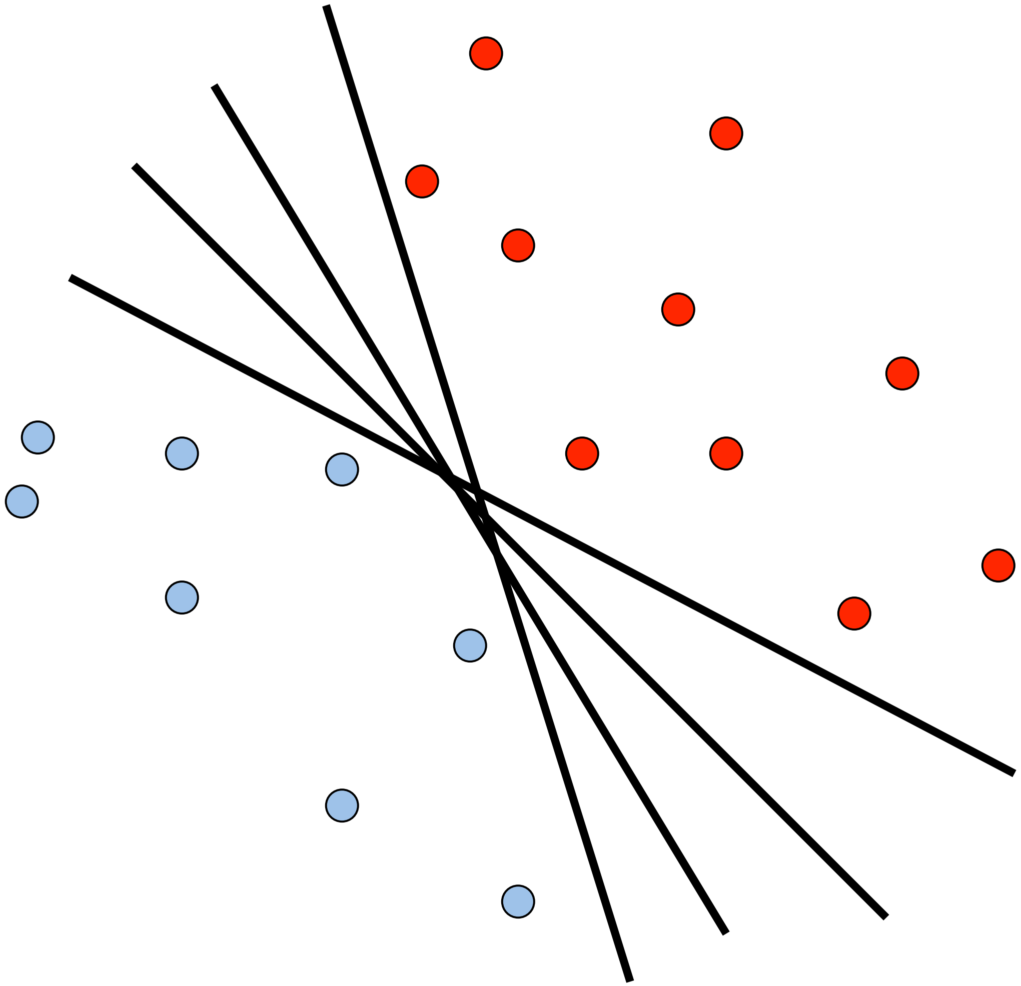
$$\mathbf{w}^T \mathbf{x} + b = 0$$

$$D = \frac{|ax_0 + cy_0 + b|}{\sqrt{a^2 + c^2}} = \frac{\mathbf{w}^{\mathrm{T}}\mathbf{x} + b}{\|\mathbf{w}\|}$$

distance from point to line

Linear classifiers

Find linear function to separate positive and negative examples

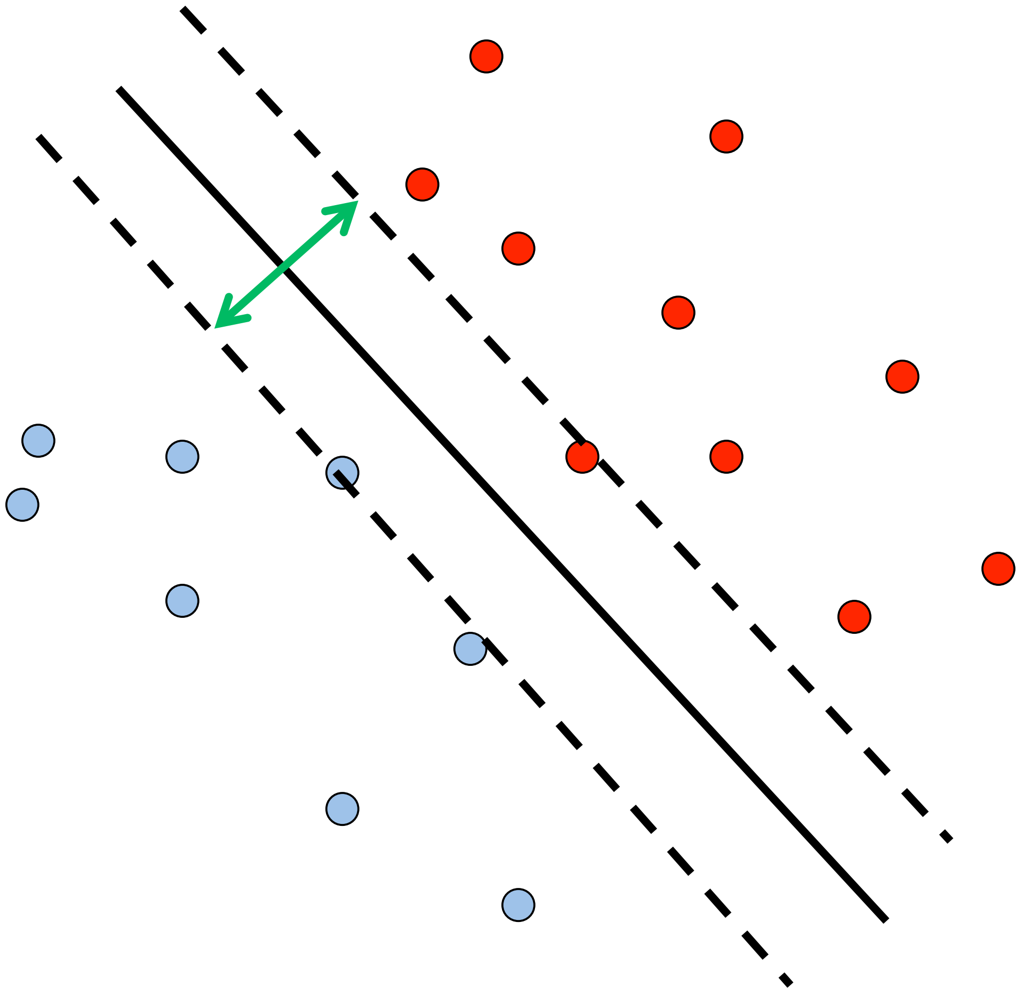


$$\mathbf{x}_i \text{ positive}: \quad \mathbf{x}_i \cdot \mathbf{w} + b \geq 0$$

$$\mathbf{x}_i \text{ negative}: \quad \mathbf{x}_i \cdot \mathbf{w} + b < 0$$

Which line
is best?

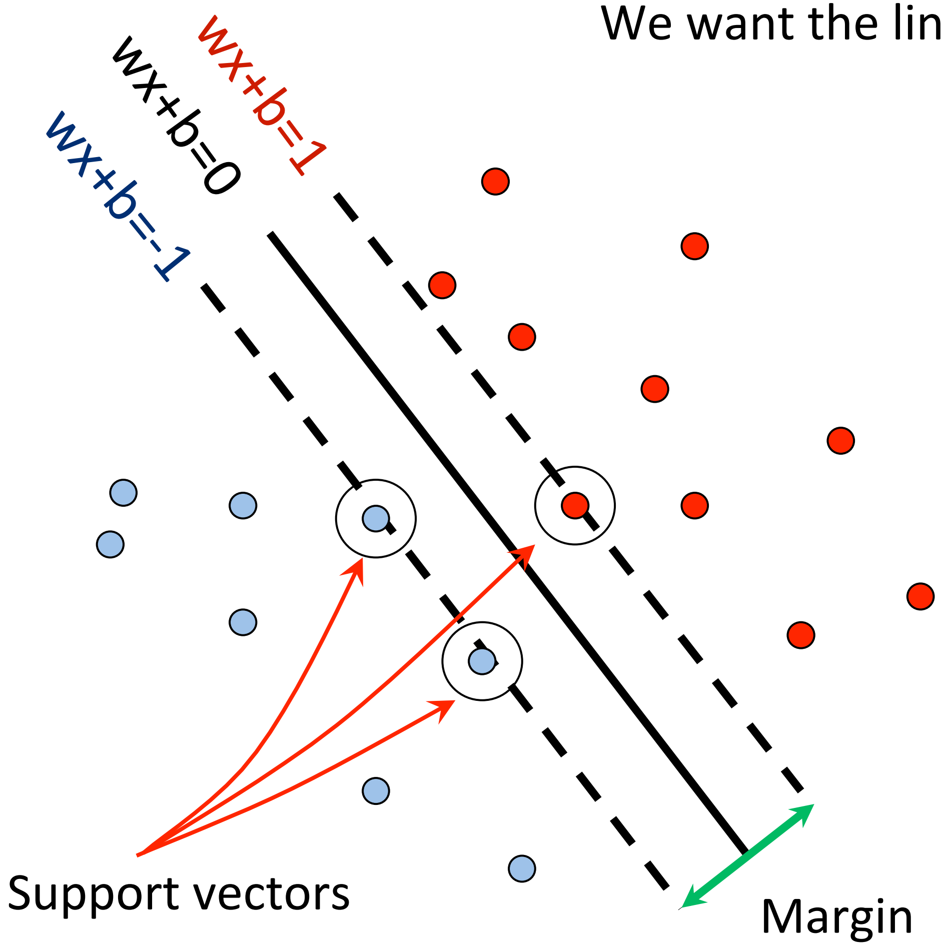
Support Vector Machines



Classifier based on *optimal separating line (for 2D case)*

Maximize the ***margin*** between the positive and negative training examples

Support Vector Machines



We want the line that maximizes the margin.

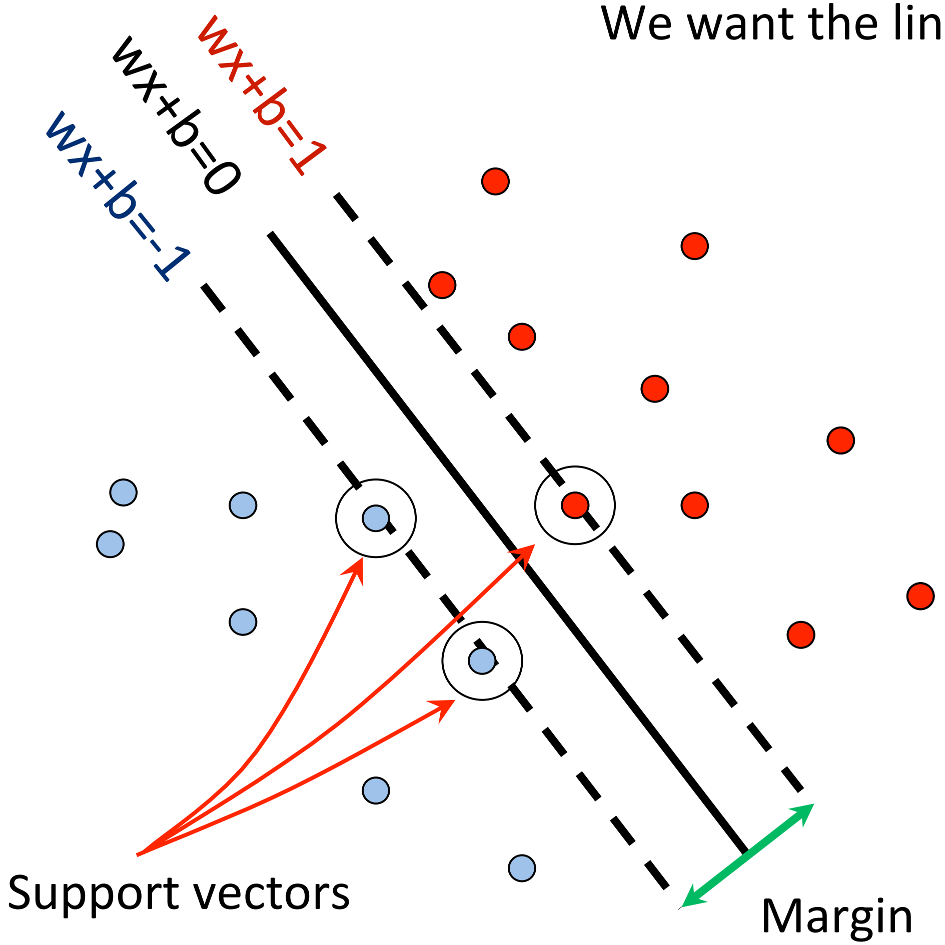
$$\mathbf{x}_i \text{ positive } (y_i = 1): \qquad \mathbf{x}_i \cdot \mathbf{w} + b \geq 1$$

$$\mathbf{x}_i \text{ negative } (y_i = -1): \qquad \mathbf{x}_i \cdot \mathbf{w} + b \leq -1$$

For support, vectors, $\mathbf{x}_i \cdot \mathbf{w} + b = \pm 1$

Support Vector Machines

We want the line that maximizes the margin.



$\mathbf{x}_i$ positive $(y_i = 1)$: $\quad \mathbf{x}_i \cdot \mathbf{w} + b \geq 1$

$\mathbf{x}_i$ negative $(y_i = -1)$: $\quad \mathbf{x}_i \cdot \mathbf{w} + b \leq -1$

For support, vectors, $\quad \mathbf{x}_i \cdot \mathbf{w} + b = \pm 1$

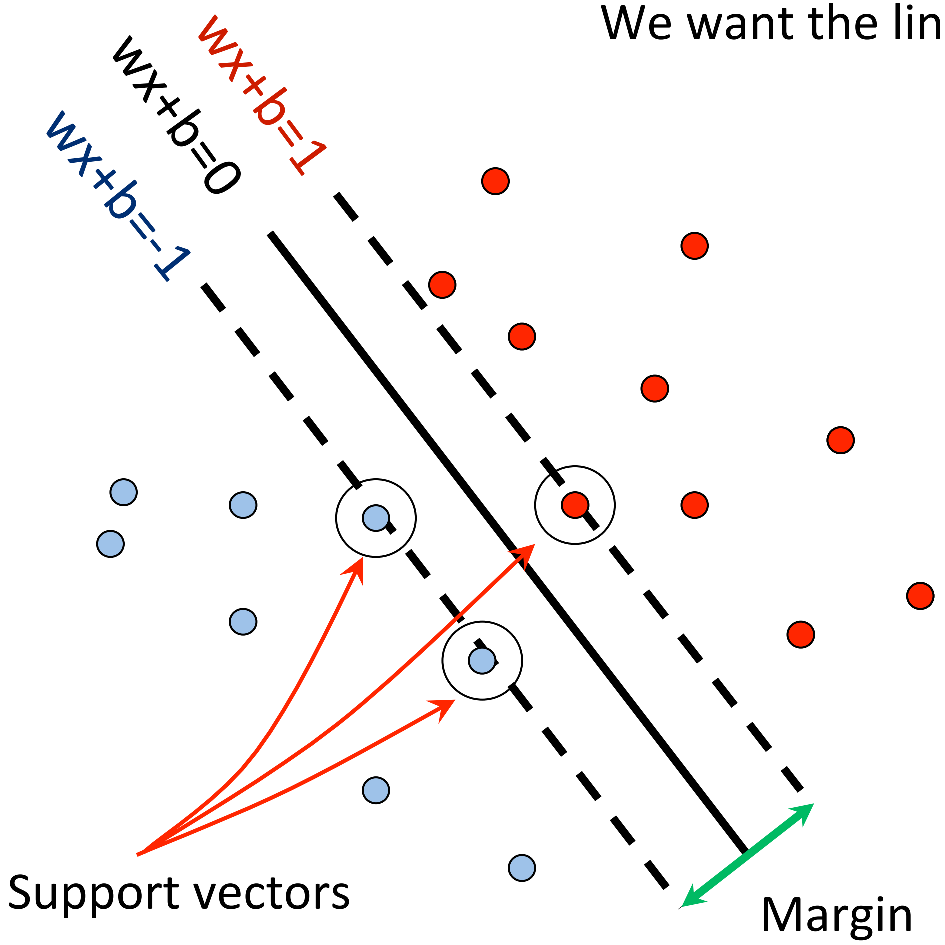
Distance between point and line: $\quad \dfrac{|\,\mathbf{x}_i \cdot \mathbf{w} + b\,|}{\|\,\mathbf{w}\,\|}$

For support vectors:

$$\frac{\mathbf{w}^T\mathbf{x}+b}{\|\mathbf{w}\|} = \frac{\pm 1}{\|\mathbf{w}\|} \qquad M = \left|\frac{1}{\|\mathbf{w}\|} - \frac{-1}{\|\mathbf{w}\|}\right| = \frac{2}{\|\mathbf{w}\|}$$

Support Vector Machines

We want the line that maximizes the margin.



$\mathbf{x}_i$ positive $(y_i = 1)$: $\quad \mathbf{x}_i \cdot \mathbf{w} + b \geq 1$

$\mathbf{x}_i$ negative $(y_i = -1)$: $\quad \mathbf{x}_i \cdot \mathbf{w} + b \leq -1$

For support, vectors, $\quad \mathbf{x}_i \cdot \mathbf{w} + b = \pm 1$

Distance between point and line: $$\frac{|\,\mathbf{x}_i \cdot \mathbf{w} + b\,|}{\|\,\mathbf{w}\,\|}$$

Therefore, the margin is 2 / ||w||

Finding the maximum margin line

1. Maximize margin $2/\|\mathbf{w}\|$
2. Correctly classify all training data points:

$$\mathbf{x}_i \text{ positive } (y_i = 1): \qquad \mathbf{x}_i \cdot \mathbf{w} + b \geq 1$$

$$\mathbf{x}_i \text{ negative} (y_i = -1): \quad \mathbf{x}_i \cdot \mathbf{w} + b \leq -1$$

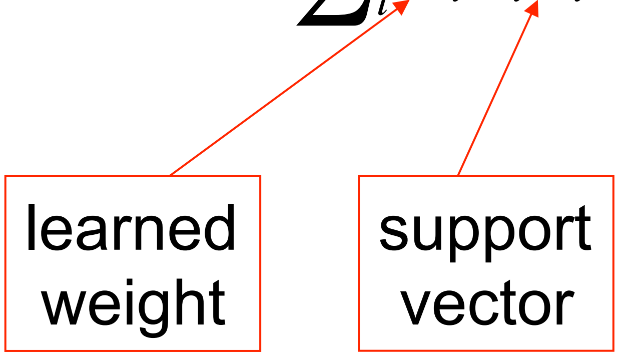
Quadratic optimization problem:

Minimize $\frac{1}{2}\mathbf{w}^T\mathbf{w}$

Subject to $y_i(\mathbf{w}\cdot\boldsymbol{x}_i+b) \geq 1$

Finding the maximum margin line

- Solution: $\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i$



learned weight

support vector

Finding the maximum margin line

- Solution: $\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i$

  $b = y_i - \mathbf{w}\cdot\mathbf{x}_i$ (for any support vector)

  $$\mathbf{w}\cdot\mathbf{x} + b = \sum_i \alpha_i y_i \mathbf{x}_i \cdot \mathbf{x} + b$$

- Classification function:

$$f(x) = \mathrm{sign}\,(\mathbf{w}\cdot\mathbf{x} + \mathrm{b})$$
$$= \mathrm{sign}\left(\sum_{\mathrm{i}} \alpha_{\mathrm{i}} \boxed{\mathbf{x}_i \cdot \mathbf{x}} + b\right)$$

If f(x) < 0, classify as negative,
if f(x) > 0, classify as positive

Questions

- **What if the features are not 2D?**
- What if the data is not linearly separable?
- What if we have more than just two categories?

Questions

- What if the features are not 2D?
  - Generalizes to d-dimensions – replace line with “hyperplane”
- What if the data is not linearly separable?
- What if we have more than just two categories?

Questions

- What if the features are not 2D?
- **What if the data is not linearly separable?**
- What if we have more than just two categories?

Soft-margin SVMs

- Introduce **slack variable** and allow some instances to fall within the margin, but penalize them
- Constraint becomes: $y_i(w\cdot x_i + b) \geq 1 - \xi_i,\ \forall x_i$

$$\xi_i \geq 0$$

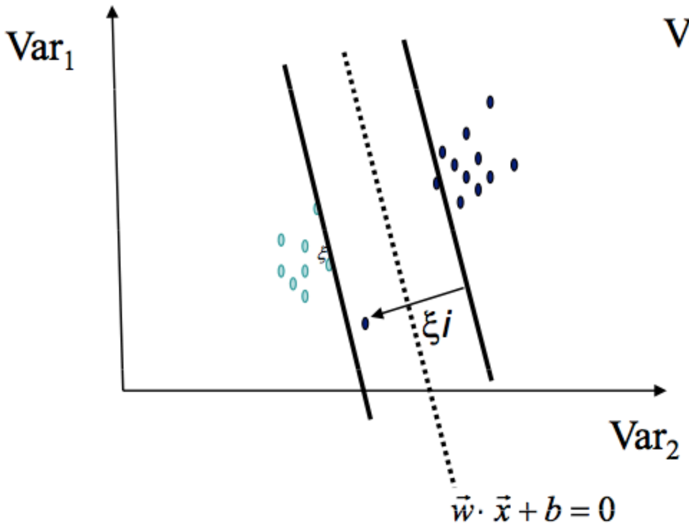
- Objective function penalizes for misclassified instances within the margin

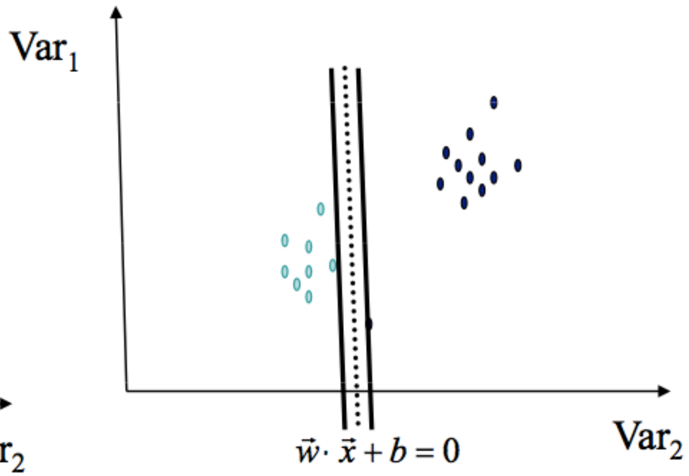
$$\min \frac{1}{2}\|w\|^2 + C\sum_i \xi_i$$

- C trades-off margin width and classifications
- As *C*→∞, we get closer to the hard-margin solution

Soft-margin vs Hard-margin SVMs

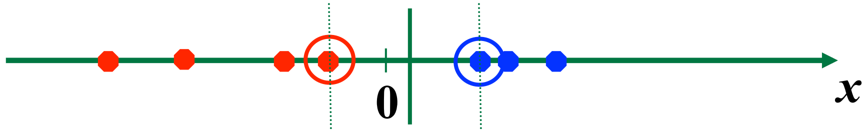
- Soft-Margin always has a solution
- Soft-Margin is more robust to outliers
  - Smoother surfaces (in the non-linear case)
- Hard-Margin does not require to guess the cost parameter (requires no parameters at all)



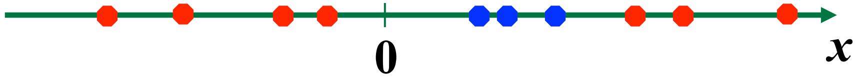


Non-linear SVMs

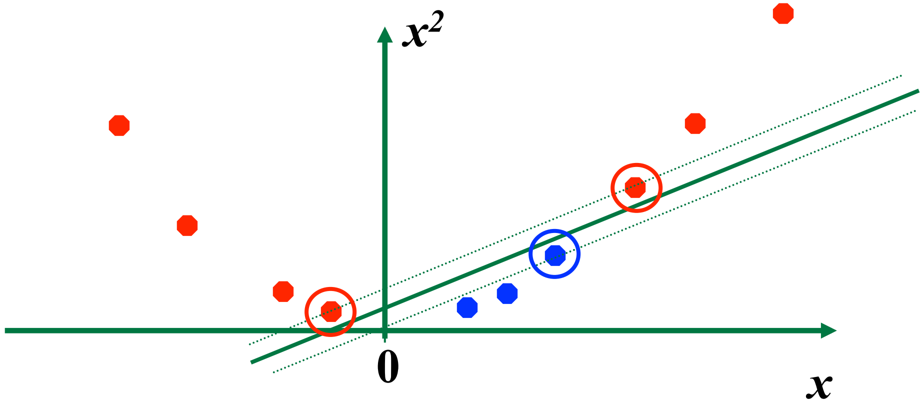
- Datasets that are linearly separable with some noise work out great:



- But what are we going to do if the dataset is just too hard?

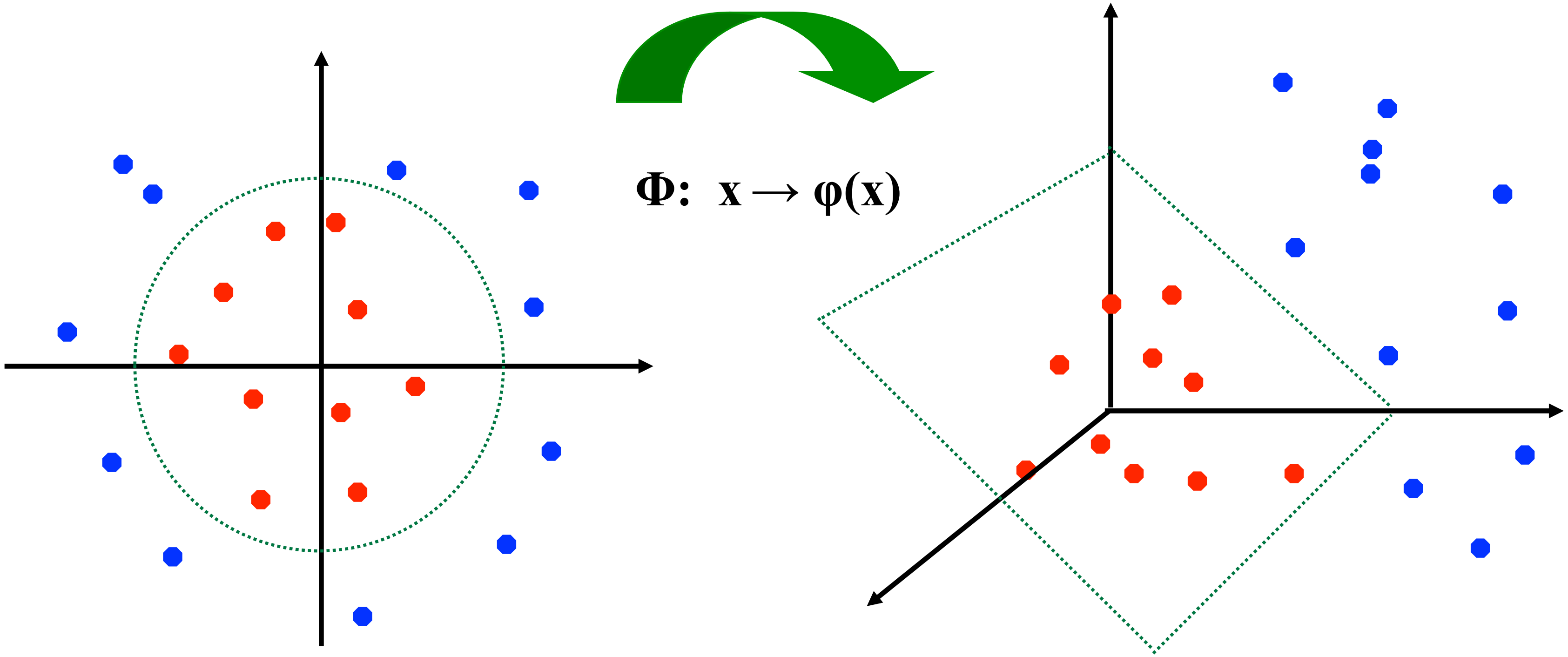


- How about… mapping data to a higher-dimensional space:



Non-linear SVMs

- General idea: the original input space can be **mapped** to some higher-dimensional feature space where the training set is separable:



The “Kernel Trick”

- The linear classifier relies on dot product between vectors $K(x_i,x_j)=x_i^T x_j$

- If every data point is mapped into high-dimensional space via some transformation Φ: $x \rightarrow \varphi(x)$, the dot product becomes:

$$K(x_i,x_j)= \varphi(x_i)^T\varphi(x_j)$$

- A *kernel function* is a similarity function that corresponds to an inner product in some expanded feature space.

Non-linear SVMs

- *The kernel trick*: instead of explicitly computing the lifting transformation $\boldsymbol{\varphi}(\mathbf{x})$, define a kernel function K such that

$$K(\mathbf{x}_i, \mathbf{x}_j) = \boldsymbol{\varphi}(\mathbf{x}_i) \cdot \boldsymbol{\varphi}(\mathbf{x}_j)$$

- This gives a nonlinear decision boundary in the original feature space:

$$\sum_i \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b$$

Examples of kernel functions

- Linear:

$$K(x_i, x_j) = x_i^T x_j$$

- Gaussian RBF:

$$K(x_i, x_j) = \exp\left(-\frac{\left\| x_i - x_j \right\|^2}{2\sigma^2}\right)$$

- Histogram intersection:

$$K(x_i, x_j) = \sum_k \min(x_i(k), x_j(k))$$

Questions

- What if the features are not 2D?
- What if the data is not linearly separable?
- **What if we have more than just two categories?**

Multi-class SVMs

- Achieve multi-class classifier by combining a number of binary classifiers

- **<u>One vs. all</u>**
  - Training: learn an SVM for each class vs. the rest
  - Testing: apply each SVM to test example and assign to it the class of the SVM that returns the highest decision value
- **<u>One vs. one</u>**
  - Training: learn an SVM for each pair of classes
  - Testing: each learned SVM "votes" for a class to assign to the test example

SVM issues

- Choice of kernel
  - Gaussian or polynomial kernel is default
  - if ineffective, more elaborate kernels are needed
  - domain experts can give assistance in formulating appropriate similarity measures
- Choice of kernel parameters
  - e.g. σ in Gaussian kernel, is the distance between closest points with different classifications
  - In the absence of reliable criteria, rely on the use of a validation set or cross-validation to set such parameters
- Optimization criterion – Hard margin v.s. Soft margin
  - series of experiments in which parameters are tested

SVM as a classifier

- Advantages
  - Many SVM packages available
  - Kernel-based framework is very powerful, flexible
  - Often a sparse set of support vectors – compact at test time
  - Works very well in practice, even with very small training sample sizes
- Disadvantages
  - No “direct” multi-class SVM, must combine two-class SVMs
  - Can be tricky to select best kernel function for a problem
  - Computation, memory
    - During training time, must compute matrix of kernel values for every pair of examples
    - Learning can take a very long time for large-scale problems

ENSEMBLE LEARNING

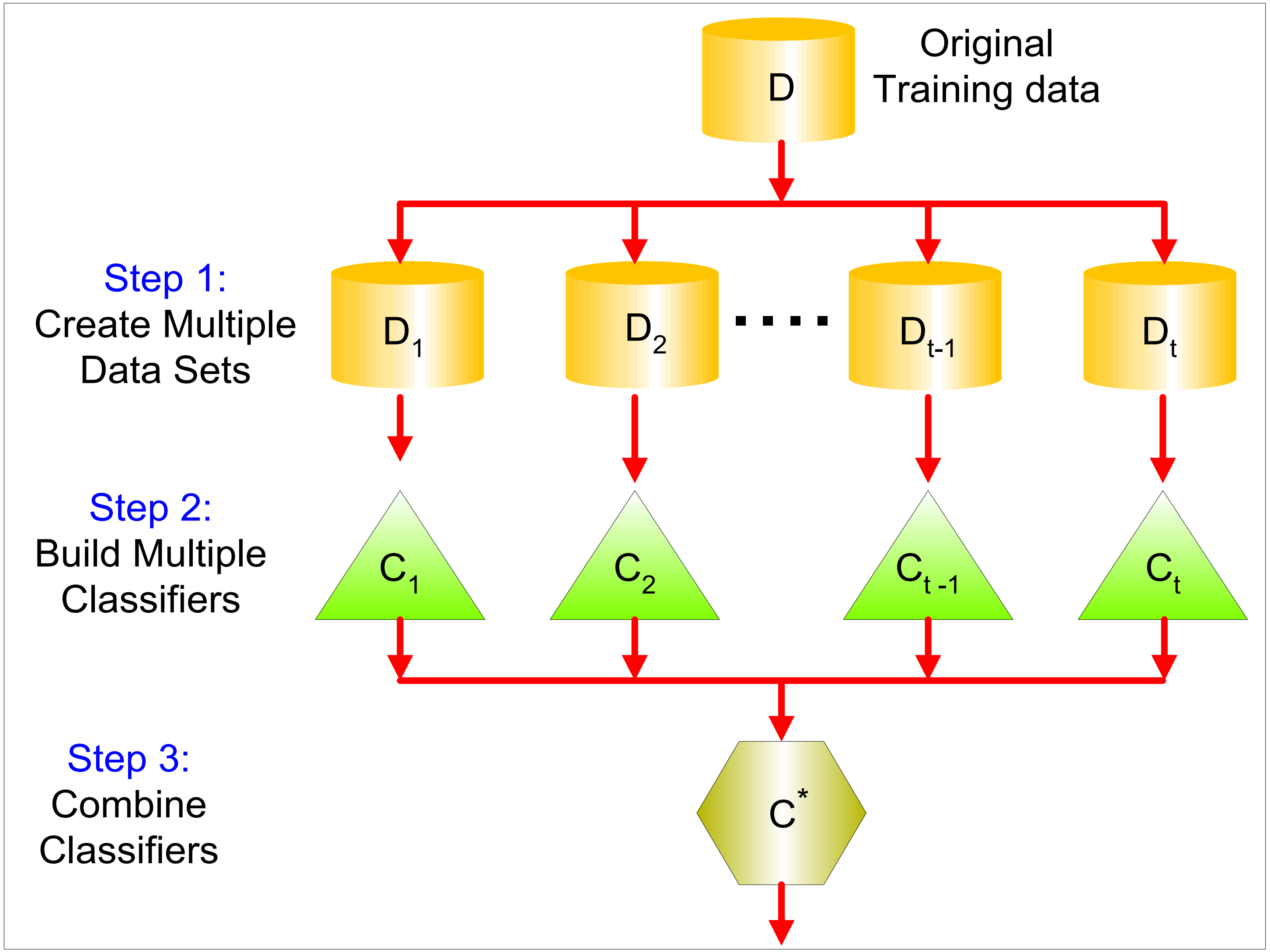
Ensemble learning

- What is ensemble learning?
  - Ensemble learning refers to a collection of methods that learn a target function by **training a number of individual learners** and **combining their predictions**
- Why ensemble learning?
  - **Accuracy**: a more reliable mapping can be obtained by combining the output of multiple "experts"
  - **Efficiency**: a complex problem can be decomposed into multiple sub-problems that are easier to understand and solve (divide-and-conquer approach)
  - There is not a single model that works for all PR problems

Ensemble learning

- When to use ensemble learning?
  - When it is possible to build component classifiers that are more accurate than chance and, more importantly, that are independent from each other
- Why does it work?
  - Because uncorrelated errors of individual classifiers can be eliminated through averaging
- Ensemble methods work better with 'unstable classifiers' – why?
- Classifiers that are sensitive to minor perturbations in the training set. Examples:
  - Decision trees
  - Rule-based
  - Artificial neural networks

Ensemble classifiers



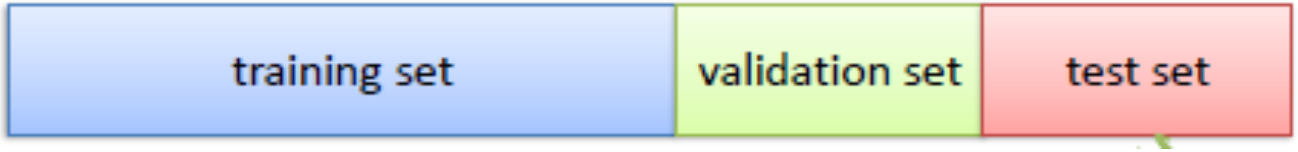
Ensemble learning

- Bagging
  - Also known as bootstrap aggregation
  - Sampling **uniformly with replacement**
  - Build classifier on each “bootstrap” sample
- Boosting
  - **focuses** more on previously **misclassified records**
  - E.g.: Adaboost
- Stacking
  - apply multiple base learners (e.g. decision trees, naïve Bayes, neural networks)
- Random Forests
  - specifically designed for **decision tree classifiers**

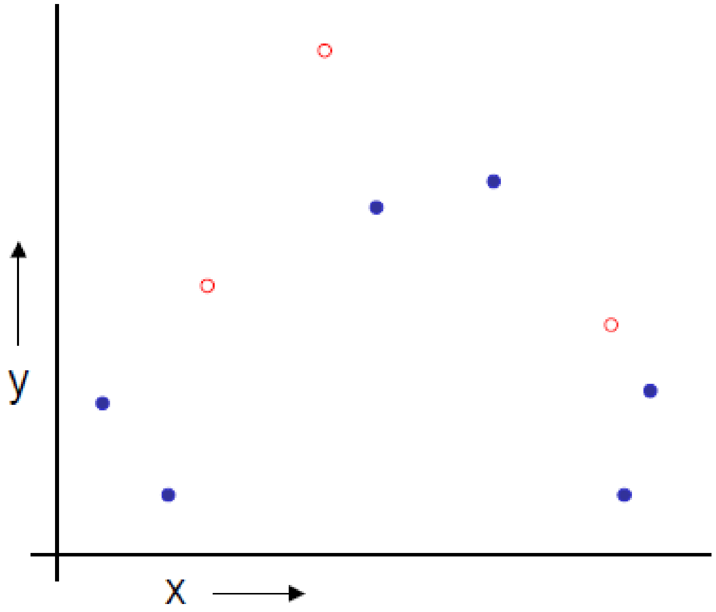
CROSS VALIDATION

Training - general strategy

- We try to simulate the real world scenario.
- Test data is our future data.
- Validation set can be our test set - we use it to select our model.
- The whole aim is to estimate the models’ true error on the sample data we have.

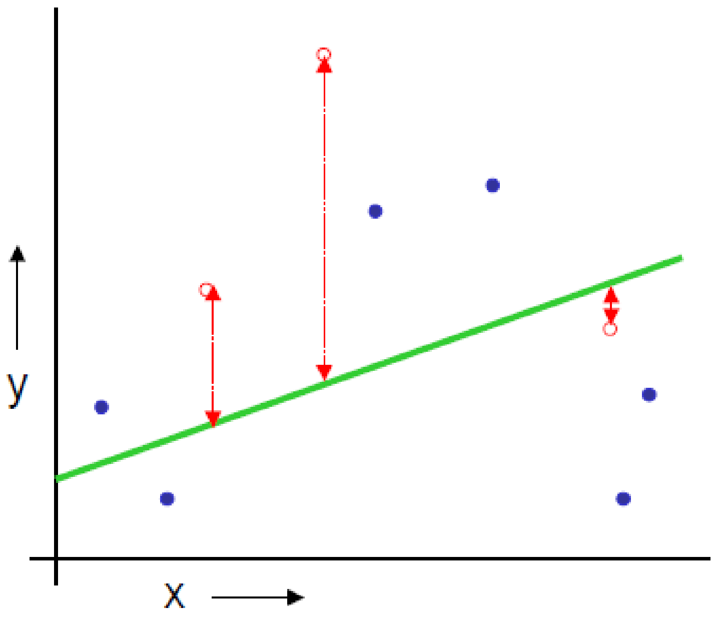


Validation set method



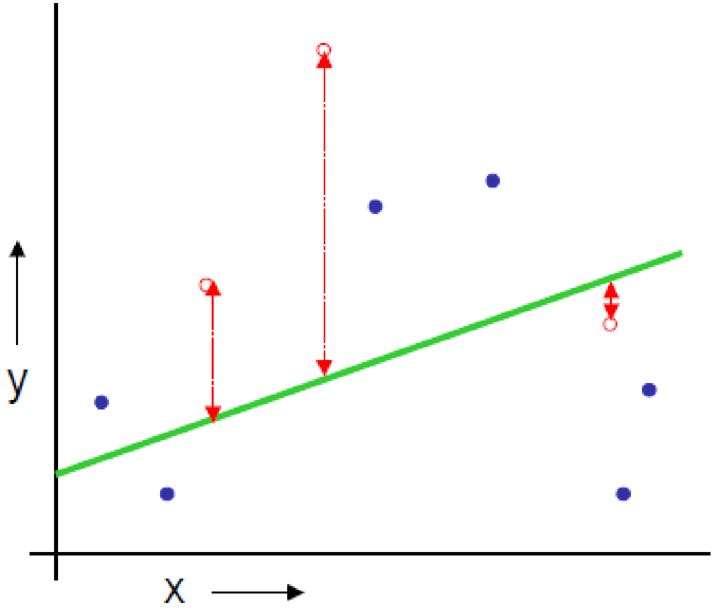
- Randomly split some portion of your data. Leave it aside as the **validation set**
- The remaining data is the **training data**

Validation set method



- Randomly split some portion of your data. Leave it aside as the **validation set**
- The remaining data is the **training data**
- Learn a **model** from the training set

Validation set method



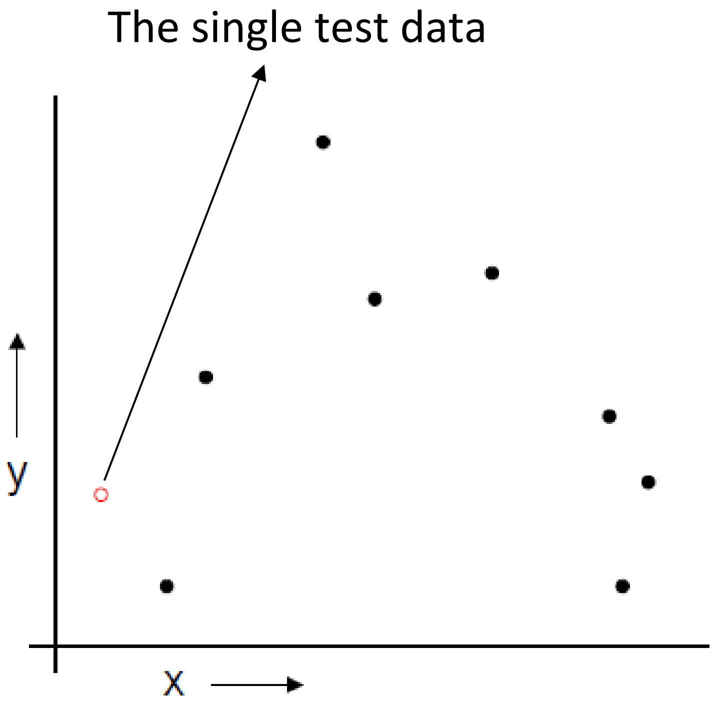
- Randomly split some portion of your data. Leave it aside as the **validation set**
- The remaining data is the **training data**
- Learn a **model** from the training set
- Estimate your future **performance** with the test data

Test set method

- It is simple, however
  - We waste some portion of the data
  - If we do not have much data, we may be lucky or unlucky with our test data

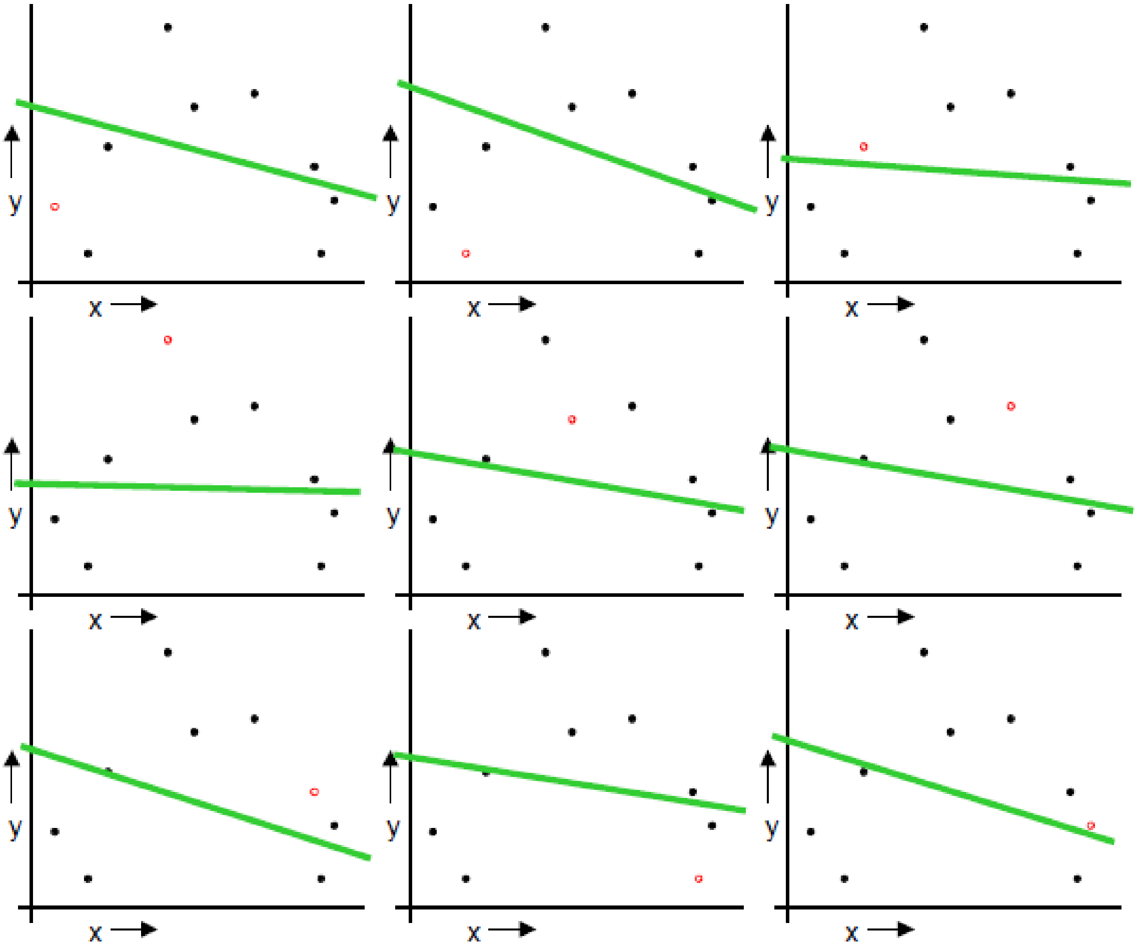
- With **cross-validation** we reuse the data

LOOCV (Leave-one-out Cross Validation)



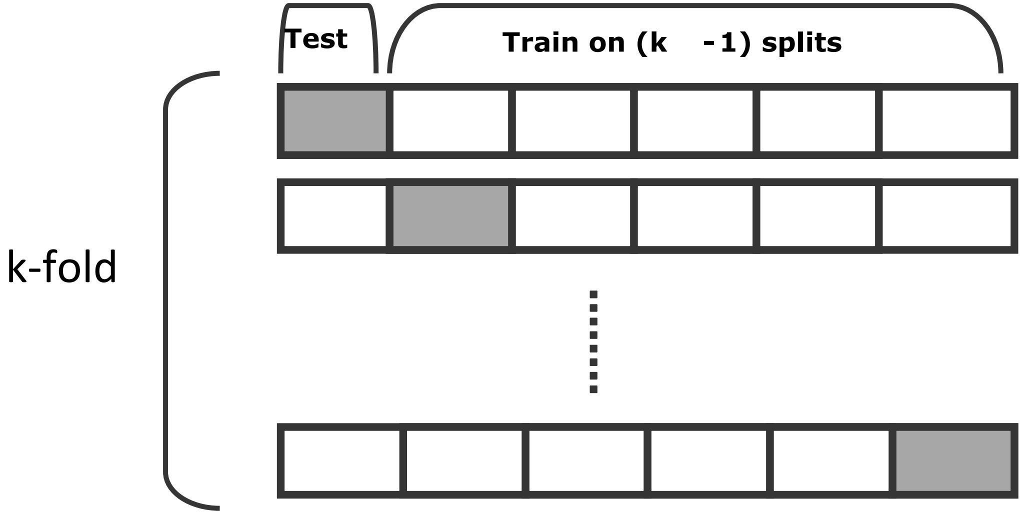
- Let us say we have N data points and k as the index for data points, k=1..N
- Let *(x_k,y_k)* be the *k*th record
- Temporarily remove (x_k,y_k) from the dataset
- Train on the remaining N-1 datapoints
- Test the error on (x_k,y_k)
- Do this for each k=1..N and report the mean error.

LOOCV (Leave-one-out Cross Validation)



- Repeat the validation N times, for each of the N data points.
- The validation data is changing each time.

K-fold cross validation



In 3 fold cross validation, there are 3 runs.
In 5 fold cross validation, there are 5 runs.
In 10 fold cross validation, there are 10 runs.

the error is averaged over all runs

References

- Christopher M. Bishop, Pattern Recognition and Machine Learning, Springer, 2006.
- Richard O. Duda, Peter E. Hart, David G. Stork, Pattern Classification, John Wiley & Sons, 2001
- Thomas Mitchell, Machine Learning, McGraw-Hill, 1997.
- P. Domingos, “A few useful things to know about machine learning,” CACM, 2012
- Andrew Moore, Support Vector Machines Tutorial, http://www.autonlab.org/tutorials/svm.html

References

- Selim Aksoy, Introduction to Pattern Recognition, Part I, http://retina.cs.bilkent.edu.tr/papers/patrec_tutorial1.pdf
- Ricardo Gutierrez-Osuna, Introduction to Pattern Recognition, http://research.cs.tamu.edu/prism/lectures/pr/pr_l1.pdf
- Pedro Domingos, Machine Learning, http://courses.cs.washington.edu/courses/cse446/14wi/
- Kristen Grauman, Discriminative classifiers for image recognition, http://www.cs.utexas.edu/~grauman/courses/spring2011/slides/lecture22_classifiers.pdf
- Victor Lavrenko and Nigel Goddard, Introductory Applied Machine Learning, http://www.inf.ed.ac.uk/teaching/courses/iaml/