Supervised Classification

Vincent Itier (IMT Nord Europe)
Víctor Elvira (IMT Nord Europe), John Klein (Univ. Lille)

UV DATA 2020

IMT Nord Europe
École Mines-Télécom
IMT-Université de Lille

Outline

Introduction

Evaluation & metrics

Generative Models for Classification

Discriminative Models for Classification

Decision Trees

Large margin classifiers

Ensemble Methods

Outline

Outline :

- What is Supervised Learning?
- How to Evaluate Classification Models?
- Methods

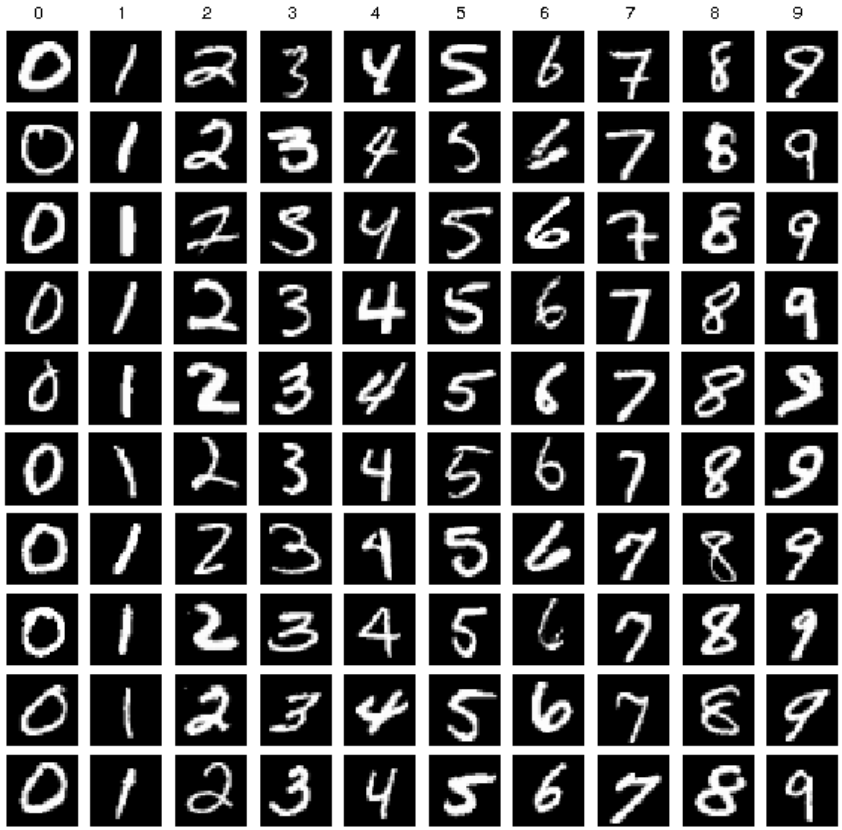
Supervised Learning?

GOAL : Make prediction based on experience.
Experience : DATA + Feedback !

Supervised Learning? Examples

Spam detection

Supervised Learning? Examples



Character recognition

Supervised Learning? Examples

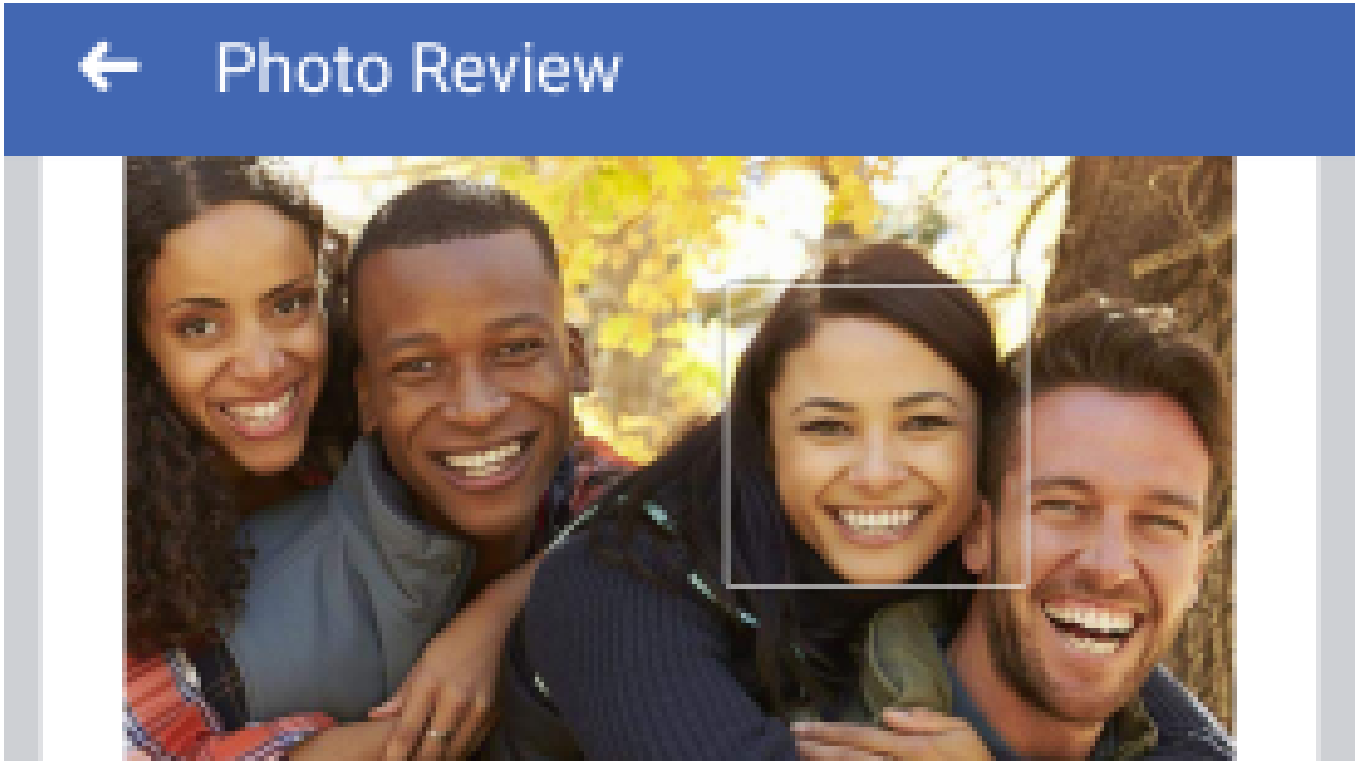
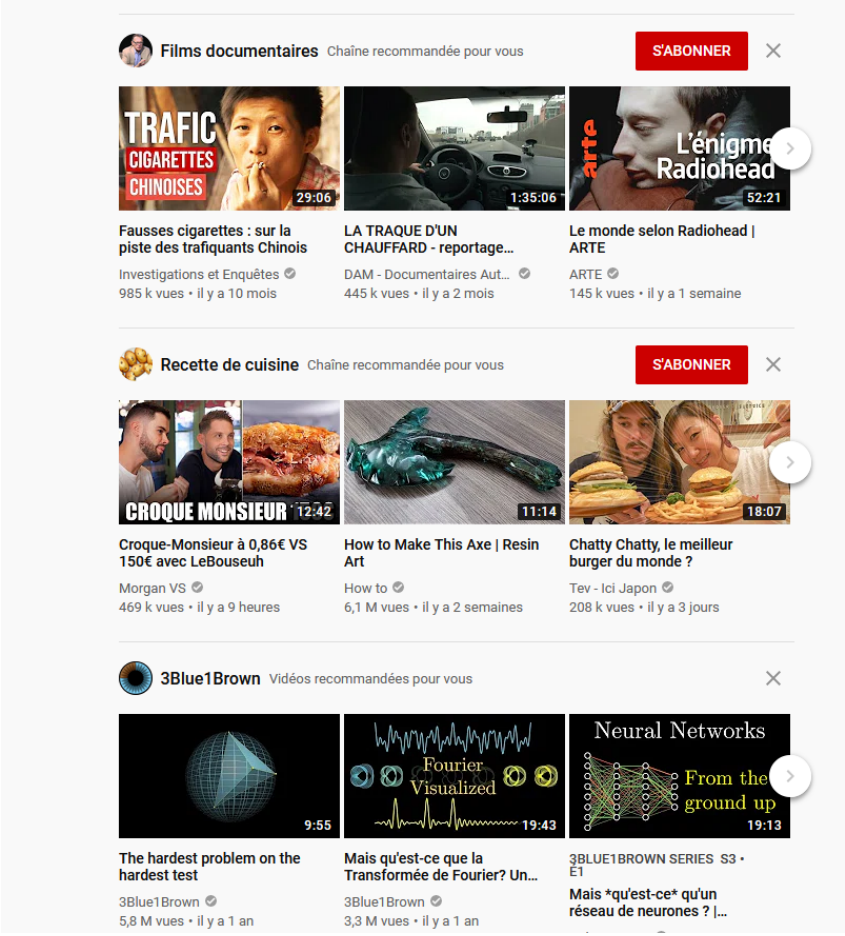


Image classification

Supervised Learning? Examples



Recommandation

Supervised Learning

- Classification
- Regression
- Reinforcement learning

Supervised classification:

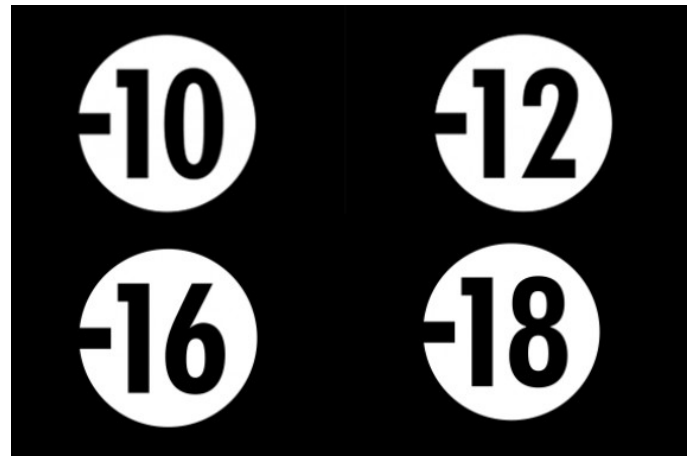
- Classification is a problem that is used to predict which class a data point is part of which is usually a discrete value.
- Classification is used to predict a discrete class or label(Y).
- Classification basically involves assigning new input variables (X) to the class to which they most likely belong in based on a classification model that was built from the training data that was already labeled.
- Labeled data is used to train a classifier so that the algorithm performs well on data that does not have a label(not yet labeled).
- Repeating this process of training a classifier on already labeled data is known as "learning".

Supervised classification:

- Binary classification.
  - Is this person is likely to go see this movie?
- Multi-class classification.
  - What is the rate of a movie? (restriction)
- Multi-label classification.
  - Classify movies based on their genres

Supervised classification:

- Binary classification.
  - Is this person is likely to go see this movie?
- Multi-class classification.
  - What is the rate of a movie? (restriction)
- Multi-label classification.
  - Classify movies based on their genres

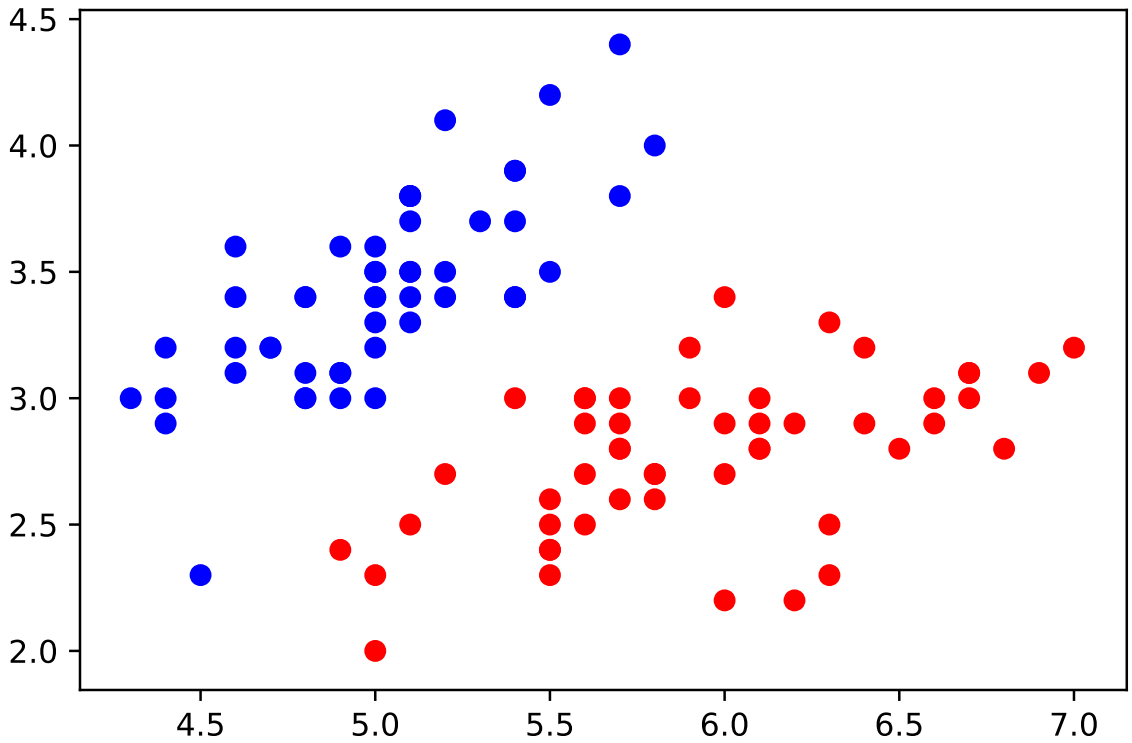


Supervised classification:

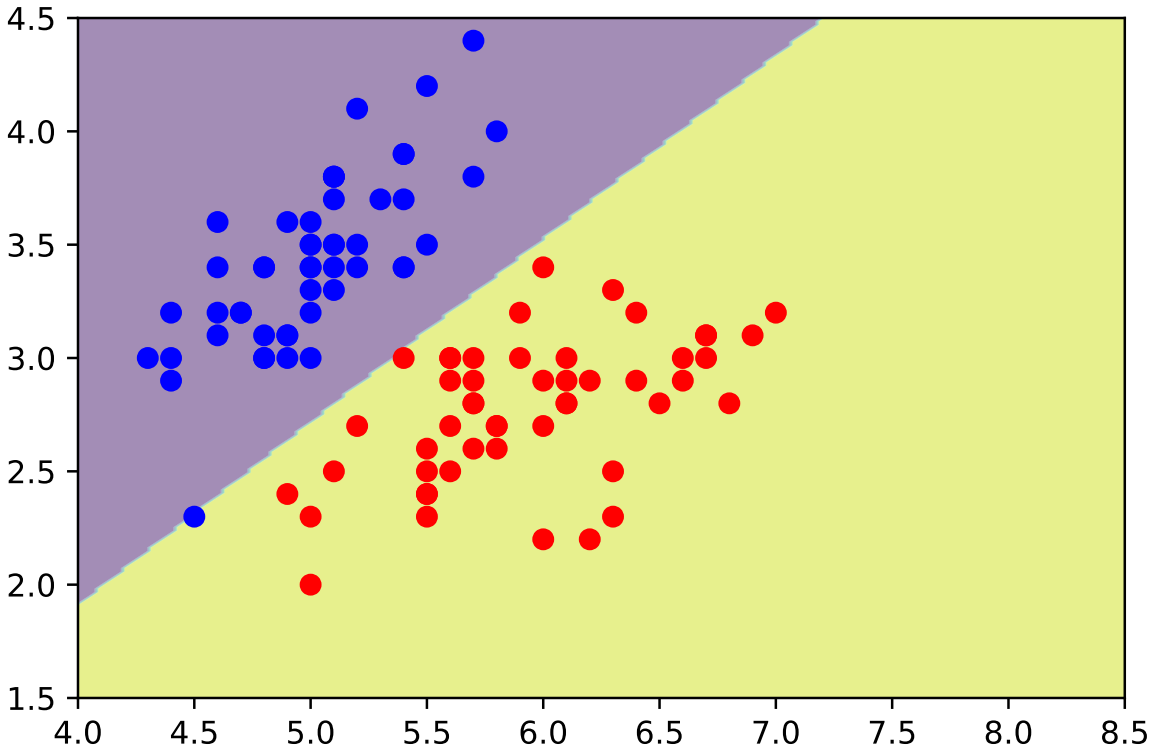
- Binary classification.
  - Is this person is likely to go see this movie?
- Multi-class classification.
  - What is the rate of a movie? (restriction)
- Multi-label classification.
  - Classify movies based on their genres

Horror: 0.02%
Romance: 0.02%
Adventure: 99.96%
Documentary: 0.0%

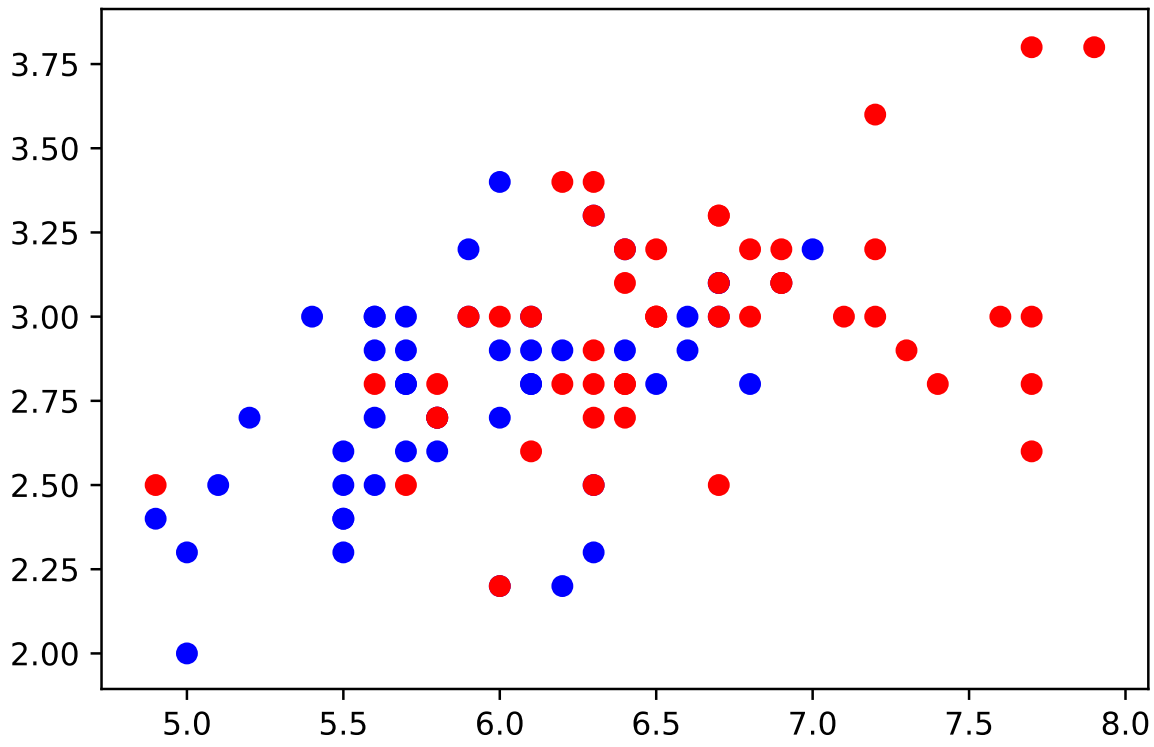
A simple example:



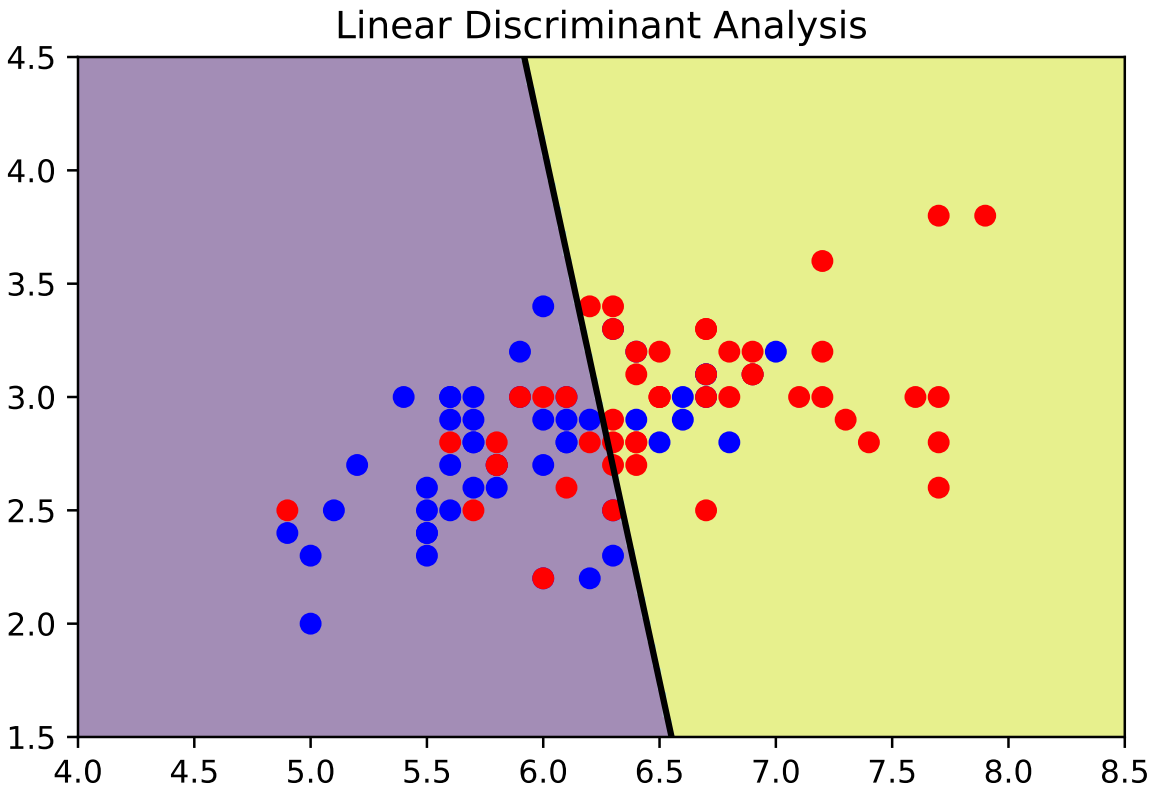
A simple example:



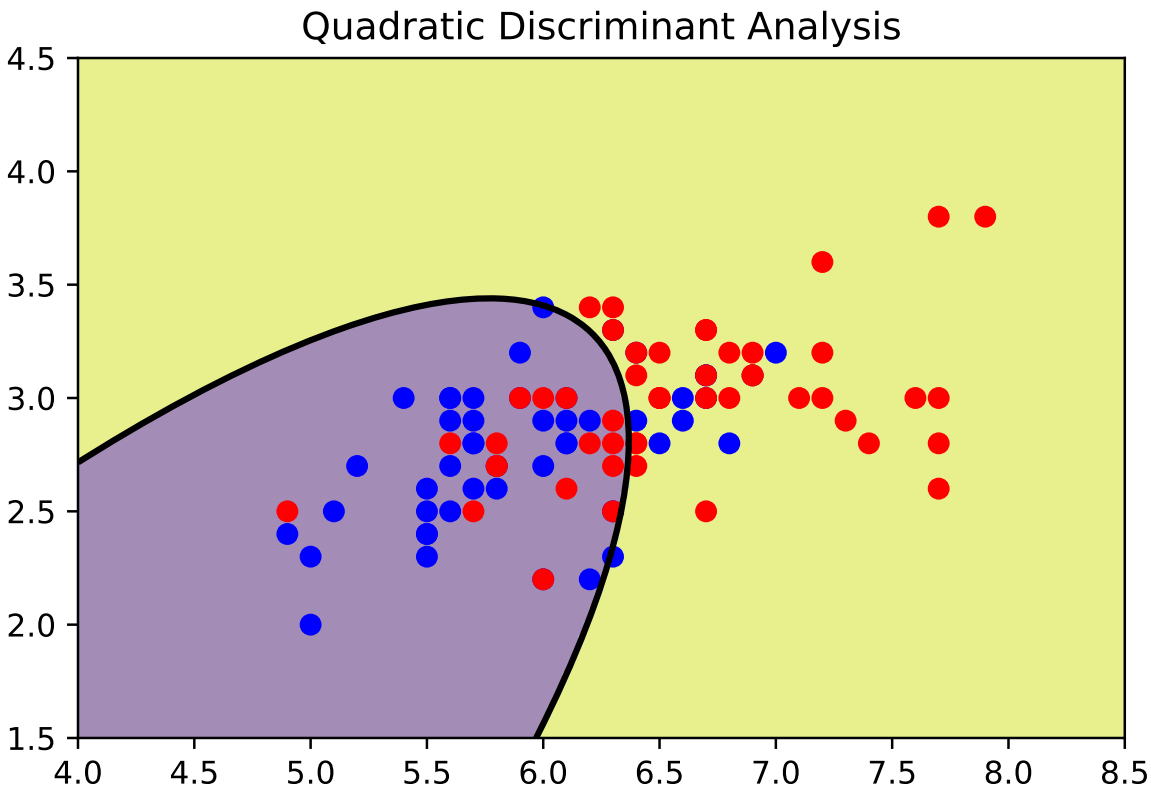
A simple example:



A simple example:



A simple example:



Data:

- Features extraction
- Dimension redcution
- Deep learning

Methods:

- KNN
- Trees
- Naive-Bayes
- SVM
- Neural network
- Ensemble methods

Classification Problems

- Many machine learning applications can be seen as classification problems
  - given a vector of p "inputs" describing an item, predict which "class" the item belongs to.
- Examples:
  - Given anatomical measurements of an animal, predict which species the animal belongs to.
  - Given information on the credit history of a customer, predict whether or not they would pay back a loan.
  - Given an image of a hand-written digit, predict which digit (0-9) it is.
  - Given the proportions of iron, nickel, carbon, etc. in a type of steel, predict whether the steel will rust in the presence of moisture.
- We assume that the set of possible classes is known, with labels $C_1, \ldots, C_K$.
- We have a training set of items, $\mathcal{D} = \{\mathbf{x}_n, y_n\}_{n=1}^N$, in which we know both the inputs $\mathbf{x}_n$ and the class $y_n \in \{C_1, ..., C_K\}$
- Once we have learned a classifier, we use it to predict the class of future items, given only the inputs for those items.
  - from which we will learn the class y^* for new data point $\mathbf{x}^*$

Approaches to Classification

- Classification problems can be solved in (at least) three ways:
  1. Learn how to directly produce a class from the inputs – that is, we learn some function that maps an input vector, $\mathbf{x}$, to a class, C_k.
  2. Learn a "discriminative" model for the probability distribution over classes for given inputs
     - that is, learn $p(C_k|\mathbf{x})$ as a function of $\mathbf{x}$. From $p(C_k|\mathbf{x})$ and a "loss function", we can make the best prediction for the class of an item.
  3. Learn a "generative" model for the probability distribution of the inputs for each class, i.e., learn $p(\mathbf{x}|C_k)$ for each class k.
     - From this, and the class probabilities, $p(C_k)$, we can find $p(C_k|\mathbf{x})$ using Bayes' Rule.
- Note that the last option above makes sense only if there is some well-defined distribution of items in a class.
  - This is not the case for the previous example of determining whether or not a type of steel will rust.

Loss functions and Classification

- Learning $p(C_k|\mathbf{x})$ allows one to make a prediction for the class in a way that depends on a "loss function", which says how costly different kinds of errors are.
- We define L_{kj} to be the loss we incur if we predict that an item is in class C_j when it is actually in class C_k. We'll assume that losses are non-negative and that $L_{kk} = 0$ for all k (i.e., there's no loss when the prediction is correct). Only the relative values of losses will matter.
  - If all errors are equally bad, we would let L_{kj} be the same for all $k \neq j$.
- **Example:** Giving a loan to someone who doesn't pay it back (class C_1) is much more costly than not giving a loan to someone who would pay it back (class C_0).
  - for example, set $L_{01} = 1$ and $L_{10} = 20$.
  - Note that in this example we should define the loss function to account both for monetary consequences (money not repaid, or interest not earned) and other effects that don't have immediate monetary consequences, such as customer dissatisfaction when their loan is not approved.

Predicting to Minimize Expected Loss

- A basic principle of decision theory is that we should take the action (here, make the prediction) that minimizes the expected loss, according to our probabilistic model.
- If we predict that an item with inputs $\mathbf{x}$ is in class C_j, the expected loss is

$$\mathcal{L}_j = \sum_{k=1}^{K} L_{kj} p(C_k|\mathbf{x}) = \sum_{k=1, k \neq j}^{K} L_{kj} p(C_k|\mathbf{x})$$

- We should predict that this item in the class, C_j, for which this expected loss is smallest.
  - The minimum might not be unique, in which case more than one prediction would be optimal.
- If all errors are equally bad (say loss of 1), the expected loss when predicting C_j is $1 - p(C_j|\mathbf{x})$, so we should predict the class which highest probability given $\mathbf{x}$.
- For binary classification ($K = 2$, with classes labelled by 0 and 1), minimizing expected loss is equivalent to predicting that an item is in class 1 if

$$\frac{p(C_1|\mathbf{x})}{p(C_0|\mathbf{x})} \frac{L_{10}}{L_{01}} > 1$$

Outline

Introduction

Evaluation & metrics

Generative Models for Classification

Discriminative Models for Classification

Decision Trees

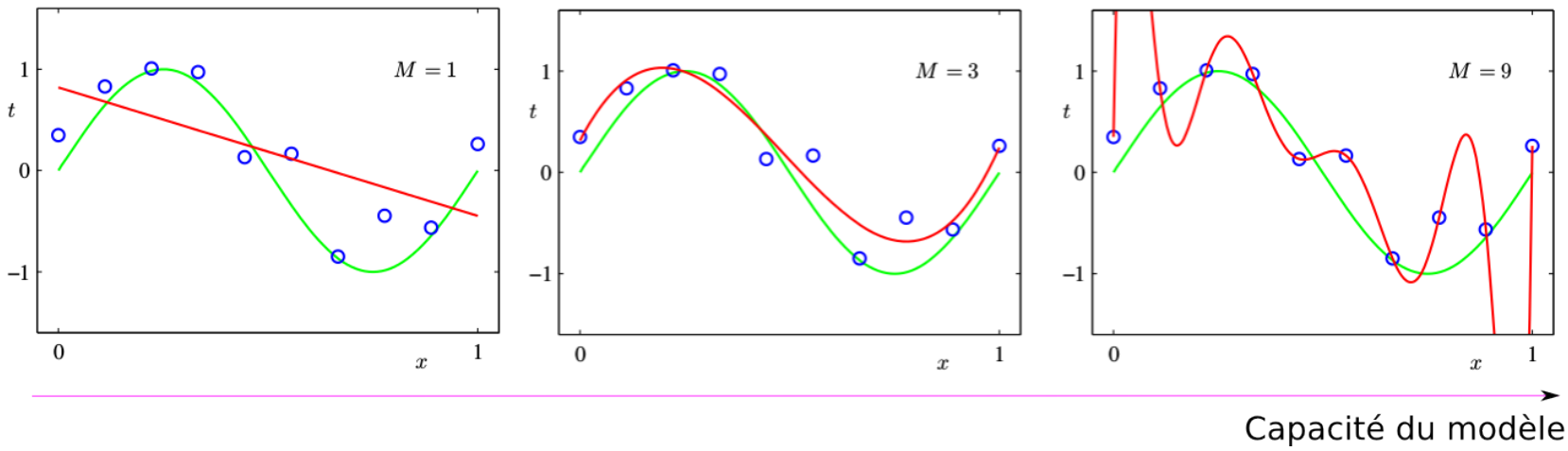
Large margin classifiers

Ensemble Methods

How to validate a classification?

- Unbiased dataset.
- K-fold cross validation.
- Metrics
  - Confusion matrix
  - Roc Curve

Has it trained?



[Bishop 2006]

How to detect over or under-fitting?

- What is "to fail"?
- Let f be the function learned by the algorithm.
- We define a loss function L :
  Example classification : **0-1 Loss**

$$L : \mathcal{C} \times \mathbb{X} \rightarrow 0; 1, \tag{1}$$

$$\left(c^{(i)}, f\left(\mathbf{x}^{(i)}\right)\right) \rightarrow \begin{cases} 0 & \text{if } c^{(i)} = f\left(\mathbf{x}^{(i)}\right) \\ 1 & \text{else} \end{cases}, \tag{2}$$

with $f\left(\mathbf{x}^{(i)}\right)$ the predicted class by f for the example $\mathbf{x}^{(i)}$ whose true class is c_i.

How to detect over or under-fitting?

- We cannot assume that $\mathcal{D} = \left(\mathbf{x}^{(i)}, y^{(i)}\right)_{i=1}^{n}$ describe all possible cases.
- We can assume that $\mathcal{D}$ is **independent and identically distributed** under the data generating function $p_{X,Y}$.
- The goal is to minimize the loss under $p_{X,Y}$ (generalization error):

$$Err_{\text{gen}}(f) = \mathbb{E}_{X,Y}[L] = \int_{\mathbb{X}} \int_{\mathbb{Y}} L(y, f(\mathbf{x}))\, dp_{X,Y}. \tag{3}$$

- If the learning algorithm is parametric then a candidate function h is in bijection with a parameter vector $\boldsymbol{\theta}$.

  $\rightarrow$ f is the function corresponding to the best $\boldsymbol{\theta}$!

How to detect over or under-fitting?

→ By estimating Err_{gen}

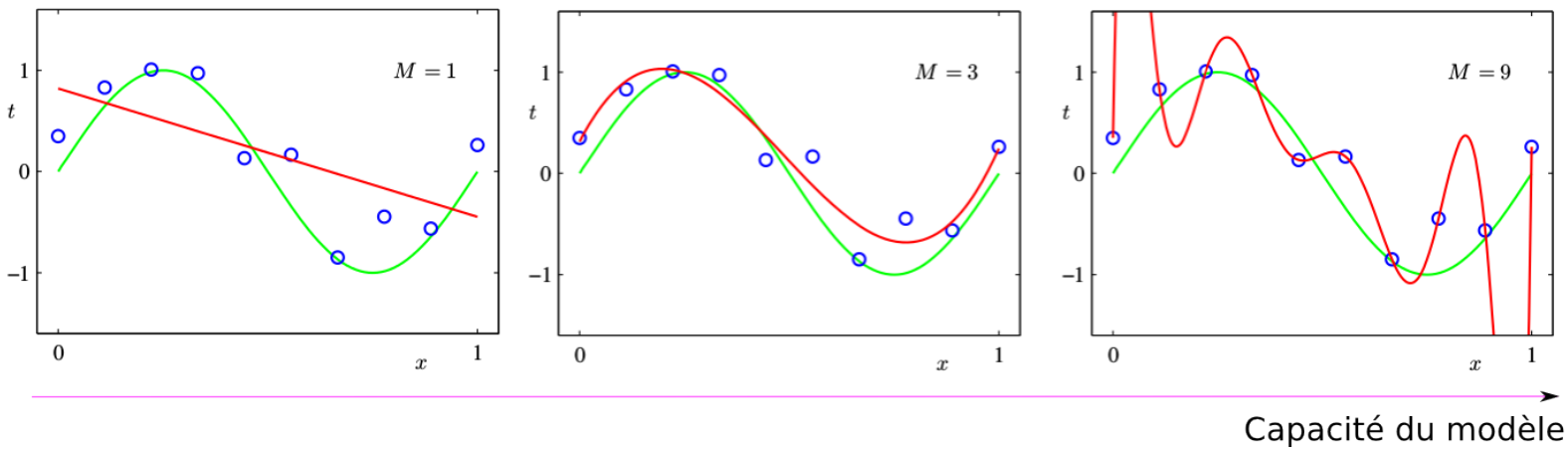
- The loss cannot be explicitly compute, the distribution is not knwon.
- In supervised case, we can compute the empirical loss (train error).

$$Err_{\text{train}}\left(f, \mathcal{D}\right) = \frac{1}{N}\sum_{i=1}^{N} L\left(y_i, f\left(\mathbf{x}_i\right)\right). \tag{4}$$

How to detect over or under-fitting?

→ By estimating Err_{gen}

- The train error decrease with the model capacity.
- It allows to detect under-fitting but not over-fitting!
- The problem is that $\mathcal{D}$ are from $p_{X,Y}$ but the predictive function f is computed from $\mathcal{D}$.
- Err_{train} is not a good estimator of Err_{gen}.

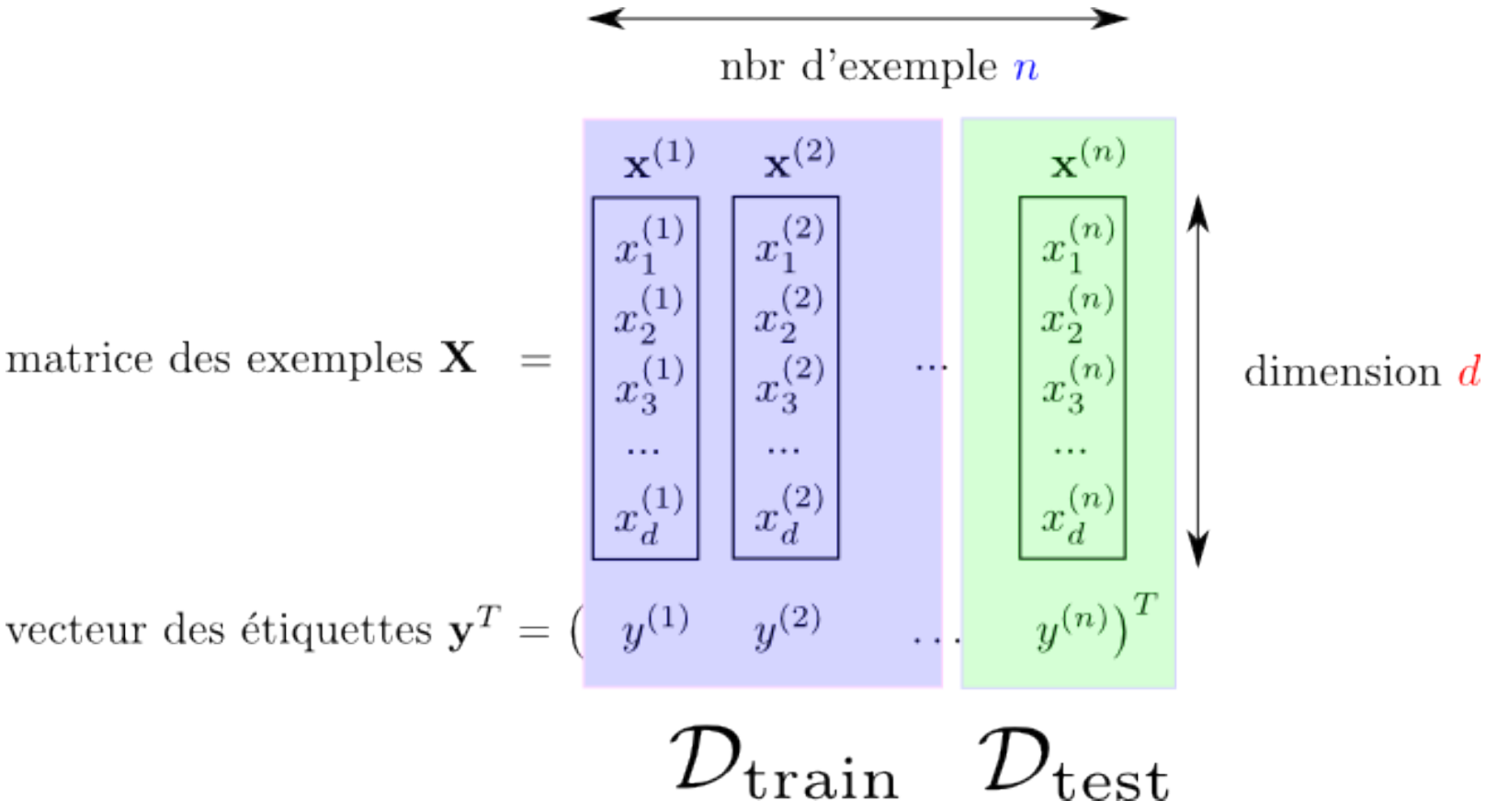


[Bishop 2006]

How to detect over or under-fitting?

→ By estimating Err_{gen}

- Approach : split $\mathcal{D}$ int 2 !



How to detect over or under-fitting?

→ By estimating Err_{gen}

- The training set $\mathcal{D}_{\text{train}}$ is used to estimate f.
- The test set $\mathcal{D}_{\text{test}}$ is used to estimate Err_{gen} after.

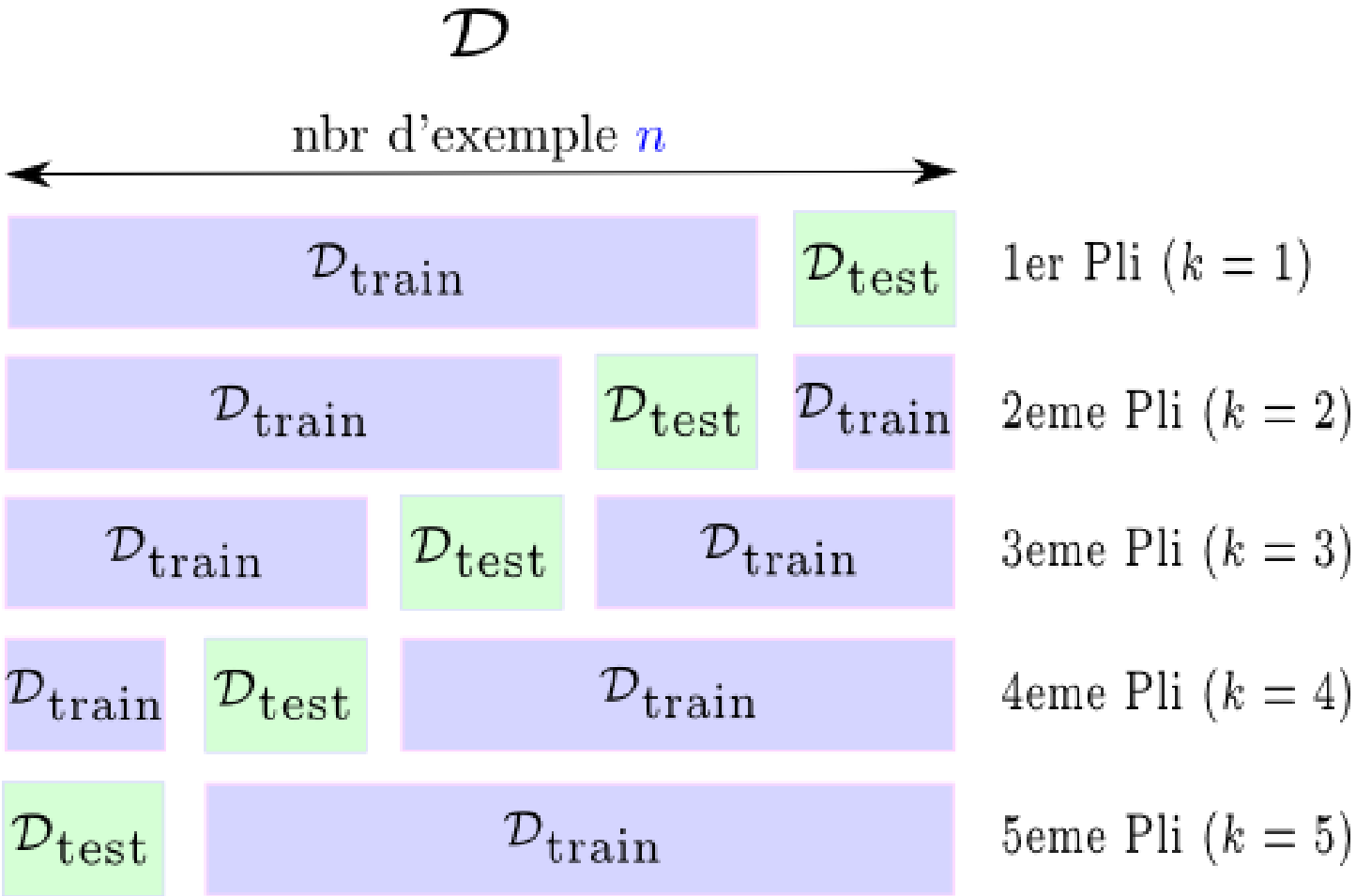
$$Err_{\text{test}}\left(f, \mathcal{D}\right) = \frac{1}{\sharp \mathcal{D}_{\text{test}}} \sum_{(y_i, f(\mathbf{x}_i)) \in \mathcal{D}_{\text{test}}} L\left(y_i, f\left(\mathbf{x}_i\right)\right). \tag{5}$$

- Err_{test} small if there is no ove or under-fitting!
- Err_{test} is pessimist about learning quality because less data are available for finding f.
- Hold out protocol.

How to detect over or under-fitting?

→ By estimating Err_{gen}

- To improve the estimation: multiple and different splits.
- K fold cross-validation.



How to detect over or under-fitting?

→ By estimating Err_{gen}

- Let $\mathcal{D}_{\text{test}}^{(k)}$ the test set at k^{th} fold.
- Test error becomes:

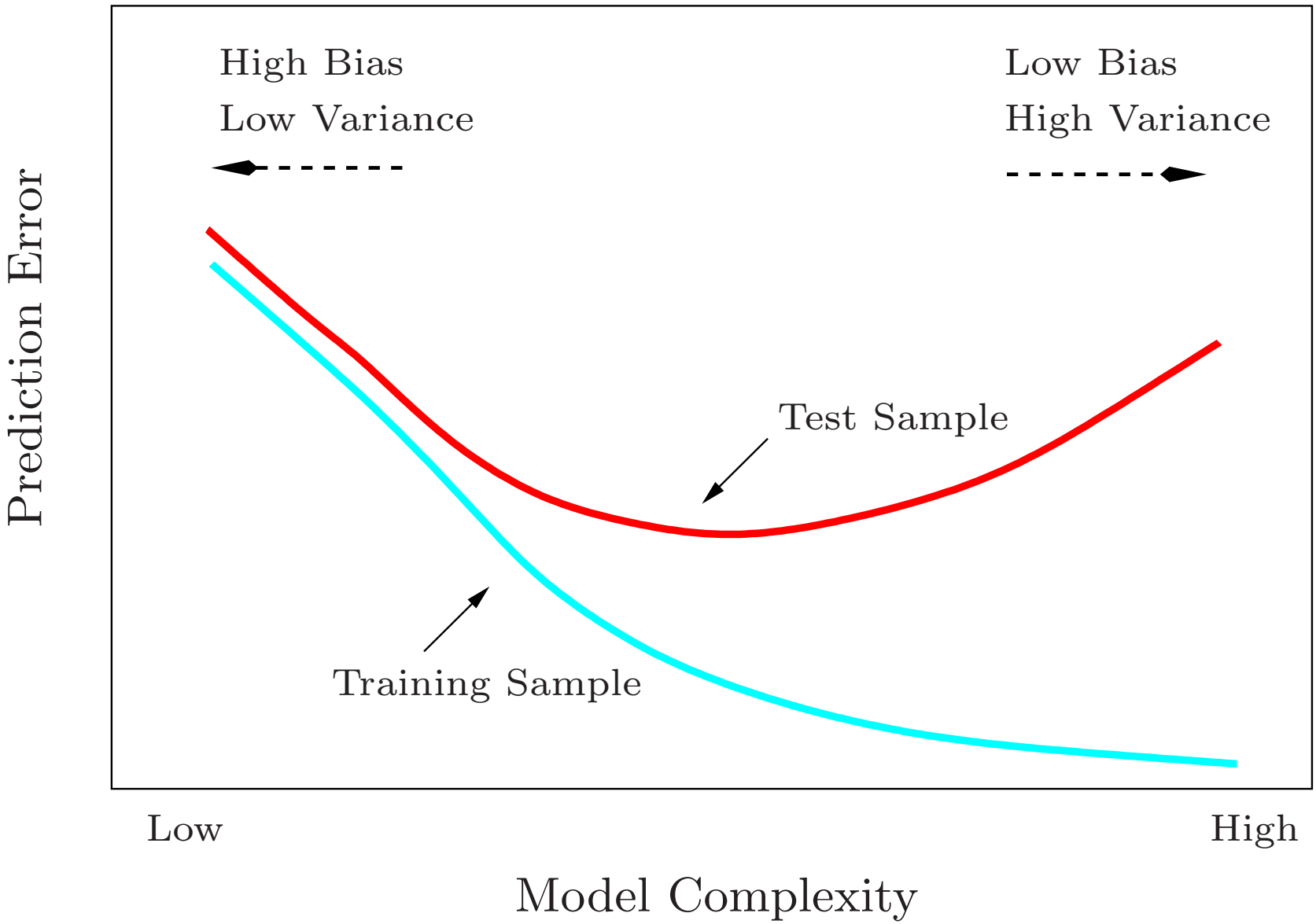
$$Err_{\text{test}}\left(f, \mathcal{D}\right) = \frac{1}{n} \sum_{k=1}^{K} \sum_{(y_i, f(\mathbf{x}_i)) \in \mathcal{D}_{\text{test}}^{(k)}} L\left(y_i, f\left(\mathbf{x}_i\right)\right). \tag{6}$$

- K-CV requires K learning phase → huge cost.
- $\mathcal{D}$ is randomly split but the same classes example proportion is maintained.
- Logical extrem : $K = n$, Leave-one-out cross validation (**LOOCV**).

How to detect over or under-fitting?

→ By estimating Err_{gen}

- Error is "U" shaped.



How to detect over or under-fitting?

$\rightarrow$ By controling Err_{gen}

Frequentist approach: find $|Err_{\text{gen}} - Err_{\text{train}}|$ bounds.

- Let $\mathcal{H}$ be the set of model hypothesis = set of learning functions.
- The $\mathcal{H}$ volum depends on hyperparameters.
- Learning consist of picking f in $\mathcal{H}$ in order to have:

$$Err_{\text{gen}}(f) = \arg\min_{h \in \mathcal{H}} Err_{\text{gen}}(h).$$

Can we learn from example?

Can we learn from example?

Is it possible to learn in theory ?

Can we learn from example?

Is it possible to learn in theory ? Yes,by ensuring that $Err_{gen} \approx 0$.

Can we learn from example?

Is it possible to learn in theory ? Yes,by ensuring that $Err_{gen} \approx 0$.

Is it possible to learn practically ?

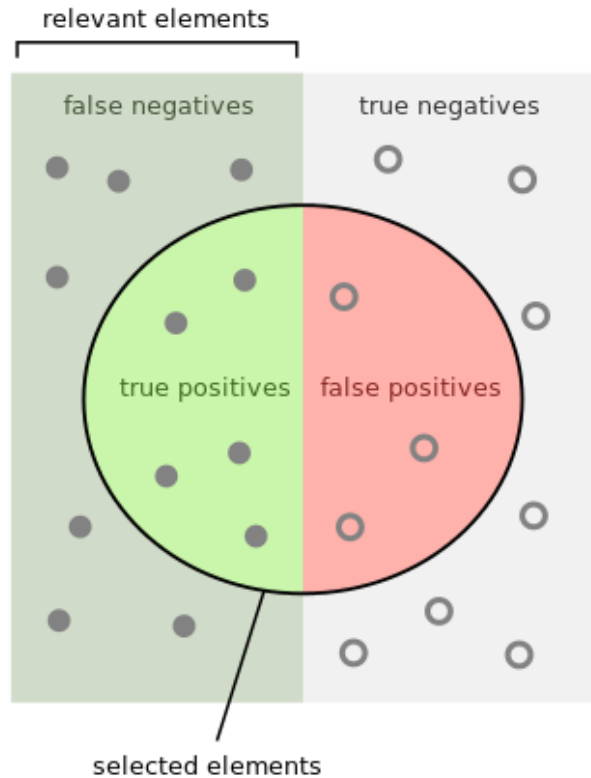
Can we learn from example?

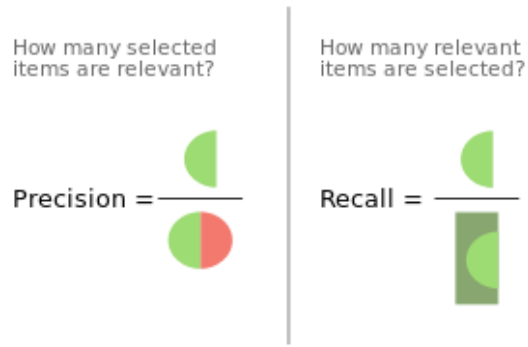
Is it possible to learn in theory ? Yes,by ensuring that $Err_{gen} \approx 0$.

Is it possible to learn practically ? Yes,by ensuring that $Err_{train} \approx 0$ and $Err_{train} \approx Err_{gen}$.

Classification metrics

- Category
  - True positive: TP
  - False positive: FP
  - False negative: FN
  - True negative: TN
- Recall, sensitivity:
  - $\frac{TP}{TP+FN}$.
- Specificity:
  - $\frac{TN}{TN+FP}$.
- Precision:
  - $\frac{TP}{TP+FP}$.
- F_1 score:
  - $F_1 = \frac{2TP}{2TP+FP+FN}$.
- MCC score:
  - $MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}$.





Confusion matrix

Example:

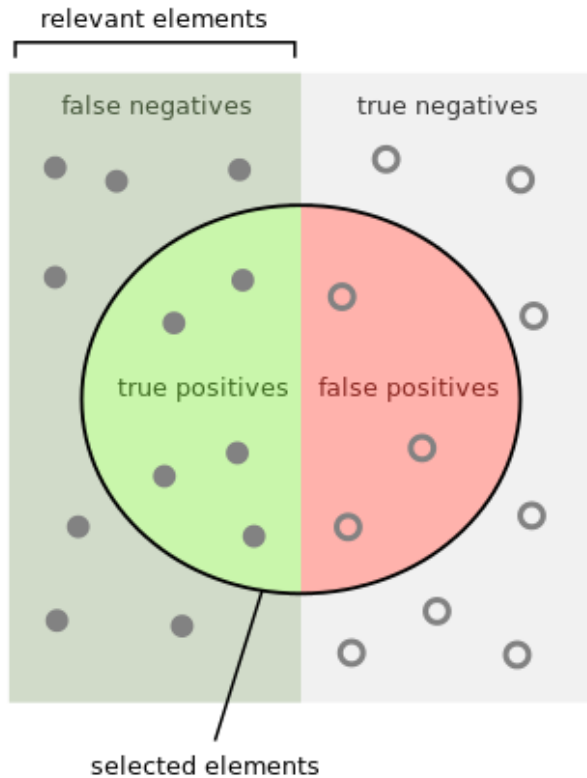
	Sick	Not sick	Total
Estimated true	190	210	400
Estimated false	10	3590	3600
Total	200	3800	4000

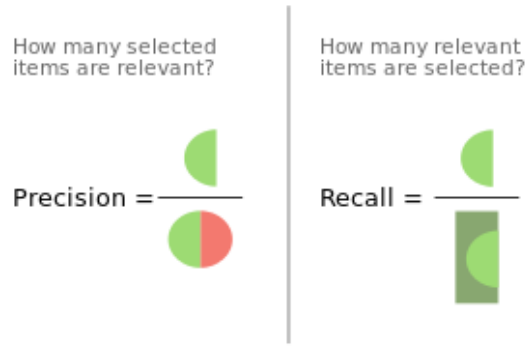
The probability of not being sick after negative estimation is $3590/3600 = 99.7\%$.

- Good for detection !

The probability of being sick after positive estimation is $190/400 = 47.5\%$.

- Bad for diagnostic !



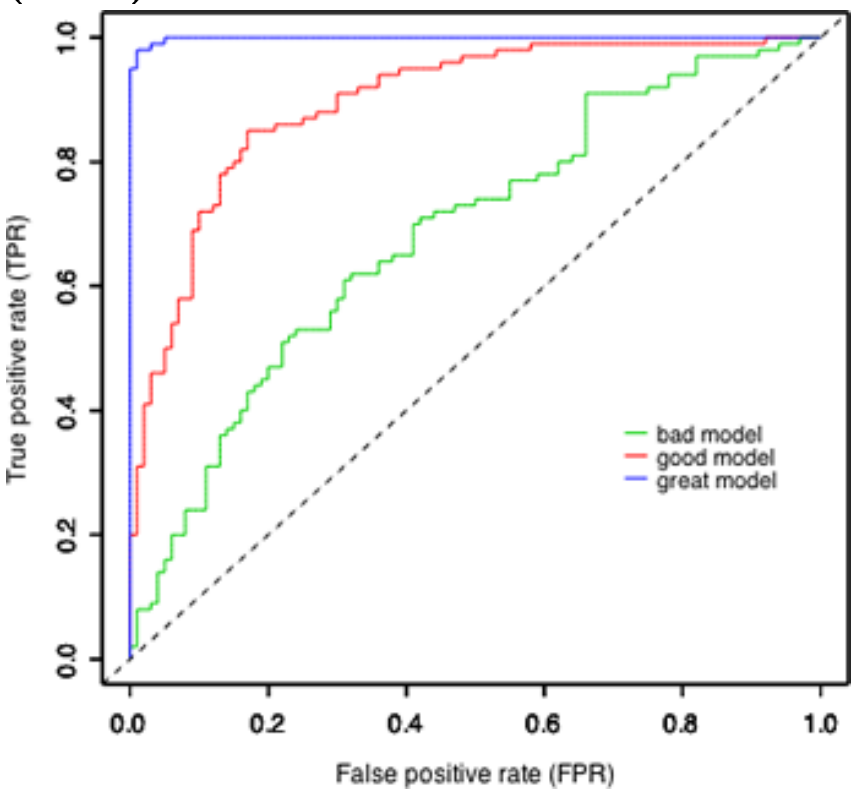


Confusion matrix

		True condition	
	Total population	Condition positive	Condition negative
Predicted condition	Predicted condition positive	**True positive**	**False positive**, Type I error
	Predicted condition negative	**False negative**, Type II error	**True negative**

Metrics two classes

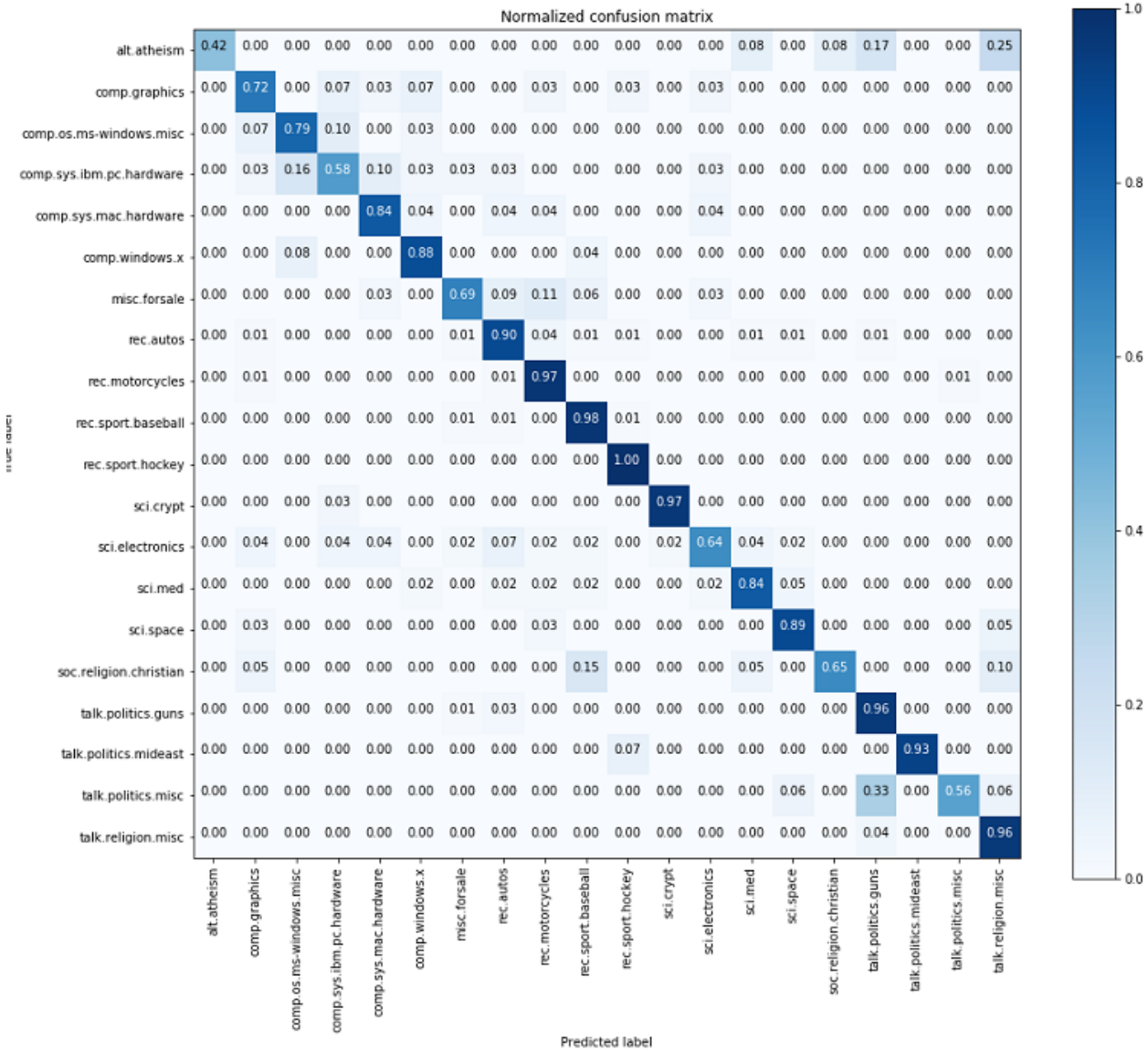
Receiver Operating Characteristic (ROC) curbe:



Area Under The Curve (AUC):

- $\in [0, 1]$.
- scale invariance.
- threshold invariant.

Confusion matrix multi-class



Metrics multi-class

Multi-class $\mathcal{C} = \{\mathcal{C}_0, \ldots \mathcal{C}_n\}$, naive approach: use binary classification

- One-versus-one: $\mathcal{C}_i$ vs.$\mathcal{C}_j$.
- One-versus-all: $\mathcal{C}_i$ vs. $\{\mathcal{C}_{j \neq i}\}$

Metrics multi-label

Multi-label classification:

- A misclassification is no longer a hard wrong or right.
- A prediction containing a subset of the actual classes should be considered better than a prediction that contains none of them, i.e., predicting two of the three labels correctly this is better than predicting no labels at all.

Score:

- Micro-averaging:

  $$\text{precision}(D) = \frac{\sum_{\mathcal{C}_i \in \mathcal{C}} TPs(\mathcal{C}_i)}{\sum_{\mathcal{C}_i \in \mathcal{C}} TPs(\mathcal{C}_i) + FPs(\mathcal{C}_i)}.$$

  $$\text{recall}(D) = \frac{\sum_{\mathcal{C}_i \in \mathcal{C}} TPs(\mathcal{C}_i)}{\sum_{\mathcal{C}_i \in \mathcal{C}} TPs(\mathcal{C}_i) + FNs(\mathcal{C}_i)}.$$

- Macro-averaging: average of score for each class (One versus all).
- Hamming-loss
- Exact match ratio

Outline

Introduction

Evaluation & metrics

Generative Models for Classification

Discriminative Models for Classification

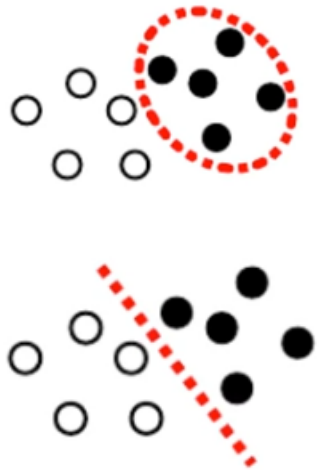
Decision Trees

Large margin classifiers

Ensemble Methods

Generative *vs.* Discriminative

- Generative
  - Probabilistic "model" of each class
  - Decision boundary:
    where one model becomes more likely
  - Natural use of unlabeled data
- Discriminative
  - Focus on the decision boundary
  - More powerful with a lots of examples
  - Not designed to use unlabeled data
  - Only supervised tasks



Linear classification

There are several different approaches to linear classification :

- logistic regression
- linear discriminant analysis
- separating hyperplanes (perceptron, linear support vector machine)

- logisitic regression, we model : $p(\text{class}|\text{data})$
- discrimant analysis, we model : $p(\text{data}|\text{class})$
- separating hyperplanes : directly optimize a linear decision boundary without any probabilistic model

Linear classification

There are several different approaches to linear classification :

- logistic regression
- linear discriminant analysis
- separating hyperplanes (perceptron, linear support vector machine)

- logisitic regression, we model : $p(\text{class}|\text{data})$
- discrimant analysis, we model : $p(\text{data}|\text{class})$
- separating hyperplanes : directly optimize a linear decision boundary without any probabilistic model

Linear classification

In classification, given a datapoint $\mathbf{x}$ we want p(class$|\mathbf{x}$).

It can be directly estimate by regression...

OR
We can assume a model for $p(\mathbf{x}|\text{class})$ and some prior $p(\text{class})$.
Then using Bayes' rule we get $p(\text{class}|\mathbf{x})$!

Linear classification

In classification, given a datapoint $\mathbf{x}$ we want p(class|$\mathbf{x}$).

It can be directly estimate by regression...

OR
We can assume a model for $p(\mathbf{x}|\text{class})$ and some prior $p(\text{class})$.
Then using Bayes' rule we get $p(\text{class}|\mathbf{x})$!

Classification from Generative Models Using Bayes' Rule

- In the **generative model** approach to classification, we learn models from the training data for the probability or pdf of the inputs, $\mathbf{x}$, for items in each of the possible classes, C_k
  - **that is, we learn models for $p(\mathbf{x}|C_k)$ for $k = 1, \ldots, K$.**
- To do classification, we instead need $p(C_k|\mathbf{x})$. We can get these conditional class probabilities using Bayes' Rule:

$$p(C_k|\mathbf{x}) = \frac{p(C_k)p(\mathbf{x}|C_k)}{\sum_{j=1}^{K} p(C_j)p(\mathbf{x}|C_j)}$$

- Here, $p(C_k)$ is the prior probability of class C_k.
  - We can easily estimate these probabilities by the frequencies of the classes in the training data.
  - Alternatively, we may have good information about $p(C_k)$ from other sources.
- For binary classification, with classes C_0 and C_1, we get

$$\begin{aligned} p(C_1|\mathbf{x}) &= \frac{p(C_1)p(\mathbf{x}|C_1)}{p(C_0)p(\mathbf{x}|C_0) + p(C_1)p(\mathbf{x}|C_1)} \\ &= \frac{1}{1 + p(C_0)p(\mathbf{x}|C_0)/p(C_1)p(\mathbf{x}|C_1)} \end{aligned}$$

Naive Bayes Models for Binary Inputs

- When the inputs is a binary vector of dimension p (i.e., $\mathbf{x}$ is a vectors of 1's and 0's) we can use the following simple generative model:

$$p(\mathbf{x}|C_k) = \prod_{i=1}^{p} \theta_{ki}^{\mathbf{x}_i} (1 - \theta_{ki}^{\mathbf{x}_i})^{1-\mathbf{x}_i}$$

  - it assumes independence across dimensions
    $p(\mathbf{x}|C_k) = p(x_1|C_k) \cdot p(x_2|C_k) \cdot \ldots \cdot p(x_p|C_k)$
- Here, θ_{ki} is the estimated probability that input i will have the value 1 in items from class k.
  - The maximum likelihood estimate for θ_{ki} is simply the fraction of 1's in training items that are in class k.
- This is called the *naive* Bayes model
  - "Bayes" because we use it with Bayes's Rule to do classification
  - "naive" because this model assumes that inputs are **independent given the class**, which is something a naive person might assume, though it's usually not true.
- It's easy to generalize naive Bayes models to discrete inputs with more than two values, and further generalizations (keeping the independence assumption) are also possible.

Binary Classification using Naive Bayes Models with Binary Inputs

- When there are two classes (C_0 and C_1) and binary inputs, applying Bayes' Rule with a naive Bayes classifier gives the following probability for C_1 given $\mathbf{x}$, (using $p(\mathbf{x}) = p(C_0)p(\mathbf{x}|C_0) + p(C_1)p(\mathbf{x}|C_1)$)

$$\begin{aligned} p(C_1|\mathbf{x}) &= \frac{p(C_1)p(\mathbf{x}|C_1)}{p(C_0)p(\mathbf{x}|C_0) + p(C_1)p(\mathbf{x}|C_1)} \\ &= \frac{1}{1 + p(C_0)p(\mathbf{x}|C_0)/p(C_1)p(\mathbf{x}|C_1)} \\ &= \frac{1}{1 + \exp(-a(\mathbf{x}))} \end{aligned}$$

where

$$\begin{aligned} a(\mathbf{x}) &= \log\left(\frac{p(C_1)p(\mathbf{x}|C_1)}{p(C_0)p(\mathbf{x}|C_0)}\right) \\ &= \log\left(\frac{p(C_1)}{p(C_0)}\prod_{i=1}^{p}\left(\frac{\theta_{1i}}{\theta_{0i}}\right)^{\mathbf{x}_i}\left(\frac{1-\theta_{1i}}{1-\theta_{0i}}\right)^{1-\mathbf{x}_i}\right) \\ &= \log\left(\frac{p(C_1)}{p(C_0)}\prod_{i=1}^{p}\frac{1-\theta_{1i}}{1-\theta_{0i}}\right) + \sum_{i=1}^{p}\mathbf{x}_i\log\left(\frac{\theta_{1i}/(1-\theta_{1i})}{\theta_{0i}/(1-\theta_{0i})}\right) \end{aligned}$$

Gaussian Generative Models: Gaussian Discriminant Analysis (GDA)

- When the **inputs are real-valued** (instead of binary), a Gaussian model for the distribution of inputs in each class may be appropriate. The conditional densities in this generative classifier will thus be of the form

$$p(\mathbf{x}|C_k) = \mathcal{N}(\mathbf{x}|\mu_k, \Sigma_k) = (2\pi)^{-p/2}|\Sigma_k|^{-1/2} \exp\left(-\frac{1}{2}(\mathbf{x}-\mu_k)^T\Sigma_k^{-1}(\mathbf{x}-\mu_k)\right),$$

where μ_k and Σ_k are an estimate of respectively the mean vector and covariance matrix for class C_k.

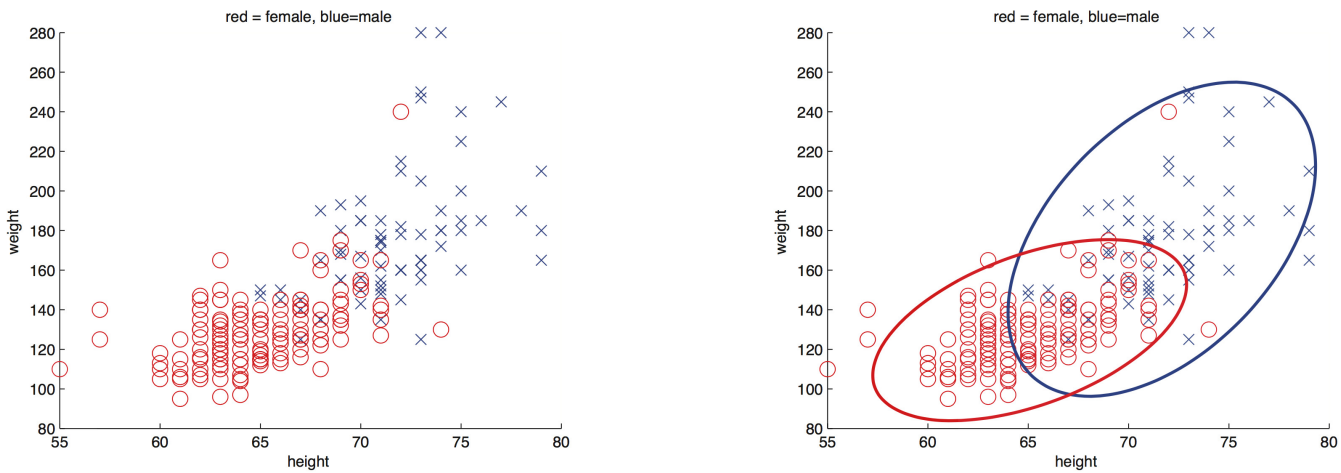
- The resulting technique is called (Gaussian) **Discriminant Analysis** (GDA)
  - note that **it is a generative, not discriminative, classifier** !
- If Σ_k is diagonal (no correlation considered between input features), this technique is equivalent to the Naive Bayes Classifier.

Gaussian Generative Models: Gaussian Discriminant Analysis (GDA)

- A feature vector $\mathbf{x}$ is classified using the following rule based on the posterior:

$$\widehat{C}_k^* = \arg\max_{C_k} p(C_k|\mathbf{x}) = \arg\max_{C_k} \log[p(C_k)p(\mathbf{x}|C_k)] = \arg\max_{C_k} [\log p(C_k)] + \log p(\mathbf{x}|C_k)]$$

with $\log p(\mathbf{x}|C_k) = -1/2 \log |\Sigma_k|^{-1/2} - \frac{1}{2}(\mathbf{x} - \mu_k)^T \Sigma_k^{-1} (\mathbf{x} - \mu_k)$



(a) (b) *(a)*

Height/weight data. (b) Visualization of 2d Gaussians fit to each class. 95% of the probability mass is inside the ellipse.

Gaussian Generative Models: Gaussian Discriminant Analysis (GDA)

- In practice, the mean vector and covariance matrix for each class are obtained using the maximum likelihood criterion - thus leading to the simple calculation:

$$\mu_k = \frac{1}{N_k} \sum_{j \in \Omega_k} \mathbf{x}_j \qquad \text{and} \qquad \Sigma_k = \frac{1}{N_k} \sum_{j \in \Omega_k} (\mathbf{x}_j - \mu_k)(\mathbf{x}_j - \mu_k)^T$$

  where $\Omega_k = \{i = 1, \ldots, N | y_i = C_k\}$ corresponds to the indices from the training set that belongs to the k-th class
  - $N_k = |\Omega_k|$ is the number of data from this class C_k

Gaussian densities

$$p(C_k|\mathbf{x}) = \frac{p(C_k)p(\mathbf{x}|C_k)}{\sum_{j=1}^{K} p(C_j)p(\mathbf{x}|C_j)}$$

$$\rightarrow p(C_k|\mathbf{x}) \sim f_k(\mathbf{x})\pi_k$$

$$f_k(\mathbf{x}) = \frac{1}{\sqrt{(2\pi)^p det\Sigma_k}} exp\left[-\frac{1}{2}(\mathbf{x}-\mu_k)^T\Sigma_k^{-1}(\mathbf{x}-\mu_k)\right].$$

Quadratic Discriminant Analysis (QDA)

Consider a binary classification problem with $\pi_1 = \pi_2 = \frac{1}{2}$.

The decision boundary is given by $p(\text{class } 1|\mathbf{x}) = p(\text{class } 2|\mathbf{x})$:

$$
\begin{aligned}
&-\frac{p}{2}log(2\pi) - \frac{1}{2}\log\det\Sigma_1 - \frac{1}{2}(\mathbf{x}-\mu_1)^T\Sigma_1^{-1}(\mathbf{x}-\mu_1) \\
&= -\frac{p}{2}log(2\pi) - \frac{1}{2}\log\det\Sigma_2 - \frac{1}{2}(\mathbf{x}-\mu_2)^T\Sigma_2^{-1}(\mathbf{x}-\mu_2)
\end{aligned}
$$

This is Quadratic Discriminant Analysis (QDA)

Quadratic Discriminant Analysis (QDA)

QDA boundary is given by $p(\text{class } 1|\mathbf{x}) = p(\text{class } 2|\mathbf{x})$:

$$\log \det\Sigma_1 - \frac{1}{2}(\mathbf{x}-\mu_1)^T\Sigma_1^{-1}(\mathbf{x}-\mu_1) = \log \det\Sigma_2 - \frac{1}{2}(\mathbf{x}-\mu_2)^T\Sigma_2^{-1}(\mathbf{x}-\mu_2)$$

Linear Discriminant Analysis (LDA)

QDA boundary is given by $p(\text{class } 1|\mathbf{x}) = p(\text{class } 2|\mathbf{x})$:

$$\log \det\Sigma_1 + (\mathbf{x} - \mu_1)^T \Sigma_1^{-1} (\mathbf{x} - \mu_1) = \log \det\Sigma_2 + (\mathbf{x} - \mu_2)^T \Sigma_2^{-1} (\mathbf{x} - \mu_2)$$

Let assume that $\Sigma_1 = \Sigma_2 = \Sigma$. Then :

$$(\mathbf{x} - \mu_1)^T \Sigma^{-1} (\mathbf{x} - \mu_1) = (\mathbf{x} - \mu_2)^T \Sigma^{-1} (\mathbf{x} - \mu_2)$$

Then :

$$2 \times \mathbf{x}^T \Sigma^{-1} (\mu_1 - \mu_2) = \underbrace{\mu_1^T \Sigma^{-1} \mu_1 - \mu_2^T \Sigma^{-1} \mu_2}_{\text{constante}}$$

$$\boxed{\mathbf{x}^T \Sigma^{-1} (\mu_1 - \mu_2) = \text{constante}}$$

Linear Discriminant Analysis (LDA)

QDA boundary is given by $p(\text{class } 1|\mathbf{x}) = p(\text{class } 2|\mathbf{x})$:

$$\log \det\Sigma_1 + (\mathbf{x} - \mu_1)^T \Sigma_1^{-1} (\mathbf{x} - \mu_1) = \log \det\Sigma_2 + (\mathbf{x} - \mu_2)^T \Sigma_2^{-1} (\mathbf{x} - \mu_2)$$

Let assume that $\Sigma_1 = \Sigma_2 = \Sigma$. Then :

$$(\mathbf{x} - \mu_1)^T \Sigma^{-1} (\mathbf{x} - \mu_1) = (\mathbf{x} - \mu_2)^T \Sigma^{-1} (\mathbf{x} - \mu_2)$$

Then :

$$2 \times \mathbf{x}^T \Sigma^{-1} (\mu_1 - \mu_2) = \underbrace{\mu_1^T \Sigma^{-1} \mu_1 - \mu_2^T \Sigma^{-1} \mu_2}_{\text{constante}}$$

$$\boxed{\mathbf{x}^T \Sigma^{-1} (\mu_1 - \mu_2) = \text{constante}}$$

Linear Discriminant Analysis (LDA)

$$\boxed{\mathbf{x}^T \Sigma^{-1}(\mu_1 - \mu_2) = \text{constante}}$$

This is linear projection of $\mathbf{x}$ onto the $\Sigma^{-1}(\mu_1 - \mu_2)$ direction.

LDA decision boundary :

$$\mathbf{x}^T \Sigma^{-1}(\mu_1 - \mu_2) = \frac{1}{2}\left(\mu_1^T \Sigma^{-1} \mu_1 - \mu_2^T \Sigma^{-1} \mu_2\right).$$

Remarks :

- Why does LDA use projection on $\Sigma^{-1}(\mu_1 - \mu_2)$ and not simply on $(\mu_1 - \mu_2)$?
- What happens if the covariance matrix is spherical? (*i.e.* $\Sigma = \sigma^2 \mathbf{I}$ *nearest centroid classifier*)

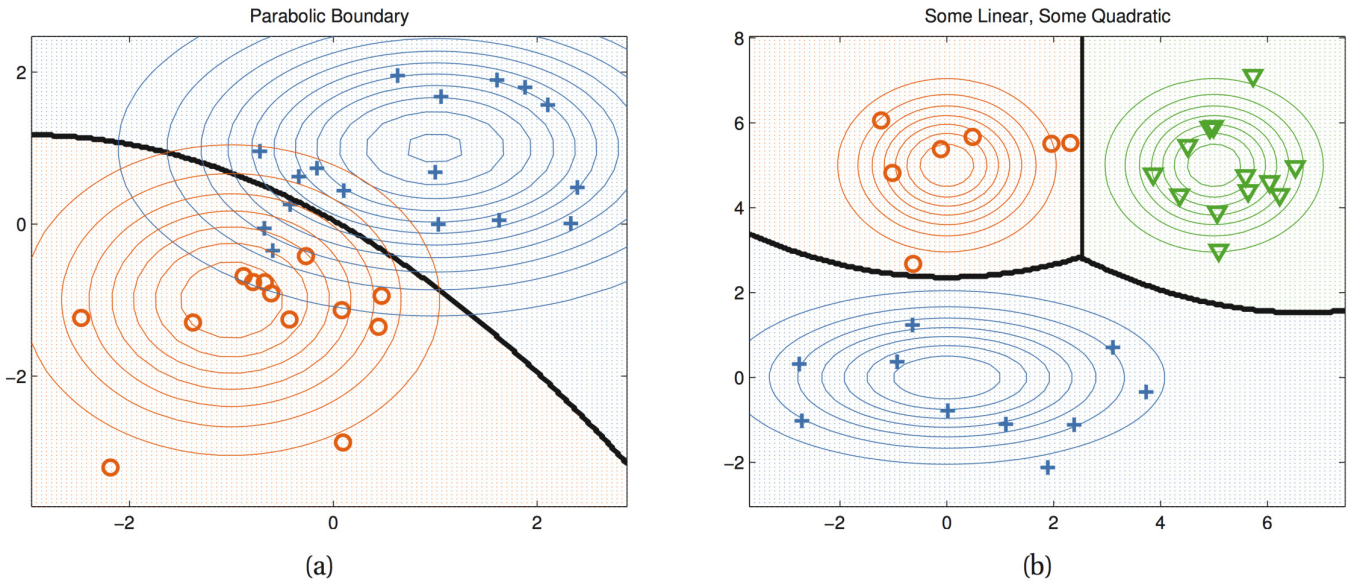
Gaussian Generative Models: Quadratic Discriminant Analysis (QDA)

- A feature vector $\mathbf{x}$ is classified using the following rule based on the posterior:

$$\widehat{C}_k^* = \arg\max_{C_k} p(C_k|\mathbf{x}) = \arg\max_{C_k} \log\left(p(C_k)p(\mathbf{x}|C_k)\right) = \arg\max_{C_k} \log p(C_k) + \log p(\mathbf{x}|C_k)$$

with $\log p(\mathbf{x}|C_k) = -1/2 \log |\Sigma_k|^{-1/2} - \frac{1}{2}(\mathbf{x}-\mu_k)^T \Sigma_k^{-1}(\mathbf{x}-\mu_k)$ (log-likelihood)

- Since in order to take the decision (thresholding) there is a quadratic function of the input vector $\mathbf{x}$, this technique is known as **quadratic discriminant analysis** (QDA)
- It is a particular case of GDA when the covariance matrices of the classes are different.



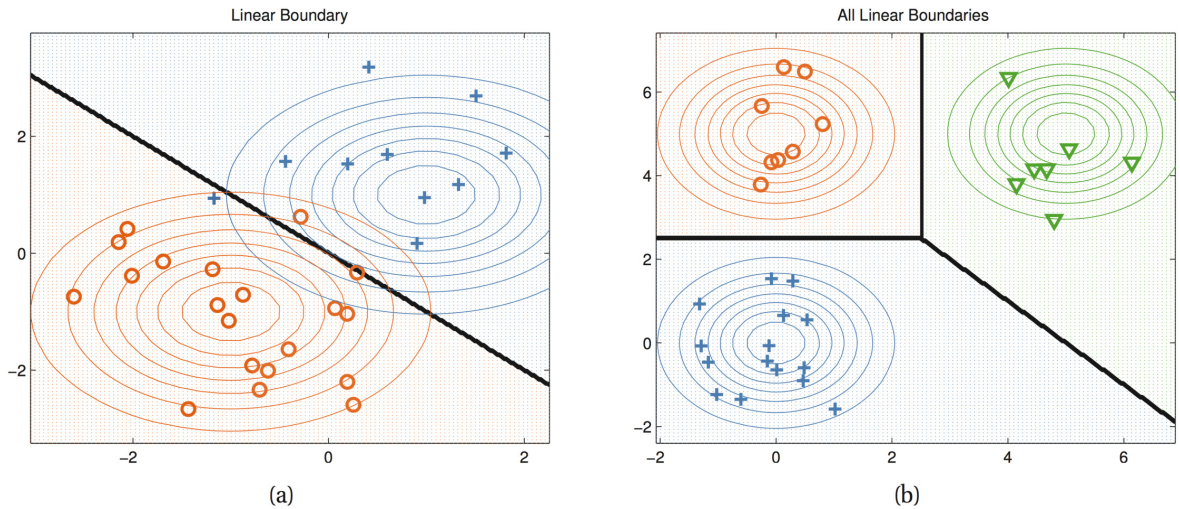
Decision boundaries obtained with a Quadratic Discriminant Analysis in 2D for the 2 and 3 class case.

Gaussian Generative Models: Linear Discriminant Analysis (LDA)

If now a special case in which the covariance matrices are tied and shared across classes, i.e $\Sigma_k = \Sigma$ $(\forall k)$. In this case the decision rule is simplified:

$$\begin{aligned}
\widehat{C}_k^* &= \arg\max_{C_k} p(C_k|\mathbf{x}) \\
&= \arg\max_{C_k} \log\left(p(C_k)p(\mathbf{x}|C_k)\right) \\
&= \arg\max_{C_k} \log p(C_k)] + \log p(\mathbf{x}|C_k) \\
&\arg\max_{C_k} \log p(C_k) - \frac{1}{2}(\mathbf{x}-\mu_k)^T\Sigma^{-1}(\mathbf{x}-\mu_k) \\
&\arg\max_{C_k} \log p(C_k) - \frac{1}{2}\mu_k^T\Sigma^{-1}\mu_k + \mu_k^T\Sigma^{-1}\mathbf{x}
\end{aligned}$$

thus producing a linear function of the input vector $\mathbf{x}$ (simplifying the decision rule)!



Decision boundaries obtained with a Linear Discriminant Analysis in 2D for the 2 and 3 class case.

Gaussian Generative Models: Linear Discriminant Analysis (LDA)

For QDA and LDA we need to know π_k, μ_k and Σ_k.

- In practice, the mean vector and covariance matrix of the LDA are obtained using the maximum likelihood criterion - thus leading to the simple calculation:

$$\hat{\mu}_k = \frac{1}{N_k} \sum_{j \in \Omega_k} \mathbf{x}_j$$

$$\hat{\Sigma}_k = \frac{1}{N_k - 1} \sum_{i \in C_k} (\mathbf{x}_i - \mu_k)(\mathbf{x}_i - \mu_k)^T$$

$$\hat{\pi}_k = \frac{N_k}{N}$$

$$\hat{\Sigma} = \sum_{k=1}^{k} \frac{N_k}{N} \Sigma_k$$

where Σ_k corresponds to the covariance matrix of the k-th class (computed like in the QDA).

Binary Classification using LDA

- For binary classification, we can now apply Bayes' Rule to get the probability of class 1 from a Gaussian model with the same covariance matrix in each class.
- As for the naive Bayes model:

$$p(C_1|\mathbf{x}) = \frac{1}{1 + \exp(-a(\mathbf{x}))}$$

where

$$a(\mathbf{x}) = \log\left(\frac{p(C_1)p(\mathbf{x}|C_1)}{p(C_0)p(\mathbf{x}|C_0)}\right)$$

- Substituting the Gaussian densities, we get

$$\begin{aligned} a(x) &= \log\left(\frac{p(C_1)}{p(C_0)}\right) + \log\left(\frac{\exp\left(-\frac{1}{2}(\mathbf{x}-\mu_1)^T\Sigma^{-1}(\mathbf{x}-\mu_1)\right)}{\exp\left(-\frac{1}{2}(\mathbf{x}-\mu_0)^T\Sigma^{-1}(\mathbf{x}-\mu_0)\right)}\right) \\ &= \log\left(\frac{p(C_1)}{p(C_0)}\right) + \frac{1}{2}\left(\mu_0^T\Sigma^{-1}\mu_0 - \mu_1\Sigma^{-1}\mu_1\right) + \mathbf{x}^T\left(\Sigma^{-1}(\mu_1 - \mu_0)\right) \end{aligned}$$

- As denoted before the quadratic terms of the form $\mathbf{x}^T\Sigma^{-1}\mathbf{x}$ cancel, producing a linear function of the inputs, as was also the case for naive Bayes models.

Outline

Introduction

Evaluation & metrics

Generative Models for Classification

Discriminative Models for Classification

Decision Trees

Large margin classifiers

Ensemble Methods

Gaussian Process Models for Classification*

- A Gaussian process logistic regression model for data $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$ with the y_i being binary can be expressed as

$$\begin{aligned} \theta &\sim \ldots \\ f &\sim \mathcal{GP}(\theta) \\ y_i|\mathbf{x}_i, f &\sim \text{Bernoulli}(1/(1+\exp(-f(\mathbf{x}_i)))) \end{aligned}$$

  where θ represents all the parameters of the Gaussian process's covariance and mean functions.

- Alternatively, we could use a probit model for the observation, with

$$y_i|\mathbf{x}_i, f \sim \text{Bernoulli}(\Phi(f(\mathbf{x}_i)))$$

- However, with neither of these models can we use simple matrix operations to evaluate $p(y|\mathbf{x}, \theta)$, or to predict a new y^* by $p(y^*|\mathbf{x}^*, y, \mathbf{x}, \theta)$.

The Latent Gaussian Process*

- To do computations for Gaussian process classification, we need to explicitly represent the "latent variables" $z_i = f(\mathbf{x}_i)$
- Using matrix operations, we can compute the joint density of the latent variables and observed responses (for given θ):

$$\begin{aligned} &p(z_1, \ldots, z_N, y_1, \ldots, y_N | \mathbf{x}_1, \ldots, \mathbf{x}_N, \theta) \\ &\quad = p(z_1, \ldots, z_N | \mathbf{x}_1, \ldots, \mathbf{x}_N, \theta) p(y_1, \ldots, y_N | z_1, \ldots, z_N) \end{aligned}$$

- The first factor above (the prior for latent variables) is Gaussian; the second (the likelihood for these latent variables) is a simple product of Bernoulli probabilities (from a logit or probit model).

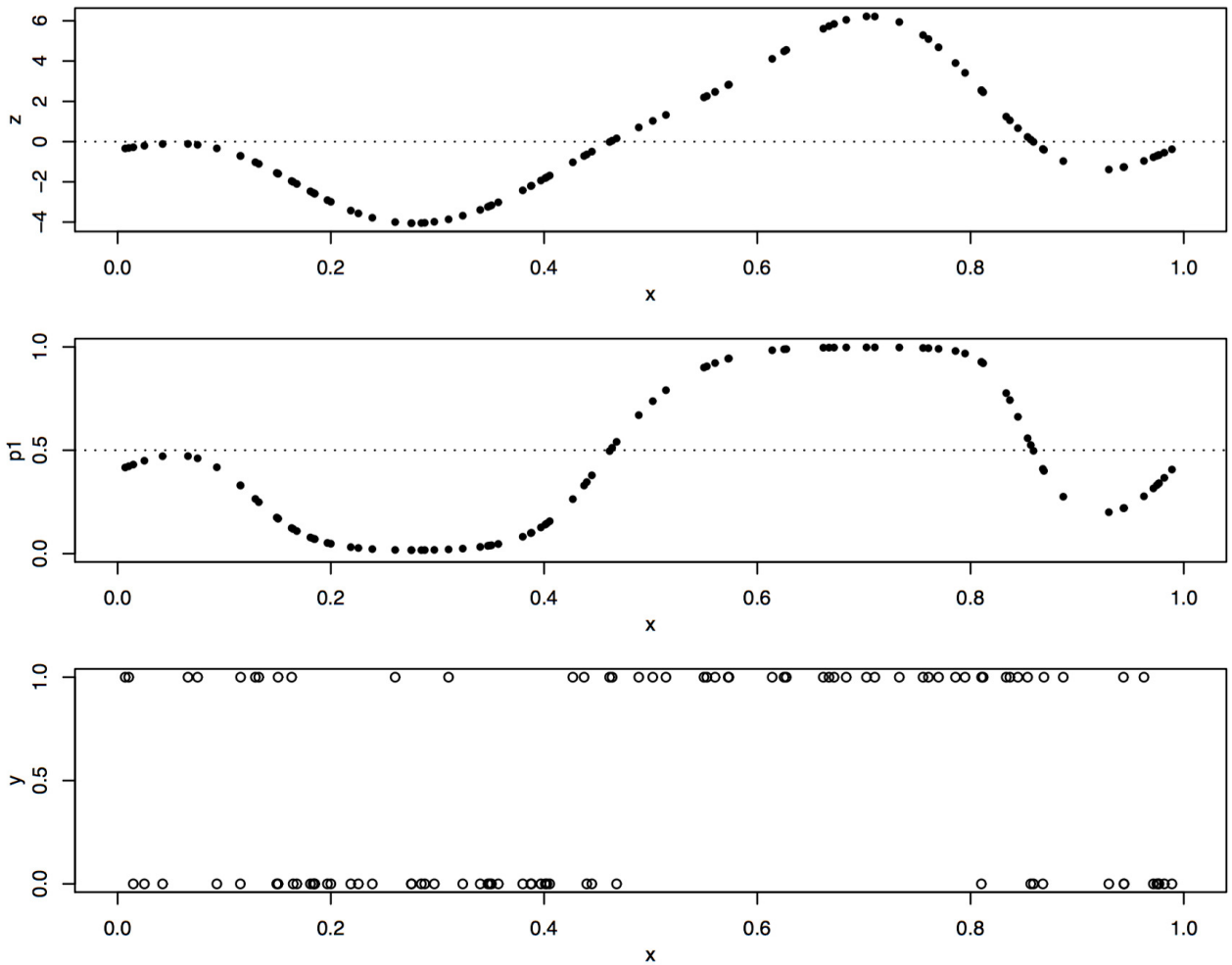
Implementing the model with Latent Variables*

Two methods are commonly used for handling the latent variables:

- **Approximate their posterior distribution by a Gaussian:**
  - The prior for $z_1, \ldots, z_N$ given θ is Gaussian.
  - The likelihood, $p(y_1, \ldots, y_N | z_1, \ldots, z_N)$, is not, but as $N \to \infty$ it will approach a Gaussian form. Maybe a Gaussian approximation of the posterior for $z_1, \ldots, z_N$ will be adequate for finite N.
- **Sample for the latent variables using Markov chain Monte Carlo:**
  - We use a Markov chain to sample $z_1, \ldots, z_N$, conditional on θ and $\mathbf{x}_1, \ldots, \mathbf{x}_N$.
  - We also sample for θ (unless it is fixed).
  - We then make a prediction for a test case at $\mathbf{x}^*$ by averaging $p(y^* | \mathbf{x}^*, z, \mathbf{x}, \theta)$ over the sample of values we obtain for z and θ.

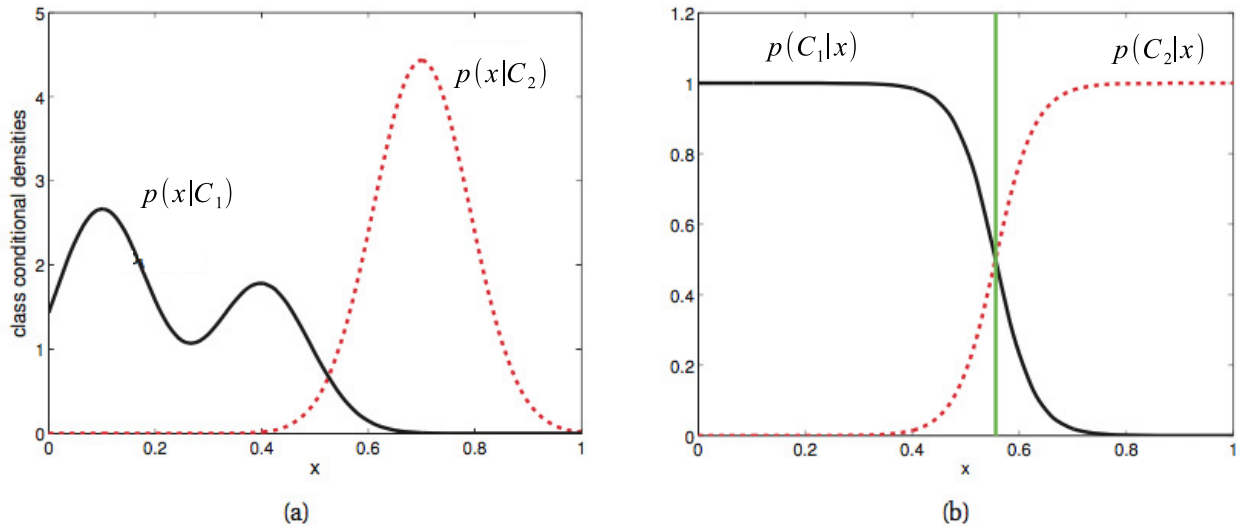
Illustration of a Gaussian Process Classification Model*

- Consider a Gaussian process classification model with one input, with covariance function $K(\mathbf{x}, \mathbf{x}') = 0.5^2 + 3^2 exp(-5^2(\mathbf{x} - \mathbf{x}')2)$.
- Below is a random sample from the latent Gaussian process at $\mathbf{x}$ values drawn uniformly from $(0, 1)$.
  - The resulting probabilities that $y = 1$, and values for y drawn according to these probabilities:



Which is Better: Generative or Discriminative?

- Even though logistic regression uses the same formula for the probability for C_1 given $\mathbf{x}$ as was derived for the earlier generative models, maximum likelihood logistic regression does not in general give the same values for β_0 and β as would be found with maximum likelihood estimation for the generative model.
- So which gives better results? It depends:
  - If the generative model accurately represents the distribution of inputs for each class, it should give better results than discriminative training
    - it effectively has more information to use when estimating parameters.
  - However, if the generative model is not a good match for the actual distributions, using it might produce very bad results, even when logistic regression would work well.
    - Logistic regression may be less sensitive to outliers than a Gaussian generative model.



The class-conditional densities $p(\mathbf{x}|C_k)$ with $k = \{0, 1\}$ (left) may be more complex than the class posteriors $p(C_k|\mathbf{x})$ (right).

Generative or Discriminative - Pros and cons

- **Easy to fit?**
  - As we have seen, it is usually very **easy to fit generative classifiers**.
    - For example, we show that we can fit a naive Bayes model and an LDA model by simple counting and averaging.
  - By contrast, logistic regression (discriminative) requires solving a convex optimization problem which is much slower.
- **Fit classes separately?**
  - In a generative classifier, we estimate the parameters of each class conditional density independently, so we do not have to re-train the model when we add more classes.
  - In contrast, in discriminative models, all the parameters interact, so the whole model must be retrained if we add a new class.
- **Handle missing features easily?**
  - Sometimes some of the inputs (components of $\mathbf{x}$) are not observed.
  - In a generative classifier, there is a simple method for dealing with this (e.g. marginalization).
  - However, in a discriminative classifier, there is no principled solution to this problem, since the model assumes that $\mathbf{x}$ is always available to be conditioned on.

Generative or Discriminative - Pros and cons

- **Can handle unlabeled training data?**
  - There is much interest in semi-supervised learning, which uses unlabeled data to help solve a supervised task.
    - easy to do using generative models (see e.g., (Lasserre et al. 2006; Liang et al. 2007)), but is much harder to do with discriminative models.
- **Can handle feature preprocessing?**
  - A big advantage of **discriminative methods** is that they allow us to **preprocess the input** in arbitrary ways, e.g., we can replace $\mathbf{x}$ with $\psi(\mathbf{x})$, which could be some basis function expansion.
  - It is often hard to define a generative model on such pre-processed data, since the new features are correlated in complex ways.
- **Well-calibrated probabilities?**
  - Some generative models, such as naive Bayes, make strong independence assumptions which are often not valid.
    - This can result in very extreme posterior class probabilities (very near 0 or 1).
  - Discriminative models, such as logistic regression, are usually better calibrated in terms of their probability estimates.

Summary of techniques used for classification (and regression)

Model	Classif/regr	Gen/Discr	Param/Non
Discriminant analysis	Classif	Gen	Param
Naive Bayes classifier	Classif	Gen	Param
Tree-augmented Naive Bayes classifier	Classif	Gen	Param
Linear regression	Regr	Discrim	Param
Logistic regression	Classif	Discrim	Param
Sparse linear/ logistic regression	Both	Discrim	Param
Mixture of experts	Both	Discrim	Param
Multilayer perceptron (MLP)/ Neural network	Both	Discrim	Param
Conditional random field (CRF)	Classif	Discrim	Param
K nearest neighbor classifier	Classif	Gen	Non
(Infinite) Mixture Discriminant analysis	Classif	Gen	Non
Classification and regression trees (CART)	Both	Discrim	Non
Boosted model	Both	Discrim	Non
Sparse kernelized lin/logreg (SKLR)	Both	Discrim	Non
Relevance vector machine (RVM)	Both	Discrim	Non
Support vector machine (SVM)	Both	Discrim	Non
Gaussian processes (GP)	Both	Discrim	Non
Smoothing splines	Both	Discrim	Non

Outline

Introduction

Evaluation & metrics

Generative Models for Classification

Discriminative Models for Classification

Decision Trees

Large margin classifiers

Ensemble Methods

Decision Trees

Decision Trees

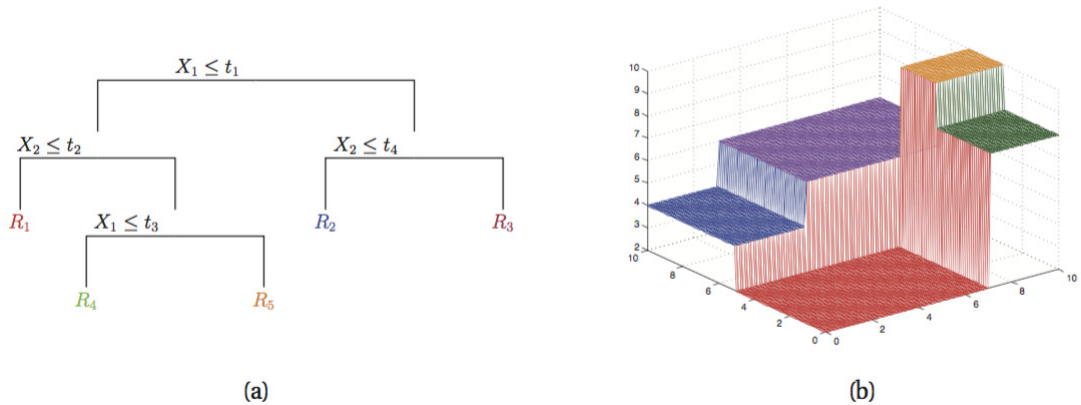
- Tree models where the target variable can take a discrete set of values are called classification trees;
  - leaves represent class labels
  - branches represent conjunctions of features that lead to those class labels
- Decision trees where the target variable can take continuous values are called regression trees.

References

- Classification and regression trees. L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, Chapman Hall, 1984.
- Pattern classification. R. Duda, P. Hart and D. Stork. Wiley, New York, 2000.
- Bagging Predictors. L. Breiman. Machine Learning 26 :123-140, 1996.
- Random Forests. L. Breiman. Machine Learning 45 :5-32, 2001.

Introduction to CART Models

- **Classification and regression trees** or **CART** models, also called *decision trees* are defined by recursively partitioning the input space, and defining a local model in each resulting region of input space. This can be represented by a tree, with one leaf per region, as we explain below.



A simple regression tree on two inputs

- Split the 2d space into regions, then a mean response can be associated with each of these regions resulting in a piecewise constant surface.
- Such a model can be written in the following form:

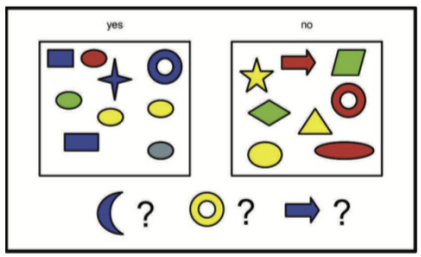
$$f(\mathbf{x}) = \mathbb{E}[y|\mathbf{x}] = \sum_{m=1}^{M} w_m \mathbb{1}(\mathbf{x} \in R_m) = \sum_{m=1}^{M} w_m \phi(\mathbf{x}; \mathbf{v}_m)$$

where R_m is the m-th region, w_m the mean response in this region and $\mathbf{v}_m$ encodes the choice of variable to split on, threshold value, on the path from the root to the m-th leaf.

- CART model is just an adaptive basis function model where the weights specify the response value in each region and the basis functions define the regions.

Classification Tree

- Generalization for classification by storing the distribution over class labels in each leaf, instead of the mean response.



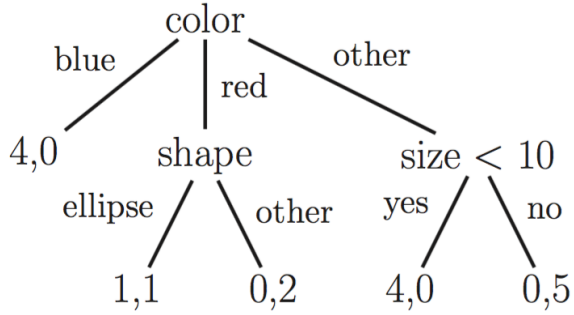
(a)

D features (attributes)

N cases

Color	Shape	Size (cm)	Label
Blue	Square	10	1
Red	Ellipse	2.4	1
Red	Ellipse	20.7	0

(b)



A simple regression tree on two inputs

- A leaf labeled as (N_1, N_0) means that there are N_1 positive examples that match this path, and N_0 negative examples.
- In this tree, most of the leaves are "pure", meaning they only have examples of one class or the other; the only exception is leaf representing red ellipses, which has a label distribution of (1, 1).
- We could distinguish positive from negative red ellipses by adding a further test based on size.
- However, it is not always desirable to construct trees that perfectly model the training data, due to overfitting.

Construction

Elementary tree for binary decision

- Binary $X_i \in \{0,1\}$, $Y_i \in \{0,1\}$
- Learning set (X_i, Y_i), $i = 1, \ldots, n$
- Contingency table

Y / X	0	1	Total
0	$n_{0\|0}$	$n_{0\|1}$	n_0
1	$n_{1\|0}$	$n_{1\|1}$	n_1
Total	$n_{:\|0}$	$n_{:\|1}$	n

- $\Pr(Y = k|X = \ell) = \pi_{k|\ell}$ with $\pi_{0|\ell} + \pi_{1|\ell} = 1$
- Maximum likelihood

$$\widehat{\pi}_{k|\ell} = \frac{n_{k|\ell}}{n_{0|\ell} + n_{1|\ell}}$$

- Decision with cost function 0/1

$$f(x) = \begin{cases} 1 & \text{si } n_{1|x} > n_{0|x} \\ 0 & \text{sinon} \end{cases}$$

Construction

- Many predictors (D = grande dimension)
- Quantity or category
- It is impossible to consider all $\mathbf{x} = (x_1, \ldots, x_D)$

$$\Longrightarrow \text{Classification and Decision Tree}$$

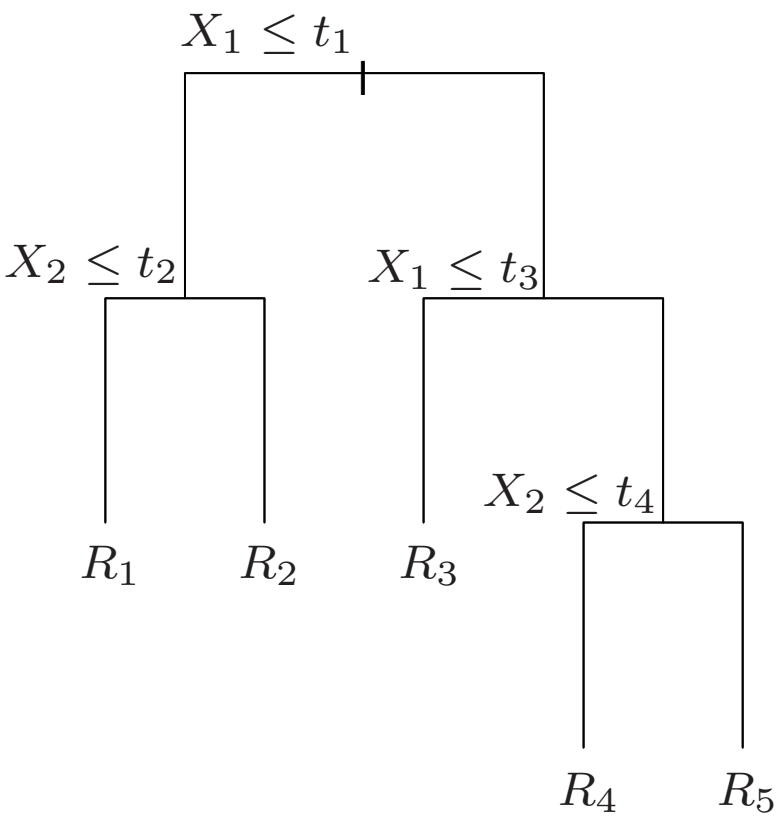
How to learn :

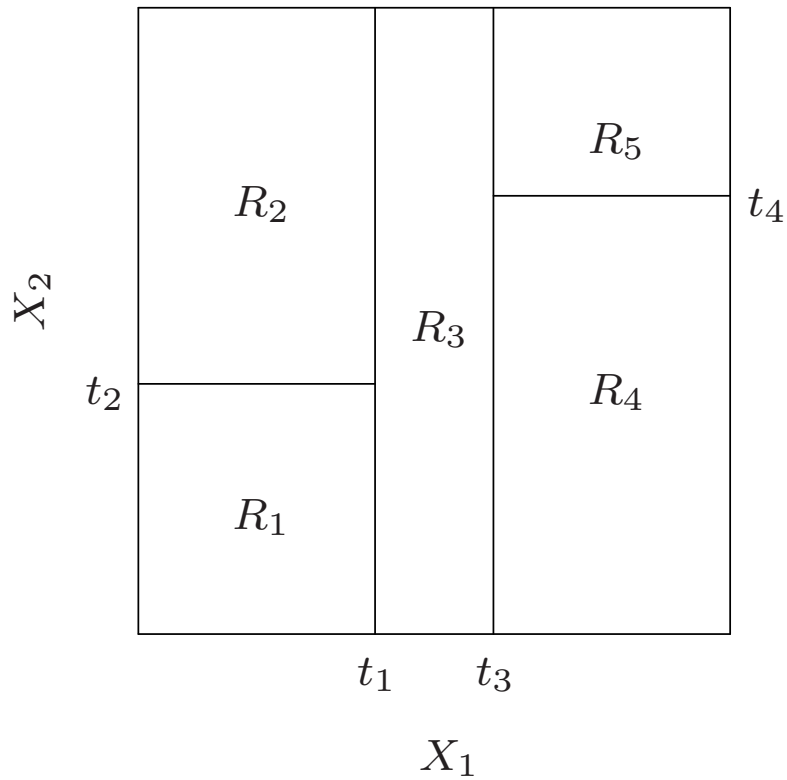
- good questions? (selectors)
- lgood answers? (decision)
- keeping only relevent questions? (pruning)

Construction principles

- (X_i) $i = 1, \ldots, n$: training set
- Classification tree= successive binary split of (X_i), starting with the whole training set
- union of children node areas
  = father node area
- each leaf node $\Longrightarrow$ classification / decision rule
- prediction = leaf associated with X

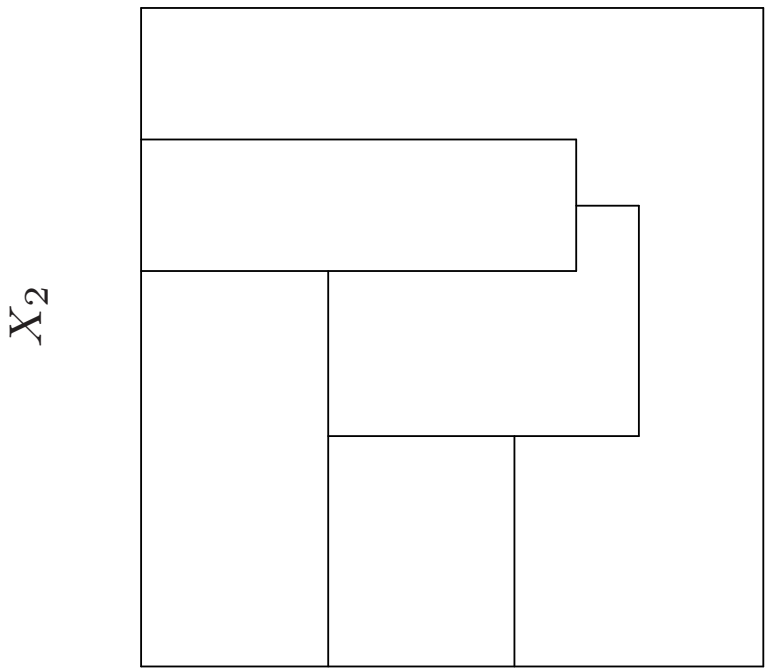
Construction principles



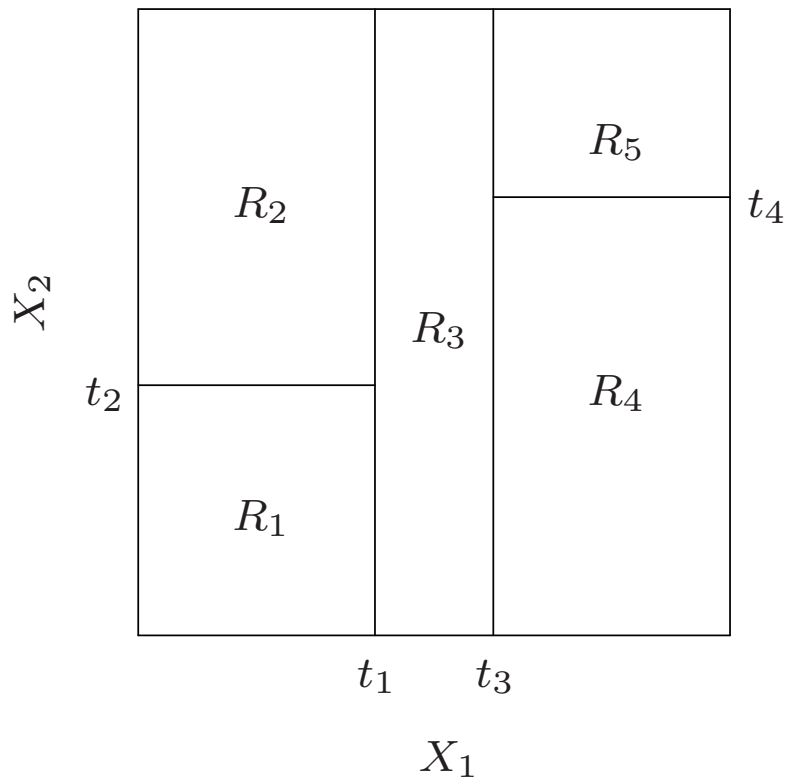


Construction principles

All splits are hierarchical!



$\neq$



Binary classes / others

- Vector: $X = (X_1, \ldots, X_p)$
- Quantitative or categorical data
- Each split at a tree node is based on only one variable
- If $X_j \in \{1, \ldots, M\}$, the question is "$X_j \in A$?", $A \subset \{1, \ldots, M\}$ (e.g. M binary questions)
- If $X_j \in \mathbb{R}$, finite training set, there exists a finite number of questions $X_j \leq c$

Algorithm

Recursive construction of a decision tree
Procedure *Build-tree (node m)*
Begin

if *Every points of m belong to the same class*
then Make a leaf denoted by the class name

else
Chose the best predicate to make a node
Test predicate in order to split m into two child nodes m_L and m_R
`Build-tree` (m_L)
`Build-tree` (m_R)

end
end

Algorithm

Three main steps:

- Node split criterion to use
- Leaf node decision
- Node spliting rule / or not

Node impurity

- R_m = set of N_m training data at the node m,

$$p_{m,k} = \frac{1}{N_m} \sum_{x_i \in R_m} I(y_i = k)$$

  the frequency of each class k at the node m

- $d(m) = \arg\max_k p_{m,k}$ "majority voting" at the node m

How to mesure the spliting rule quality at a node m?

Node impurity

How to mesure the spliting rule quality at a node m?

$$p_{m,k} = \frac{1}{N_m} \sum_{x_i \in R_m} I(y_i = k)$$

Impurity function:
ϕ defined on $(p_1, \ldots, p_K)$ with $p_k \geq 0$ and $\sum_{k=1}^{K} p_k = 1$ such as

- ϕ has unique maximum at $(\frac{1}{K}, \ldots, \frac{1}{K})$
- ϕ is minimal at points $(1, 0, \ldots, 0)$, $(0, 1, \ldots, 0)$, $\ldots$
- $\phi(p_1, \ldots, p_K) = \phi(p_{\sigma(1)}, \ldots, p_{\sigma(K)})$ for any perturbation σ

Impurity metrics

$$p_{m,k} = \frac{1}{N_m} \sum_{x_i \in R_m} I(y_i = k)$$

- Standard impurity metrics
  - Classification error:

$$\frac{1}{N_m} \sum_{i \in R_m} I(y_i \neq d(m)) = 1 - p_{m,d(m)}$$

  - Gini index:

$$\sum_{k \neq k'} p_{m,k} p_{m,k'} = \sum_{k=1}^{K} p_{m,k}(1 - p_{m,k})$$

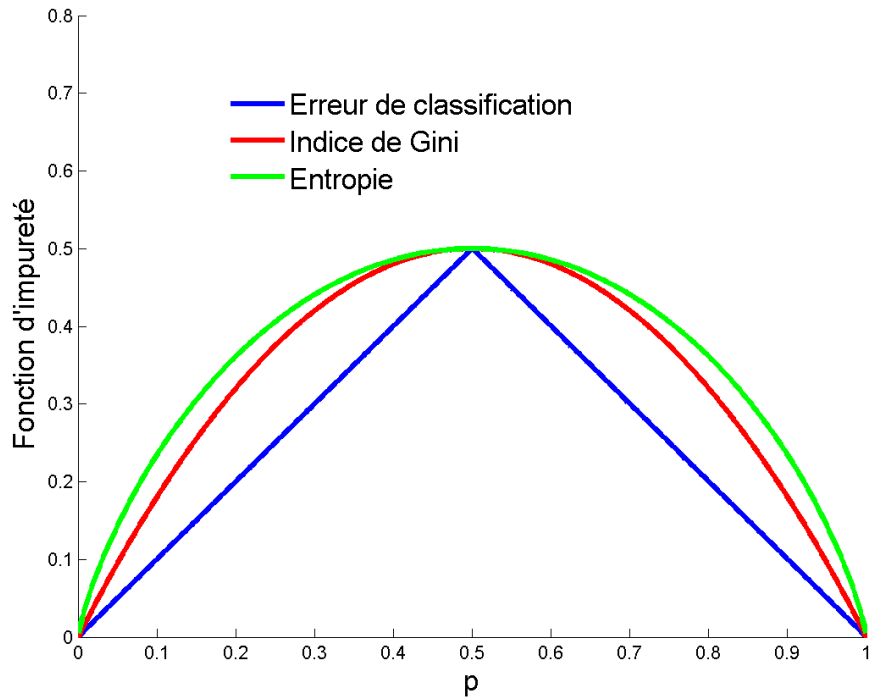
  - Cross-entropy:

$$-\sum_{k=1}^{K} p_{m,k} \log p_{m,k}$$

Impurity metrics

Simple case: $K = 2$

- If $K = 2$ and p the probability of the second class
  - Classification error: $1 - \max(p, 1 - p)$
  - Gini index: $2p(1 - p)$
  - Cross-entropy: $-p \log p - (1 - p) \log(1 - p)$



Impurity metrics and selector choice

- m_L and m_R children node of the node m
- $\phi(p_{m_L})$ and $\phi(p_{m_R})$ impurity measures of the nodes m_L and m_R
- $\phi(p_m)$ = ifather node impurity,
- **Spliting quality** s at the node m is measured with:

$$\Delta\phi(s, m) = \phi(p_m) - (\pi_L\phi(p_{m_L}) + \pi_R\phi(p_{m_R}))$$

  where π_L and π_R are the proportions of data distributed between the left and right nodes
- **Split** s = variable choice then best split for this variable
- Exhaustively computable for moderate dataset

Pruning: stoping criterion

- Intuitively: stop at node m when

$$\arg\max_s \Delta\phi(s, m) < \varepsilon$$

where ε a given threshold
$\Rightarrow$ Not efficient: many successive not decisive splitting may cause an important gain gap

- **Best strategy**
  1. Build a very deep tree T_0 and stop when a minimal size is reached at a node (e.g. 5 elements)
  2. Prune branches based on a complexity criterion

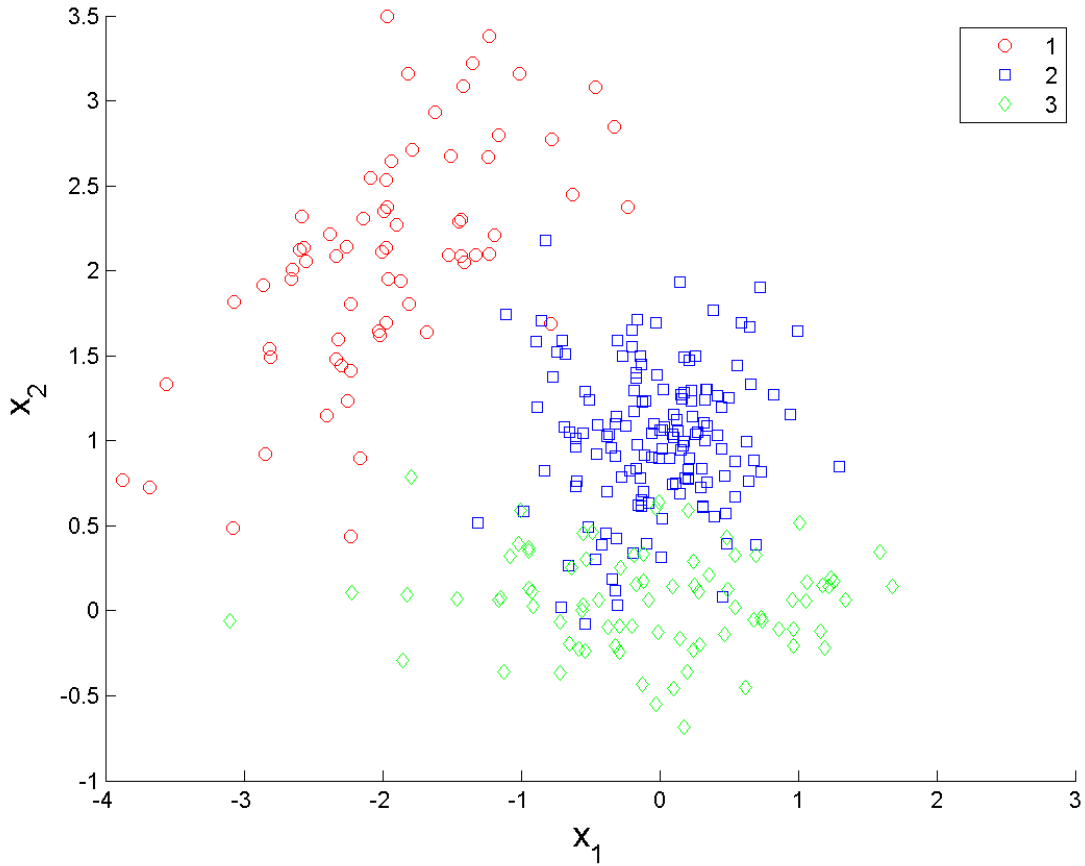
Pruning: regularisation

- We define a complexity-cost criterion with:

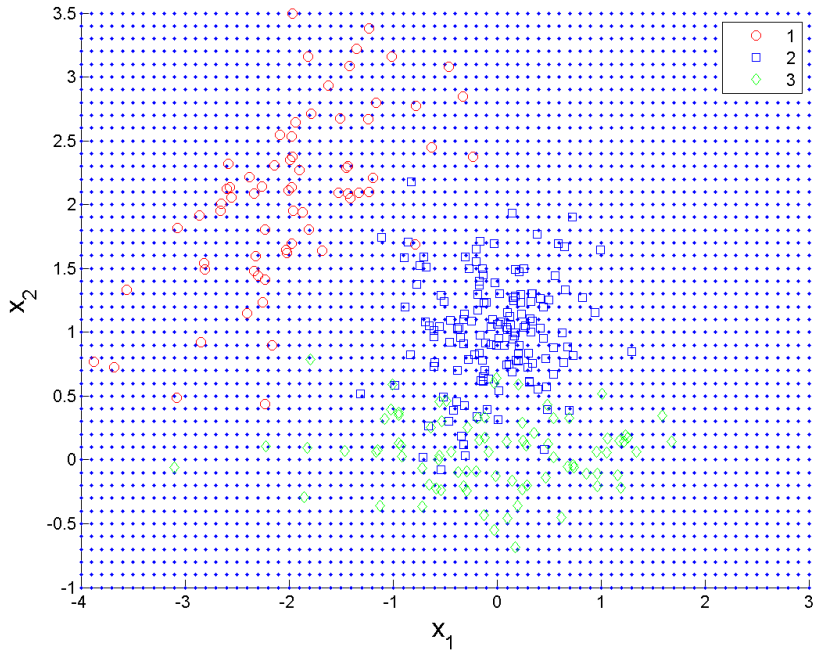
$$C_\alpha(T) = \sum_{m=1}^{|T|} N_m \phi(p_m) + \alpha |T|$$

  where $|T|$ is the number of leafs of T and $m = 1, \ldots, |T|$ corresponds to leafs
- The α parameter govern the tree complexity
  - $\alpha = 0$ corresponds to the tree T_0
  - α large: induces low depth tree
- For fixed α, find the sub-tree $T_\alpha \subseteq T_0$ minimizing $C_\alpha(T)$
- We can show that there is one unique sub-tree minimizing $C_\alpha(T)$
- α can be fixed by cross-validation

Example: synthetic data

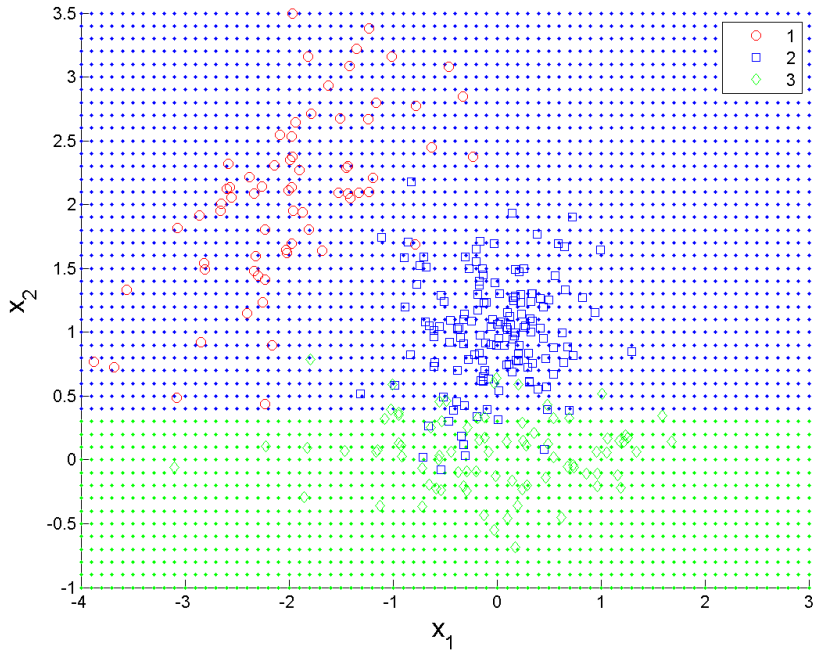


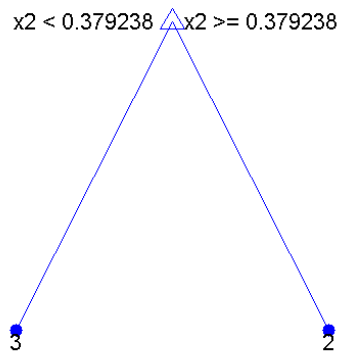
Example: synthetic data



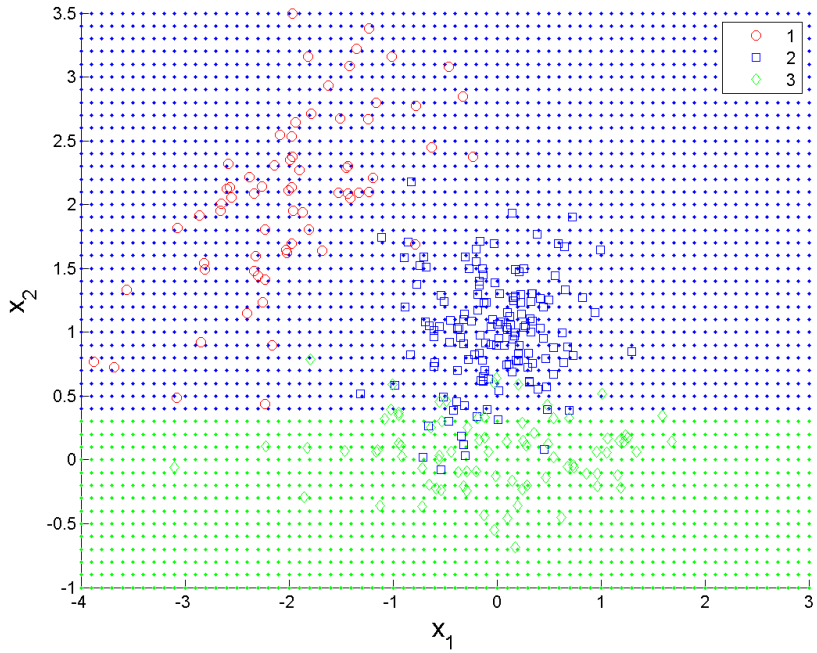
2

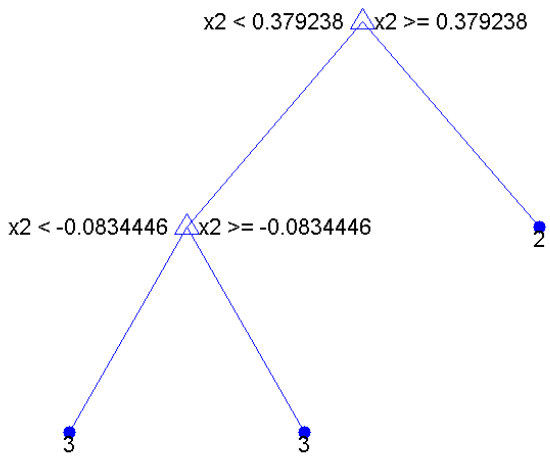
Example: synthetic data



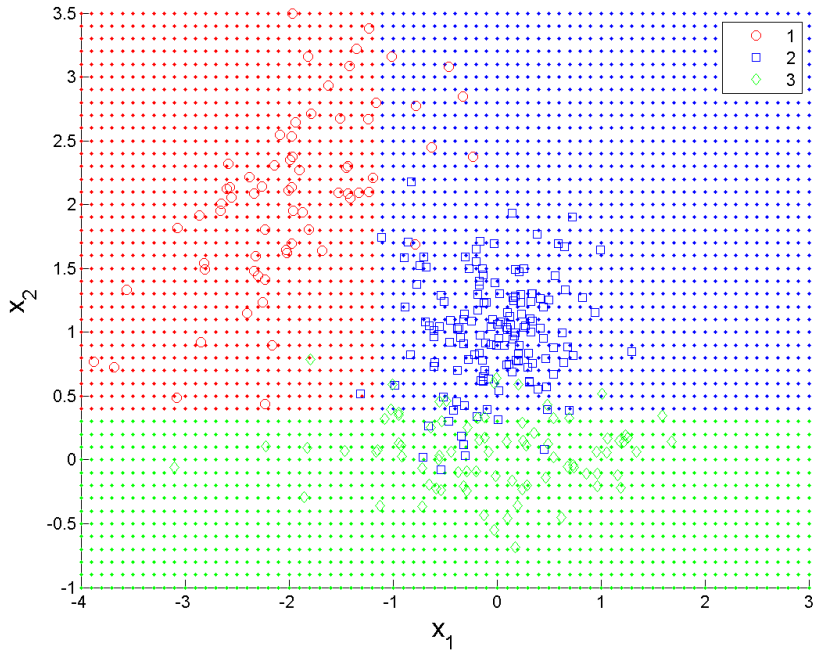


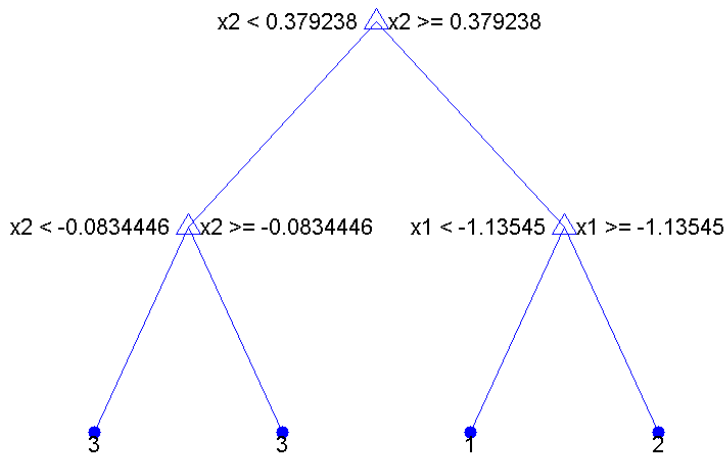
Example: synthetic data



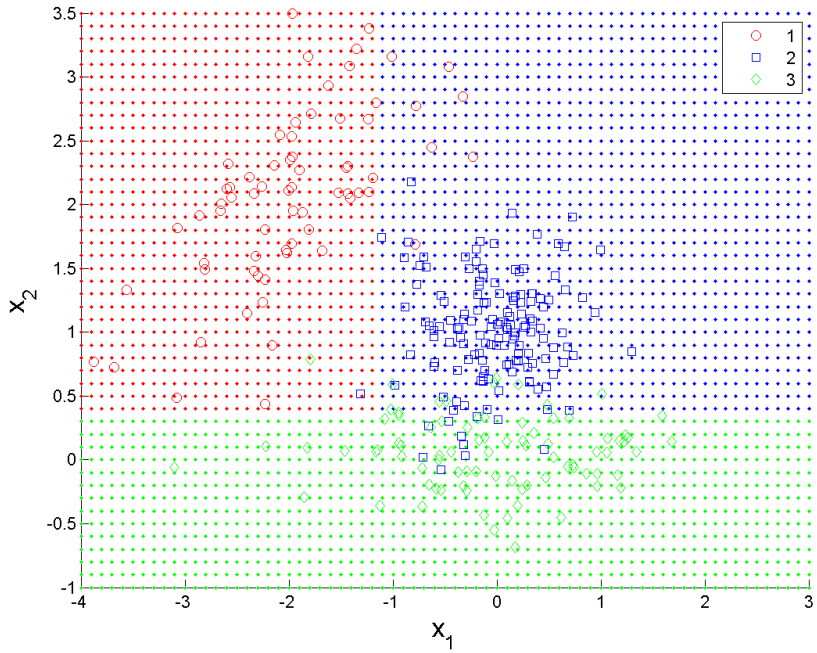


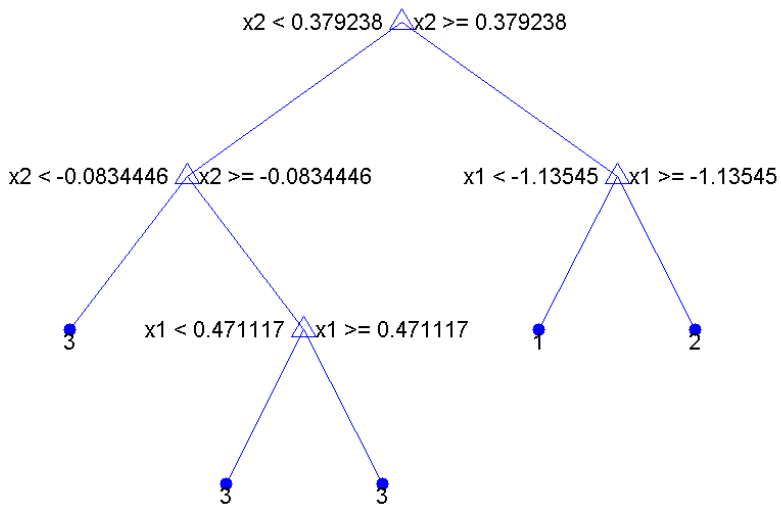
Example: synthetic data



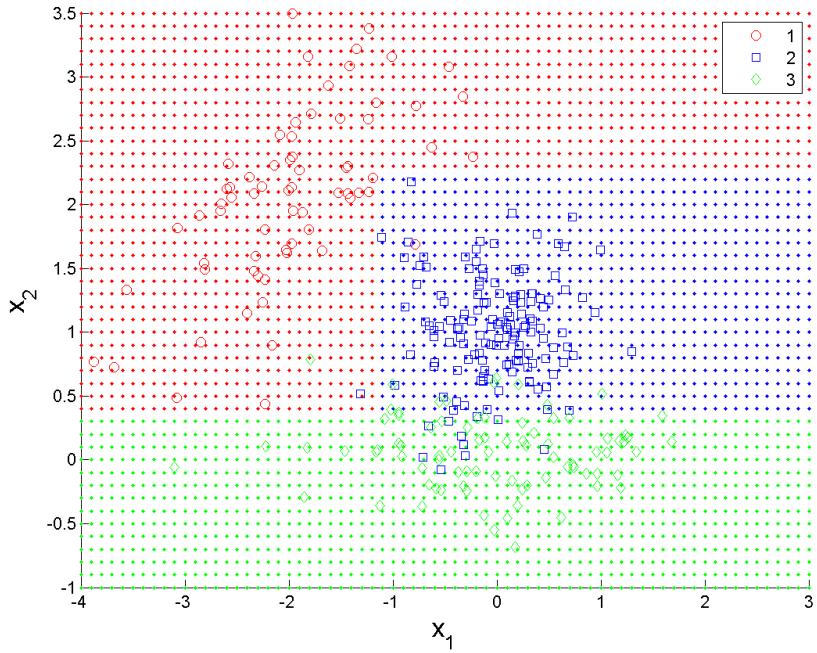


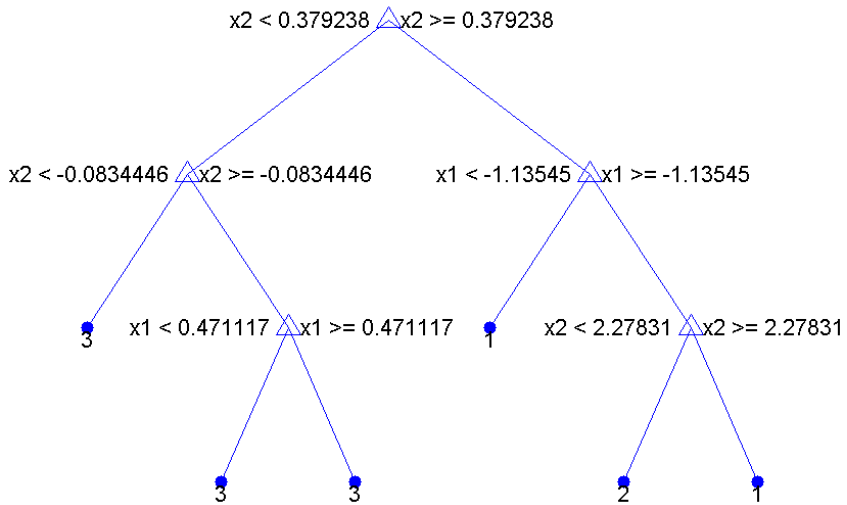
Example: synthetic data



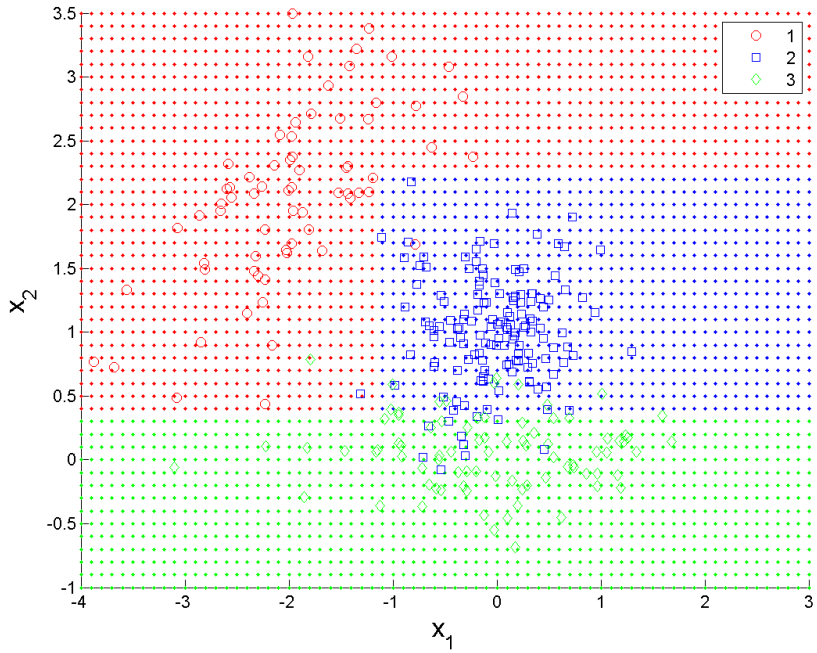


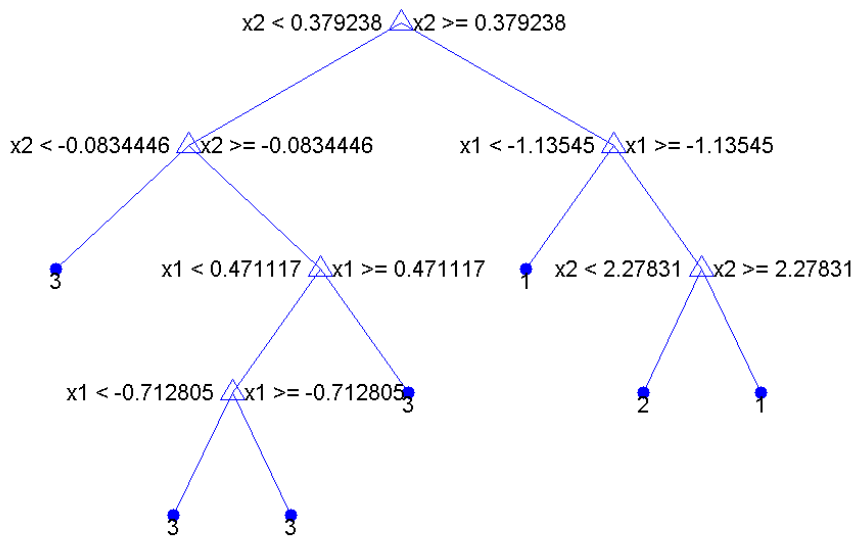
Example: synthetic data



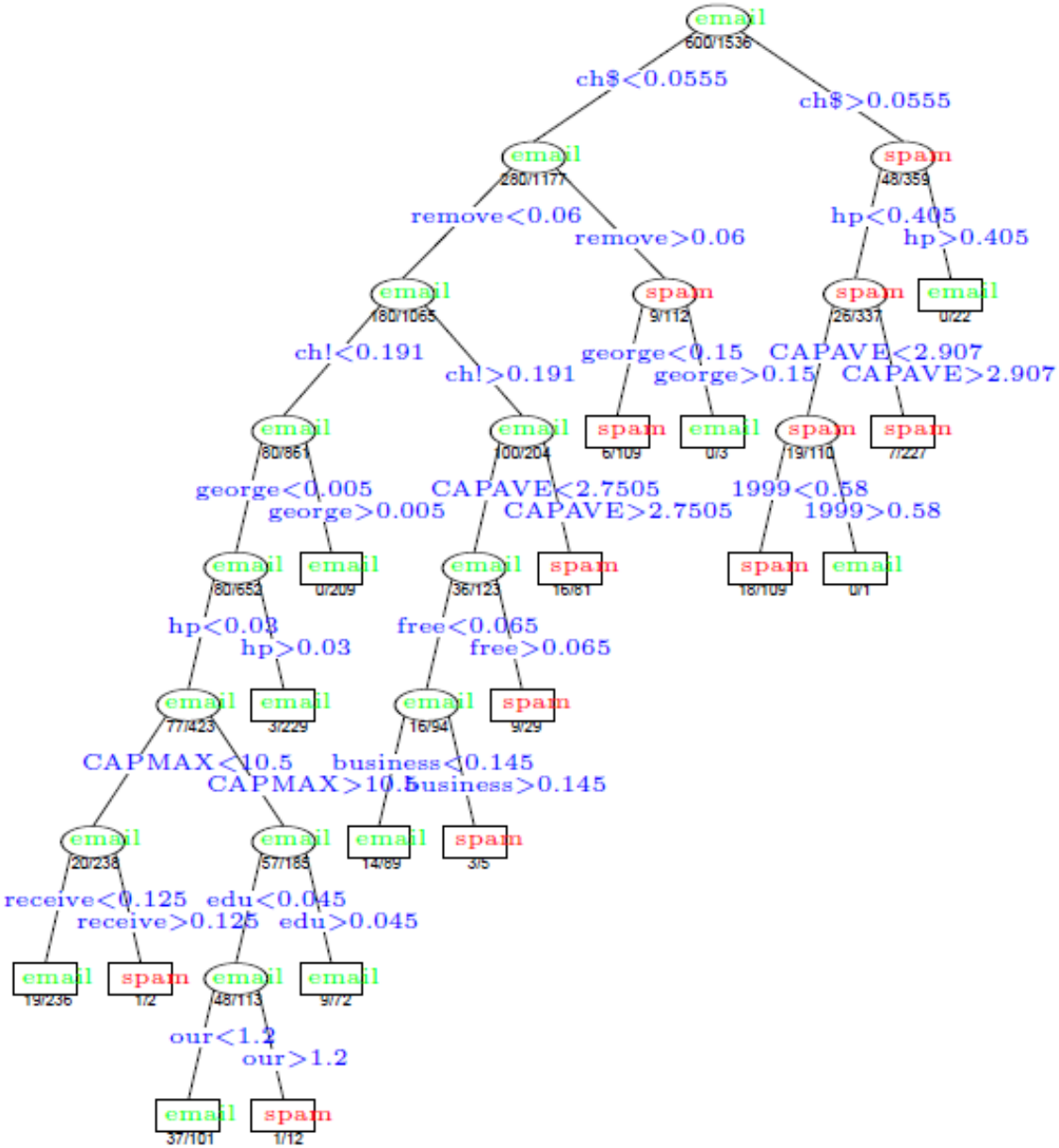


Example: synthetic data





Example: Spam vs Ham



Classification Tree Example

rec	Age	Income	Student	Credit _rating	Buys_computer
r1	<=30	Hight	No	Fair	No
r2	<=30	Hight	No	Excellent	No
r3	31...40	Hight	No	Fair	Yes
r4	>40	Medium	No	Fair	Yes
r5	>40	Low	Yes	Fair	Yes
r6	>40	Low	Yes	Excellent	No
r7	31...40	Low	Yes	Excellent	Yes
r8	<=30	Medium	No	Fair	No
r9	,=30	Low	Yes	Fair	Yes
r10	>30	Medium	Yes	Fair	Yes
r11	<=30	Medium	Yes	Excellent	Yes
r12	31...40	Medium	No	Excellent	Yes
r13	31...40	High	Yes	Fair	Yes
r14	>40	Medium	No	Excellent	No

Example data

Learning a Tree

- **Growing a Tree**
  - Finding the optimal partitioning of the data is NP-complete $\Rightarrow$ learning algorithms are based on heuristics such as the greedy algorithm where locally-optimal decisions are made at each node.
  - Such algorithms cannot guarantee to return the globally-optimal decision tree.
  - The split function chooses the best feature, and the best value for that feature by minimizing a cost function,
- **Pruning a Tree**
  - To prevent overfitting, we can stop growing the tree if the decrease in the error is not sufficient to justify the extra complexity of adding an extra subtree.
- However, the standard approach is therefore to grow a "full" tree, and then to perform pruning. This can be done using a scheme that prunes the branches giving the least increase in the error. See (Breiman et al. 1984) for details.

Pros and Cons of Trees

CART models are popular for several reasons:

- easy to interpret (with these logical rules),
- easily to handle mixed discrete and continuous inputs,
- perform automatic **variable selection** and scale well to large data sets.

However, CART models have some disadvantages:

- do not predict very accurately compared to other kinds of model. This is in part due to the greedy nature of the tree construction algorithm.
- could be unstable: small changes to the input data can have large effects on the structure of the tree, due to the hierarchical nature of the tree-growing process, causing errors at the top to affect the rest of the tree.
  - trees are high variance estimators.

Outline

Introduction

Evaluation & metrics

Generative Models for Classification

Discriminative Models for Classification

Decision Trees

Large margin classifiers
- Support Vector Machines

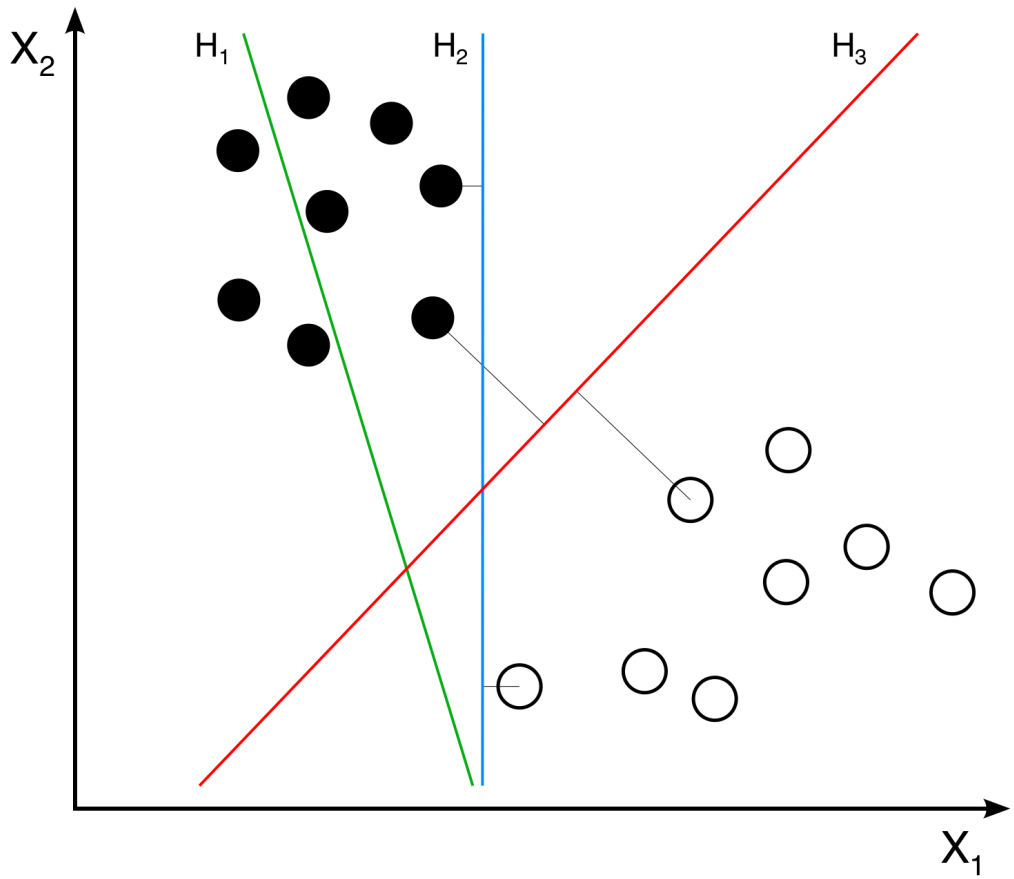
Ensemble Methods

The Large Margin Classfier

- Some classification methods produce *only* a predicted class for a test case, with no probability distribution for that class.
- *Large margin* classifiers are in this category. They are the basis for the popular *Support Vector Machine* (SVM) classifiers.
- In their simplest form, large margin classifiers **apply only to perfectly separable binary classification problems**, in which there is a **hyperplane that separates** the training cases in one class from the training cases in the other class.
- There will usually be many hyperplanes separating the classes. The idea is to pick the separating hyperplane that has the **largest** *margin*,
  - where the margin is the minimum distance of a training case from the line

An Illustration with Two Inputs

Here is a large margin classifier in two dimensions (where a hyperplane is a straight line). There are eight training cases in Class -1 (white) on the left, and eight in Class +1 (black) on the right. The line H1 does not separate the classes. $H2$ does, but only with a small margin. $H3$ separates them with the maximum margin.



Finding the Separating Hyperplane with Largest Margin

- We can define a hyperplane by the equation $w^T \mathbf{x} + b = 0$.
  - We can use w and b to classify test cases to the class sign$(w^T \mathbf{x} + b)$.
  - We will use -1 and +1 as class labels.
- Note that negating both w and b will swap which side of the hyperplane has which class, and multiplying both w and b by any positive constant will not change classifications.
- When finding w and b from the training cases, we will impose the constraint that all training cases are classified correctly – that is,

$$y_i(w^T \mathbf{x} + b) > 0, \text{ for } i = 1, \ldots, N$$

- But we want to also maximize the margin, which is

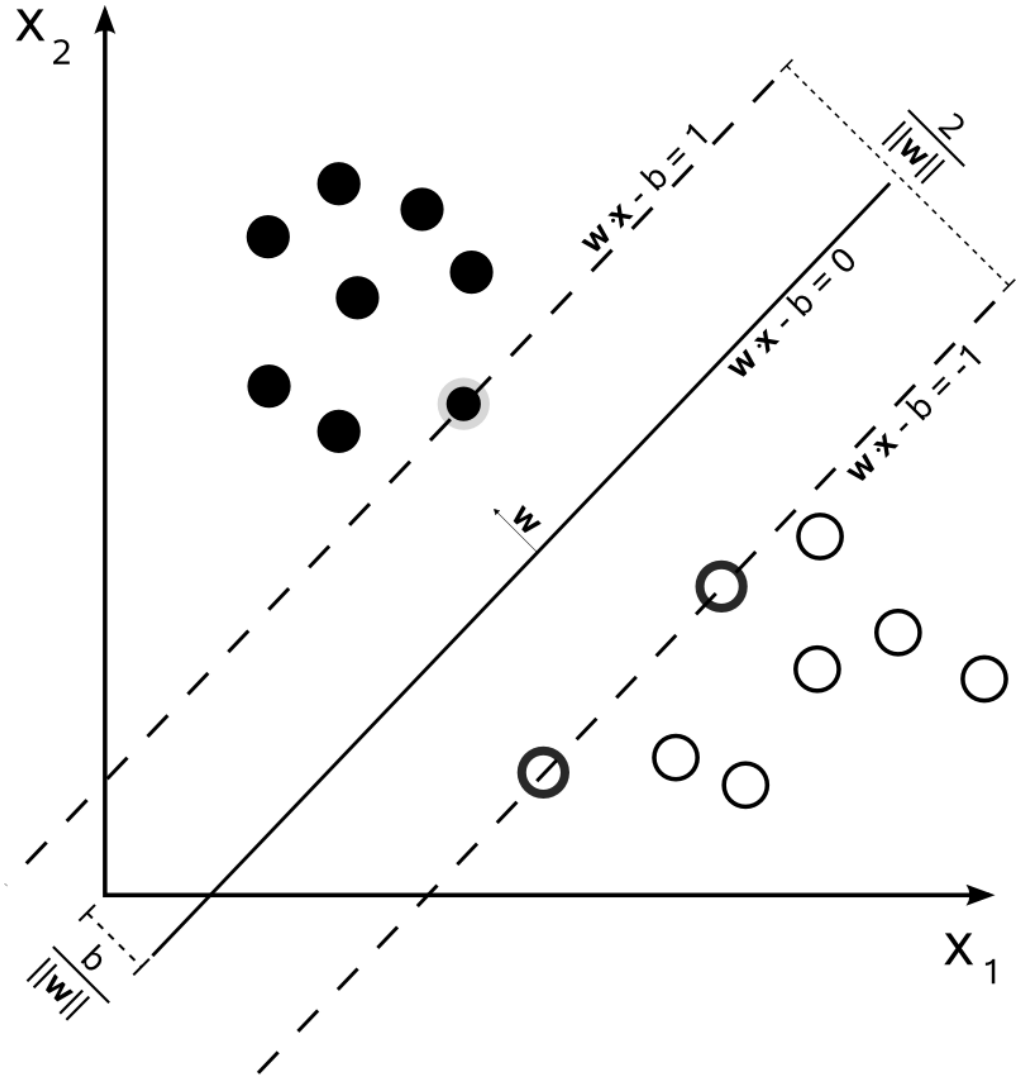
$$\min_{i=1,\ldots,N} y_i(w^T \mathbf{x} + b)/||w||$$

- This is equivalent to the following optimization problem:

$$\text{minimize} ||w||^2, \text{ subject to } y_i(w^T \mathbf{x} + b) \geq 1, \text{ for } i = 1, \ldots, N$$

- The minimizaton will shrink w to where at least one inequality above is an equality, at which point the margin will be $1/||w||$, so maximizing the margin is the same as minimizing $||w||^2$.

Illustration



Maximum-margin hyperplane and margins for a Large Margin Classifier trained with samples from two classes. Samples on the margin are called the support vectors.

Characteristics of the Maximum Margin Separating Hyperplane

- The previous slide characterizes the maximum margin hyperplane as minimizing a quadratic function, subject to linear inequality constraints.
  - This is a convex optimization problem. It has a unique solution, which can be found reasonably efficiently by standard methods (or more efficiently using specialized methods).
  - The solution is **locally sensitive to a subset of the training cases**, called the *support vectors*
    - typically, but not always, less (often much less) than the full set of training cases.
- Of course, all training cases have to be checked at before the support vectors can be identified.
  - But when there are only a few support vectors, the computations go faster.

Outline

Introduction

Evaluation & metrics

Generative Models for Classification

Discriminative Models for Classification

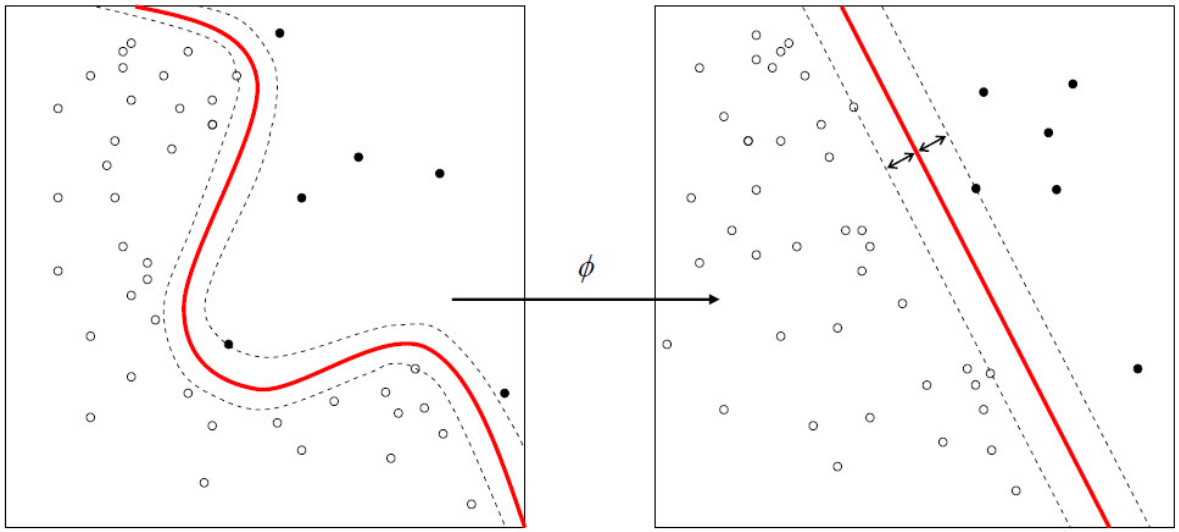
Decision Trees

Large margin classifiers
- Support Vector Machines

Ensemble Methods

SVM

- Previous algorithm suppose that the classes can be separated linearly.
- **Idea:** Rather than find a large margin classifier based on the original input vector, $\mathbf{x}$, we can use a vector of basis function values, $\phi(\mathbf{x}) = [\phi_1(\mathbf{x}), \phi_2(\mathbf{x}) \cdots \phi_m(\mathbf{x})]^T$.
- The classes may be separable by a hyperplane in this space even if they aren't in the original space.



- Classification (and regression) methods based on this "kernel trick" are known as *Support Vector Machines* (abbreviated to "SVM").
- Imagine the example of two classes roughly around two 2D dishes with different radius
  - A great transformation for discrimination would be $\phi_1(\mathbf{x}) = \sqrt{x_1^2 + x_2^2}$

More Elaborations on Support Vector Machines

- Which kernel function is best is usually not clear. Cross validation can be used to choose one.
- Finding a separating hyperplane (even if always possible in an infinite dimensional space) may not be a good idea, when class labels are actually "noisy".
  - Introducing "slack variables" allows for some mis-classified points.
- Classification problems with more than two classes can be handled in various ways, e.g., combining results from pairwise binary classifiers.
- Regression problems can be handled by using an appropriate "loss" function [see work of Vapnik et al. 1996]

Support Vector Machines vs. Gaussian Process Models*

SVM and GP models have a strong common element – the positive semi-definite kernel/covariance function. How do they compare otherwise?

Advantages of support vector machines:

- The number of support vectors is often much less than the total size of the training set, reducing computation time for training and prediction.
- Binary classification can be done directly, with a relatively fast optimization procedure, whereas Gaussian process classification requires handling a distribution over "latent variables".

Advantages of Gaussian process models:

- The covariance function has a probabilistic interpretation – one can sample from the prior over functions that it defines – which can guide the choice of a suitable covariance function.
- Finding good parameters of the covariance function can be done by maximum likelihood (or by Bayesian methods), without the need for cross validation.
- Classification problems with more than two classes can be handled naturally.
- Regression can be done using a conventional Gaussian model for residuals.

Outline

Introduction

Evaluation & metrics

Generative Models for Classification

Discriminative Models for Classification

Decision Trees

Large margin classifiers

Ensemble Methods

Introduction to ensemble methods

- *Ensemble learning* is the process by which multiple models, such as classifiers or regressors, are generated and combined to solve a particular computational intelligence problem.
- Ensemble learning is primarily used to improve the (classification, prediction, function approximation, etc.) performance of a model, or reduce the likelihood of an unfortunate selection of a poor one.
- **Simple Example**: Suppose we have $T = 5$ completely independent classifiers for a binary classification problems
  - the accuracy of each classifier being $a = 70\%$
  - thus by using the Majority of vote (i.e., choosing the class that receives the largest total vote), the probability of number of vote,k, is a binomial (sum of Bernoulli random variables)
  - thus the probability of choosing the correct class is therefore:

$$P(k > \lfloor T/2 + 1 \rfloor) \sum_{k=lfloorT/2+1\rfloor}^{T} \begin{bmatrix} T \\ k \end{bmatrix} a^k (1-k)^{T-k}$$

  In this case a final accuracy of 83.7% (for $T = 101 \Rightarrow$ 99.9%)
- **Problem:** Assumption of **independence** of the learning algorithm !!

Commonly used Ensemble technique

There are 3 main ensemble learning techniques used:

1. Bagging,
2. Boosting,
3. Stacking.

1. Bagging

- *Bootstrap aggregating*, is one of the earliest, most intuitive and perhaps the simplest ensemble based algorithms, with a surprisingly good performance [Breiman 1996]
- Diversity of classifiers in bagging is obtained by using bootstrapped replicas of the training data. That is, different training data subsets are randomly drawn **with replacement** from the entire training dataset.
- Each training data subset is used to train a different machine learning algorithms, and then combine using for example
  - average in regression
  - majority vote in classification
- Need to define the percentage to create the bootstrapped training set.
- *Remarks:*
  - A random selection of a subset of input variables can be used
    - instead of using all p covariates randomly select a subset of them
  - The *random forest* is a decision tree that uses this bagging technique by trying moreover to decorrelate the learners by learning on trees based on a **random subset of both data and input covariates**.

- The algorithm works by applying the (weak) learner sequentially to weighted versions of the data, where more weight is given to examples that were misclassified by earlier rounds.
- **Example in classification:** the first classifier C_1 is trained with a random subset of the available training data, $\mathcal{D}_1$.
  - The training data subset for the second classifier C_2 is chosen such that 50% in average of the samples from this set are missclassified by C_1.
  - The third classifier C_3 is trained with data from $\mathcal{D}$ on which C_1 and C_2 disagree.
  - The three classifiers are combined through a three-way majority vote.
- **AdaBoost** (Adaptive Boosting) extends boosting to multi-class and regression problems [Freund, 2001]

Combination rules

For regression problems:

- average
- median
- weighted sum rules

For classification problems:

- Majority voting: decide the class that receives the majority of votes: $\max_{j=1,\ldots,C} \sum_{t=1}^{T} d_{t,j}$
  - when we can provide the certainty of each classifier
- Weighted majority voting: $\max_{j=1,\ldots,C} \sum_{t=1}^{T} w_t d_{t,j}$
  - it can be shown that the optimal weights is $w_t \propto \frac{q_t}{1-q_t}$ if the T classifiers are class-conditionally independent with accuracies $q_1, \ldots, q_T$

3. Stacking

- More advanced combination rule.
- The last ensembling method, *Stacking* can be seen as a way to combine the decisions of an ensemble of learners in a complex way by considering that the output value of the learners are going to be used as inputs of another learner algorithm, called *metalearner*.