

---

# Computer Vision – Lecture 8

## Pattern recognition concepts

UCT2 – Information Technologies  
MAP-I Doctoral Programme

***Pedro Quelhas***

30 November 2010

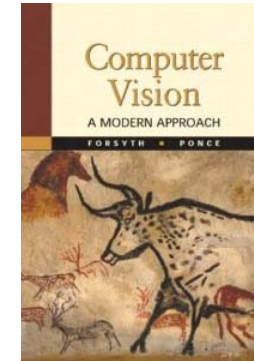
---

# References

---

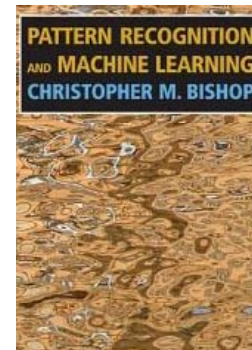
- **Computer vision**

- David A. Forsyth and Jean Ponce, “Computer Vision: A Modern Approach” (chapters 22,23,24)



- **Machine learning/ Patter recognition**

- Christopher M. Bishop , “Pattern Recognition and Machine Learning”



- **Tutorials and lectures from previous years:**

- [http://www.dcc.fc.up.pt/~mcoimbra/lectures/mapi\\_0809\\_materials.html](http://www.dcc.fc.up.pt/~mcoimbra/lectures/mapi_0809_materials.html) (lectures 7b, 8a and 8b)
- <http://www.autonlab.org/tutorials/>

---

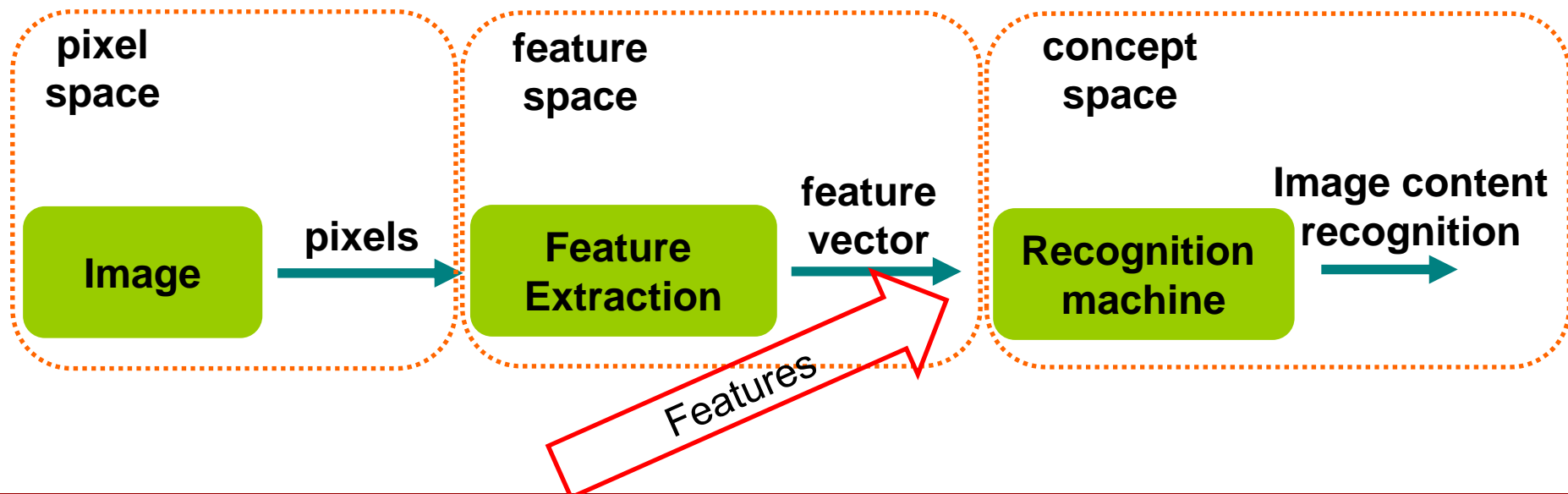
# Outline

---

- **Features and feature extraction (revision)**
- **Statistical pattern recognition**
  - Bayesian Decision Theory
    - MAP
    - MLE Gaussian estimation
    - Plug-in classifier
  - Dimensionality
  - Feature space selection
  - Non-Bayesian classifiers
    - Distance-based classifiers
      - KNN
    - Decision boundary-based classifiers
      - Decision trees
      - ANN
      - SVMs
  - Unsupervised learning and clustering

# Computer vision system

- **Image recognition system:**
  - **Feature extraction:** captures meaningful information from the image (for the specific task at hand), reducing dimensionality.
  - **Pattern recognition:** Does the actual job of classifying or describing observations, relying on the extracted features.
- **System diagram:**



---

# Features and feature extraction (revision)

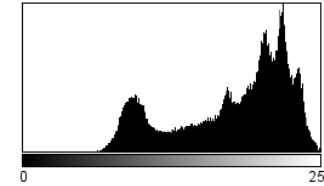
---

- Observers capture the meaning of an image discarding all unnecessary information. By choosing adequate features we do the same!
- By selecting some specific feature we introduce prior knowledge!
- Different image content is described by different features:
  - Shape, colour, texture...
- There are a wide range of different feature types
  - Low/middle/high level
  - Global/local
- The choice of feature selection is based on both prior knowledge we may have and the availability of certain features!
- Feature can be concatenated for decision based on joint information.

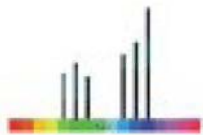
# Features and feature extraction (revision)

- **Classic features:**

- Global colour and edge histograms
- Texture through co-occurrence matrices and fractal analysis



- **MPEG-7 features:**

- |                    |  |
|--------------------|--|
| Colour + structure | – Dominant colour – clusters colours into a small number of colours in the image (salient colours) |
|                    | – Scalable colour – HSI histogram (H with 16 levels, S with 4 levels and I with 4 levels)          |
|                    | – Colour Layout – divides image into block and obtains average colours (sketch like feature)       |
|                    | – Colour structure – histogram of local colour structures  |
- 
- |         |   |
|---------|---|
| texture | – Homogeneous structure – filter response that indicates structure at different scales/orientations |
|         | – Local Edge histogram - Image divided into 4x4 sub-regions, 5 bin edge histogram for each region   |
- |       |   |
|-------|---|
| shape | – Freeman Chain Code – represents the border of an object by a code of relative steps.  |
|       | – Region-Based Shape – uses an ART basis functions response to describe shape along various angular and radial directions.  |
|       | – Contour-Based Descriptor - Finds curvature zero crossing points of the shape's contour (key points). The position of key points are expressed relative to the length of the contour curve |

---

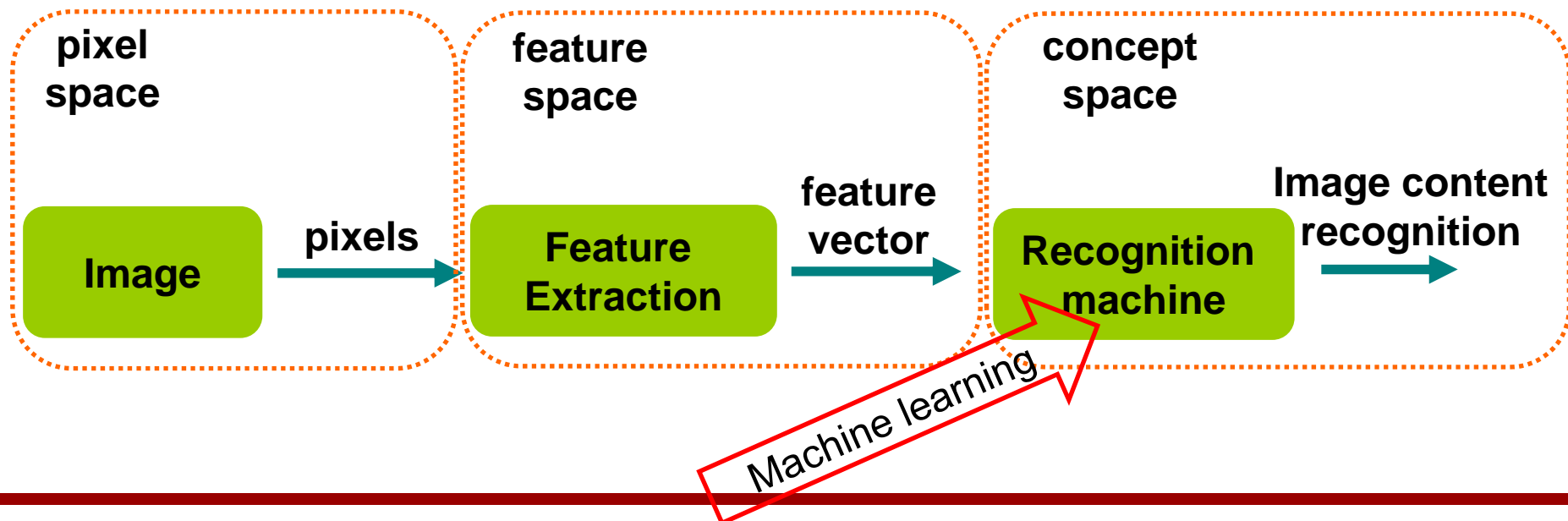
# Features and feature extraction (revision)

---

- **Image subdivision**
  - Global feature rarely have the descriptive power to capture all information in an image
  - This leaves global features usable only for some limited image recognition tasks and motivates localized image analysis
- **An image often requires a part based analysis:**
  - Context is global, but object are defined locally.
  - Most image content is described at a local level.
  - By dividing an image into parts we simplify recognition.
  - Separating objects from context makes recognition more robust
- **There are several way an image can be subdivided for analysis**
  - Object segmentation (when possible, not na easy taks in “normal images”!)
  - Grid subdivision (easy but leads to sampling problems)
  - Exhaustive search (slow)

# Computer vision system

- **Image recognition system:**
  - **Feature extraction:** captures meaningful information from the image (for the specific task at hand), reducing dimensionality.
  - **Pattern recognition:** Does the actual job of classifying or describing observations, relying on the extracted features.
- **System diagram:**



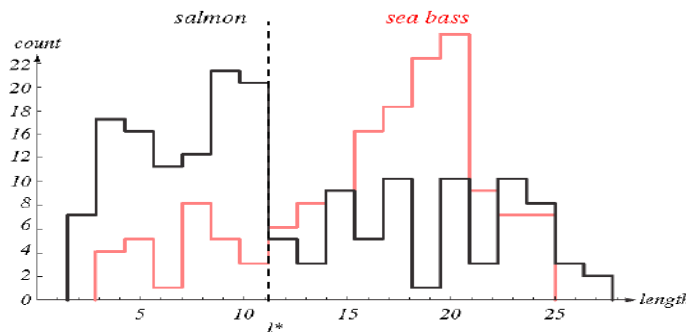


# Bayesian Decision Theory

- ▶ 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  $w$ , the type of fish we observe (state of nature), as a random variable where
  - ▶  $w = w_1$  for sea bass
  - ▶  $w = w_2$  for salmon
  - ▶  $P(w_1)$  is the **a priori probability** that the next fish is a sea bass
  - ▶  $P(w_2)$  is the **a priori probability** that the next fish is a salmon



Figure: Picture taken from a camera.



---

# Bayesian Decision Theory

---

- **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  $P(w_1)$  and  $P(w_2)$ ?
  - ▶ Set  $P(w_1) = P(w_2)$  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

$$P(w_1) + P(w_2) = 1$$

(exclusivity and exhaustivity)

- ▶ How can we make a decision with only the prior information?  
Decide  $\begin{cases} w_1 & \text{if } P(w_1) > P(w_2) \\ w_2 & \text{otherwise} \end{cases}$
- ▶ What is the **probability of error** for this decision?

$$P(\text{error}) = \min\{P(w_1), P(w_2)\}$$

---

# Bayesian Decision Theory

---

- **Class-conditional Probabilities:**
  - ▶ Let's try to improve the decision using the lightness measurement  $x$  ( $\in \mathbf{R}$ ).
  - ▶ Let  $x$  be a continuous random variable.
  - ▶ Define  $p(x|w_j)$  as the **class-conditional probability density** (probability of  $x$  given that the state of nature is  $w_j$  for  $j = 1, 2$ ).
  - ▶  $p(x|w_1)$  and  $p(x|w_2)$  describe the difference in lightness between populations of sea bass and salmon.

---

# Bayesian Decision Theory

---

- **Posterior Probabilities:**

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

MAP estimate

$$P(w_j|x) = \frac{P(x|w_j)P(w_j)}{P(x)} \longrightarrow \text{posterior} = \frac{\text{prior} \times \text{likelihood}}{\text{evidence}}.$$

where  $P(x) = \sum_{j=1}^2 P(x|w_j)P(w_j)$

# Bayesian Decision Theory

- Making a Decision:

- ▶  $P(x|w_j)$  is called the **likelihood** and  $P(x)$  is called the **evidence**.
- ▶ How can we make a decision after observing the value of  $x$ ?

$$\text{Decide } \begin{cases} w_1 & \text{if } P(w_1|x) > P(w_2|x) \\ w_2 & \text{otherwise} \end{cases}$$

- ▶ Rewriting the rule gives

$$\text{Decide } \begin{cases} w_1 & \text{if } \frac{P(w_1|x)}{P(w_2|x)} > \frac{P(w_1)}{P(w_2)} \\ w_2 & \text{otherwise} \end{cases}$$

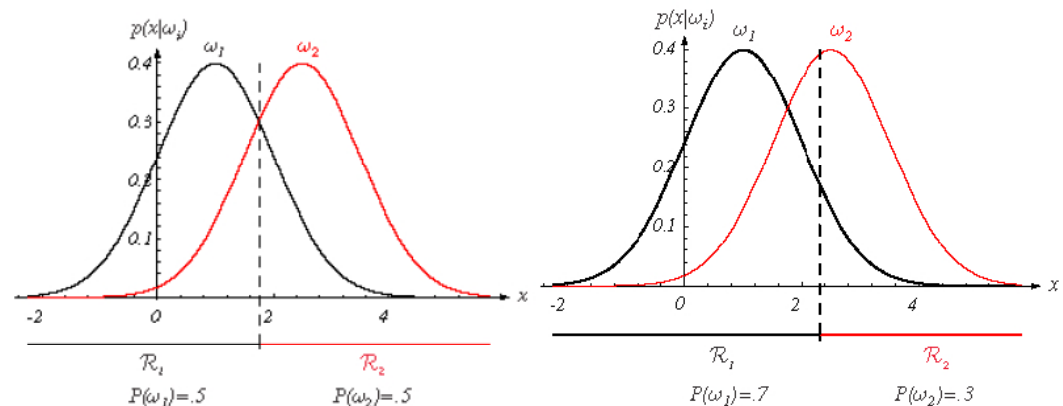


Figure: Optimum thresholds for different priors.

# Bayesian Decision Theory

- Probability of Error:

- What is the probability of error for this decision?

$$P(error|x) = \begin{cases} P(w_1|x) & \text{if we decide } w_2 \\ P(w_2|x) & \text{if we decide } w_1 \end{cases}$$

- What is the average probability of error?

$$p(error) = \int_{-\infty}^{+\infty} P(error, x) dx = \int_{-\infty}^{+\infty} P(error|x) P(x) dx$$

- Bayes decision rule minimizes this error because

$$P(error|x) = \min\{P(w_1|x), P(w_2|x)\}$$

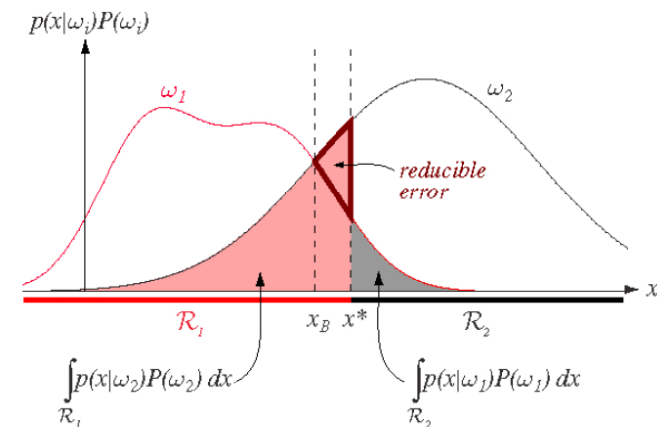


Figure: Components of the probability of error for equal priors and the non-optimal decision point  $x^*$ . The optimal point  $x_B$  minimizes the total shaded area and gives the Bayes error rate.

---

# Bayesian Decision Theory

---

How can we generalize to

- ▶ more than one feature?
  - ▶ replace the scalar  $x$  by the feature vector  $\mathbf{x}$
- ▶ more than two states of nature?
  - ▶ just a difference in notation
- ▶ allowing actions other than just decisions?
  - ▶ allow the possibility of rejection
- ▶ different risks in the decision?
  - ▶ define how costly each action is

---

# Bayesian Decision Theory

---

- **Minimum-error-rate Classification:**

- ▶ Let  $\{w_1, \dots, w_c\}$  be the finite set of  $c$  states of nature (**classes**, **categories**).
- ▶ Let  $\mathbf{x}$  be the  $d$ -component vector-valued random variable called the **feature vector**.
- ▶ If all errors are equally costly, the minimum-error decision rule is defined as  
Decide  $w_i$  if  $P(w_i|x) > P(w_j|x) \forall j \neq i$
- ▶ 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.
- ▶ How can we estimate (learn) the unknown  $p(\mathbf{x}|w_j), j = 1, \dots, c$ ?
- ▶ Parametric models: assume that the form of the density functions are known
  - ▶ Density models (e.g., Gaussian)
  - ▶ Mixture models (e.g., mixture of Gaussians)
  - ▶ Hidden Markov Models
  - ▶ Bayesian Belief Networks
- ▶ Non-parametric models: no assumption about the form
  - ▶ Histogram-based estimation
  - ▶ Parzen window estimation
  - ▶ Nearest neighbour estimation

# Bayesian Decision Theory

- The Gaussian Density:

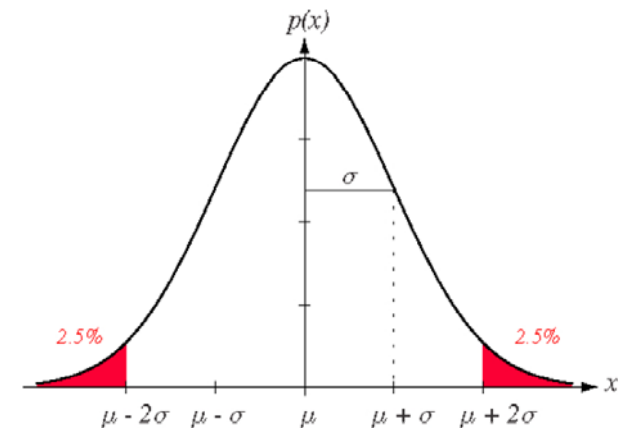
- ▶ Gaussian can be considered as a model where the feature vectors for a given class are continuous-valued, randomly corrupted versions of a single typical or prototype vector.
- ▶ Some properties of the Gaussian:
  - ▶ Analytically tractable
  - ▶ Completely specified by the 1st and 2nd moments
  - ▶ Has the maximum entropy of all distributions with a given mean and variance
  - ▶ Many processes are asymptotically Gaussian (Central Limit Theorem)
  - ▶ Uncorrelatedness implies independence
- ▶ For  $x \in \mathbf{R}$ :

$$p(x) = N(\mu, \sigma^2) = \frac{1}{\sqrt{2\pi}\sigma} \exp \left[ -\frac{1}{2} \left( \frac{x - \mu}{\sigma} \right)^2 \right]$$

where

$$\mu = E[x] = \int_{-\infty}^{+\infty} xP(x)dx$$

$$\sigma^2 = E[(x - \mu)^2] = \int_{-\infty}^{+\infty} (x - \mu)^2 P(x)dx$$



# Bayesian Decision Theory

- Multivariate Gaussian:

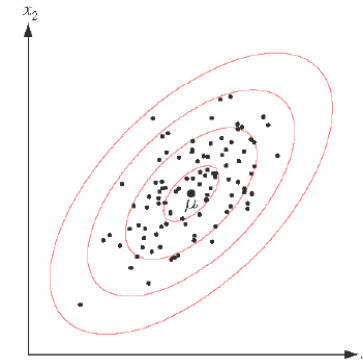
► For  $\mathbf{x} \in \mathbf{R}^d$ :

$$p(\mathbf{x}) = N(\boldsymbol{\mu}, \Sigma) = \frac{1}{(2\pi)^{d/2} |\Sigma|^{1/2}} \exp \left[ -\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu})^T \Sigma^{-1} (\mathbf{x} - \boldsymbol{\mu}) \right]$$

where

$$\boldsymbol{\mu} = E(\mathbf{x}) = \int \mathbf{x} P(\mathbf{x}) d\mathbf{x}$$

$$\Sigma = E[(\mathbf{x} - \boldsymbol{\mu})(\mathbf{x} - \boldsymbol{\mu})^T]$$



**Figure:** Samples drawn from a two-dimensional Gaussian lie in a cloud centered on the mean  $\boldsymbol{\mu}$ . The loci of points of constant density are the ellipses for which  $(\mathbf{x} - \boldsymbol{\mu})^T \Sigma^{-1} (\mathbf{x} - \boldsymbol{\mu})$  is constant, where the eigenvectors of  $\Sigma$  determine the direction and the corresponding eigenvalues determine the length of the principal axes. The quantity  $r^2 = (\mathbf{x} - \boldsymbol{\mu})^T \Sigma^{-1} (\mathbf{x} - \boldsymbol{\mu})$  is called the squared **Mahalanobis distance** from  $\mathbf{x}$  to  $\boldsymbol{\mu}$ .

# Bayesian Decision Theory

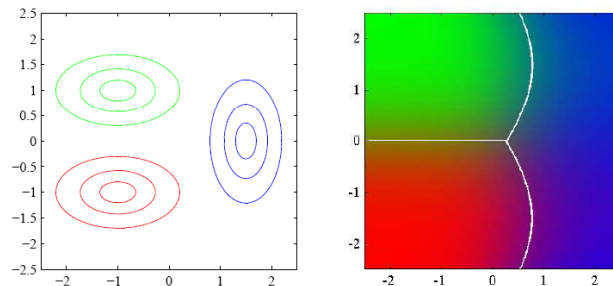
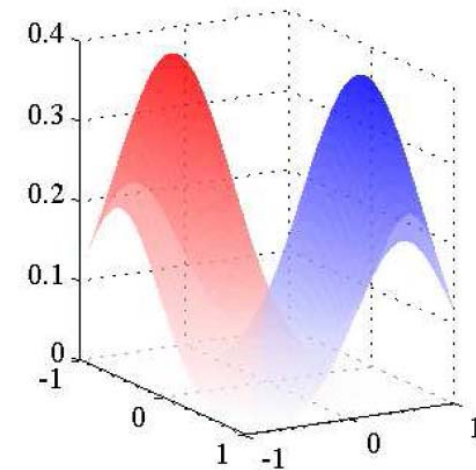
- **Bayes Linear Classifier:**

- ▶ 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

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

- ▶ assuming only 2 classes the decision boundary is linear: check this!

The decision surface is planar when the covariance matrices are the same and quadratic when they are not.



# Bayesian Decision Theory

- Plug-in classifier:

Finding Templates Using Classifiers Chap. 22

**Algorithm 22.2:** A plug-in classifier can be used to classify objects into classes if the class-conditional densities are known to be normal

Assume we have  $N$  classes, and the  $k$ th class contains  $N_k$  examples, of which the  $i$ th is written as  $\mathbf{x}_{k,i}$ .

For each class  $k$ , estimate the mean and standard deviation for that class-conditional density.

$$\mu_k = \frac{1}{N_k} \sum_{i=1}^{N_k} \mathbf{x}_{k,i}; \quad \Sigma_k = \frac{1}{N_k - 1} \sum_{i=1}^{N_k} (\mathbf{x}_{k,i} - \mu_k)(\mathbf{x}_{k,i} - \mu_k)^T;$$

To classify an example  $\mathbf{x}$ ,

Choose the class  $k$  with the smallest value of  $\delta(\mathbf{x}; \mu_k, \Sigma_k)^2 - Pr\{k\} + \frac{1}{2} \log |\Sigma_k|$

where

$$\delta(\mathbf{x}; \mu_k, \Sigma_k) = \frac{1}{2} \left( (\mathbf{x} - \mu_k)^T \Sigma_k^{-1} (\mathbf{x} - \mu_k) \right)^{(1/2)}.$$

# Bayesian Decision Theory

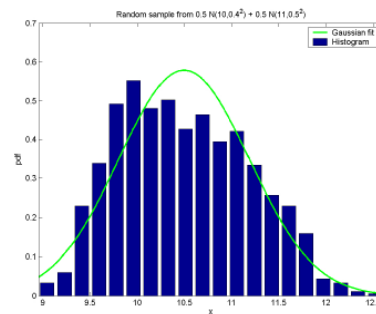
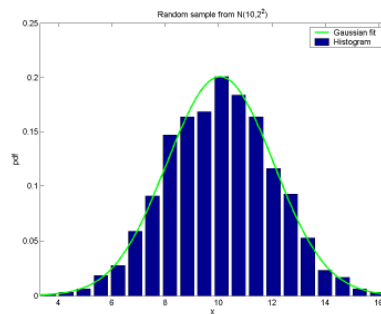
- Gaussian Density Estimation:

- ▶ Bayesian Decision Theory shows us how to design an optimal classifier if we know the prior probabilities  $P(w_i)$  and the class-conditional densities  $p(\mathbf{x}|w_i)$ .
- ▶ Unfortunately, we rarely have complete knowledge of the probabilistic structure.
- ▶ However, we can often find design samples or **training data** that include particular representatives of the patterns we want to classify.
- ▶ The **maximum likelihood estimates** of a Gaussian are  $\hat{\mu} = \frac{1}{n} \sum_{i=1}^n \mathbf{x}_i$  and  $\hat{\Sigma} = \frac{1}{n} \sum_{i=1}^n (\mathbf{x}_i - \hat{\mu})(\mathbf{x}_i - \hat{\mu})^T$



For more details see:

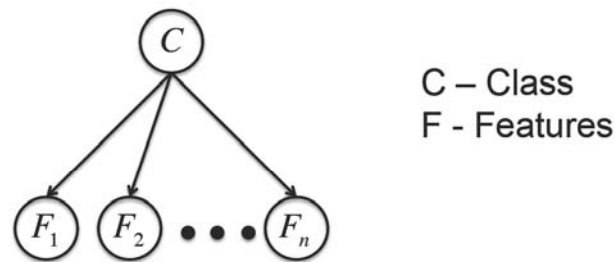
<http://www.autonlab.org/tutorials/mle13.pdf>



# Bayesian Decision Theory

- **Naïve Bayes classifier:**

- Given a large number of features it is very difficult to estimate  $P(F|C)$  due to all the correlations between features (sorry about the variable name change).
- It is easier if we assume independent features (uncorrelated Gaussian features).



$$P(C, F_1, F_2, \dots, F_n) = P(C) \prod_i P(F_i | C)$$

We only specify (parameters):

$P(C)$  prior over class labels

$P(F_i | C)$  how each feature depends on the class

---

# Bayesian Decision Theory

---

- Naïve Bayes classifier:

- Assuming uncorrelated features makes all the math much simpler:

$$p(C|F_1, \dots, F_n) = \frac{p(C) p(F_1, \dots, F_n|C)}{p(F_1, \dots, F_n)}.$$

$$p(C, F_1, \dots, F_n) = p(C) p(F_1|C) p(F_2|C, F_1) p(F_3|C, F_1, F_2) \dots p(F_n|C, F_1, F_2, F_3, \dots, F_{n-1}).$$

- Naïve assumption:

$$p(F_i|C, F_j) = p(F_i|C)$$

- Allowing for:

$$p(C, F_1, \dots, F_n) = p(C) p(F_1|C) p(F_2|C) p(F_3|C) \dots = p(C) \prod_{i=1}^n p(F_i|C).$$

- PCA can be use to de-correlate the features!



---

# Bayesian Decision Theory

---

- **Classification error:**

- ▶ To apply these results to multiple classes, separate the training samples to  $c$  subsets  $\mathcal{D}_1, \dots, \mathcal{D}_c$ , with the samples in  $\mathcal{D}_i$  belonging to class  $w_i$ , and then estimate each density  $p(x|w_i, \mathcal{D}_i)$  separately.
- ▶ Different sources of error:
  - ▶ Bayes error: due to overlapping class-conditional densities (related to features used)
  - ▶ Model error: due to incorrect model
  - ▶ Estimation error: due to estimation from a finite sample (can be reduced by increasing the amount of training data)

---

# Dimensionality

---

- **Curse of Dimensionality:**

- ▶ In practical multiclass applications, it is not unusual to encounter problems involving tens or hundreds of features.
- ▶ Intuitively, it may seem that each feature is useful for at least some of the discriminations.
- ▶ There are two issues that we must be careful about:
  - ▶ How is the classification accuracy affected by the dimensionality (relative to the amount of training data)?
  - ▶ How is the computational complexity of the classifier affected by the dimensionality?
- ▶ In general, if the performance obtained with a given set of features is inadequate, it is natural to consider adding new features.
- ▶ Unfortunately, it has frequently been observed in practice that, beyond a certain point, adding new features leads to worse rather than better performance.
- ▶ This is called the **curse of dimensionality**.

---

# Dimensionality

---

- How to avoid problems with dimensionality:
  - ▶ All of the commonly used classifiers can suffer from the curse of dimensionality.
  - ▶ While an exact relationship between the probability of error, the number of training samples, the number of features, and the number of parameters is very difficult to establish, some guidelines have been suggested.
  - ▶ It is generally accepted that using at least ten times as many training samples per class as the number of features ( $n/d > 10$ ) is a good practice.
  - ▶ The more complex the classifier, the larger should the ratio of sample size to dimensionality be.
- The complexity of the classifier increases as features are more heavily correlated, far from Gaussian and overlapped.
- The rule of  $n/d > 10$  is only really valid for independent features

---

# Dimensionality

---

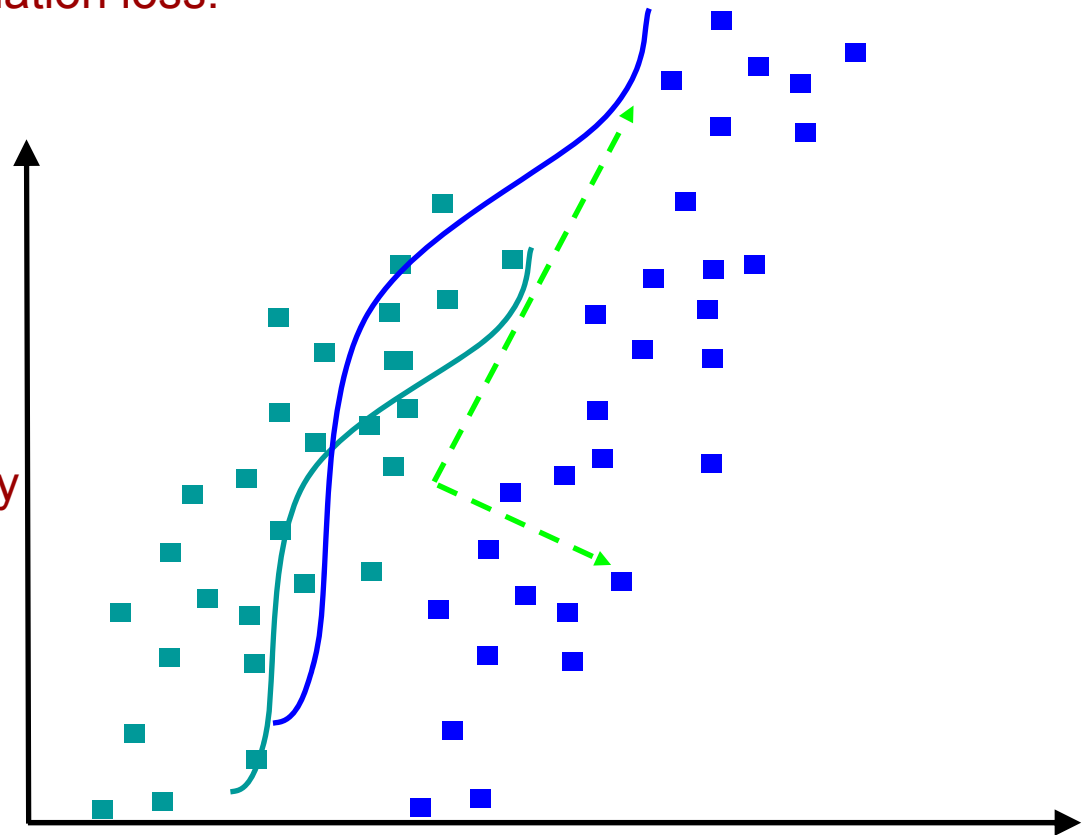
- Reducing dimensionality:

Dimensionality can be reduced by

- ▶ redesigning the features
- ▶ selecting an appropriate subset among the existing features (feature selection)
- ▶ transforming to different feature spaces
  - ▶ Principal Components Analysis (PCA) seeks a projection that best represents the data in a leastsquares sense.
  - ▶ Linear Discriminant Analysis (LDA) seeks a projection that best separates the data in a least squares sense.

# PCA – Principal Component analysis

- Finds the most accurate data representation and enable data representation in a lower dimensional space without much information loss.
  - Assumes that variance is equal to information
  - Assumes Gaussian variables
  - Uses L2 norm
- In the resulting basis axis are aligned with major co-variance directions and axis which do not represent large variances may be removed without loss of information.

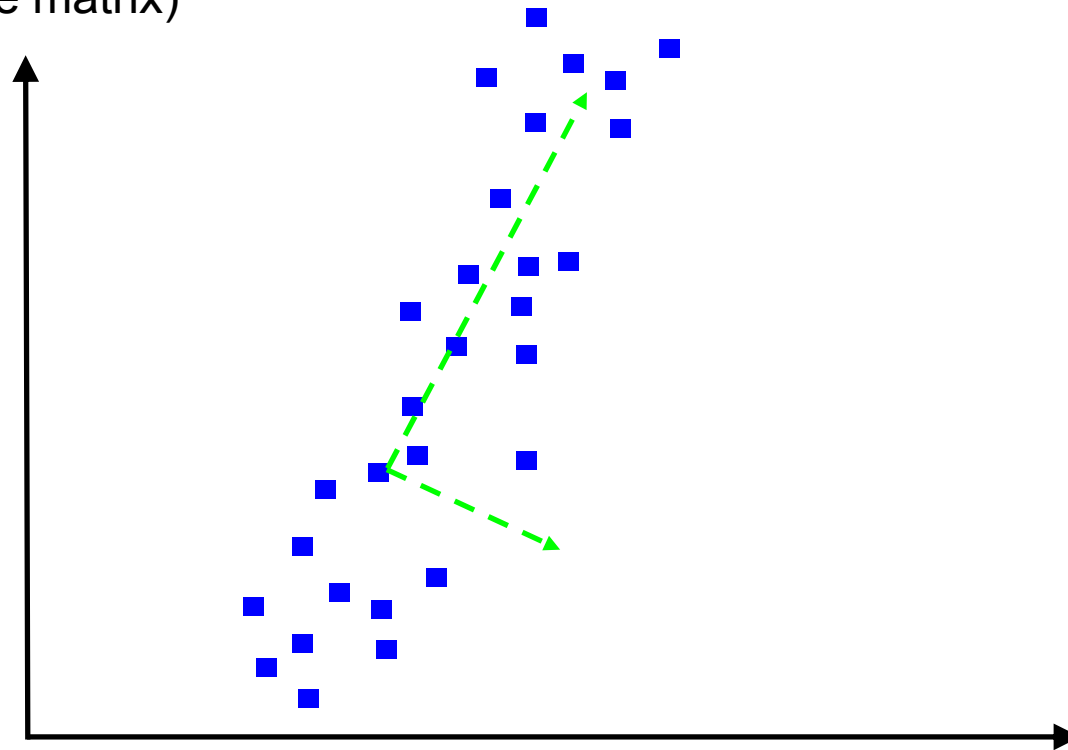


---

# PCA – Principal Component analysis

---

- PCA – Principal Component analysis
  - Useful for better Gaussian modeling of the data ( eliminates the need for a full covariance matrix)



# Feature space selection

- Principle component analysis (PCA)

- PCA algorithm:

- Obtain input data vectors and compute mean input vector
    - Subtract mean input vector to all data vectors
    - Compute input data covariance matrix
    - Compute eigenvectors and eigenvalues from the covariance matrix
    - Choosing the components and forming the feature vector
      - removing components which correspond to the small eigen vectors

- Computation:

SVD for PCA

- SVD can be used to efficiently compute the image basis

$$PP^T = (U \Sigma V^T)(U \Sigma V^T)^T = U \Sigma V^T V \Sigma^T U^T = U \Sigma^T \Sigma U^T = U \Sigma^2 U^{-1}$$

$$(PP^T)U = U \Sigma^2$$

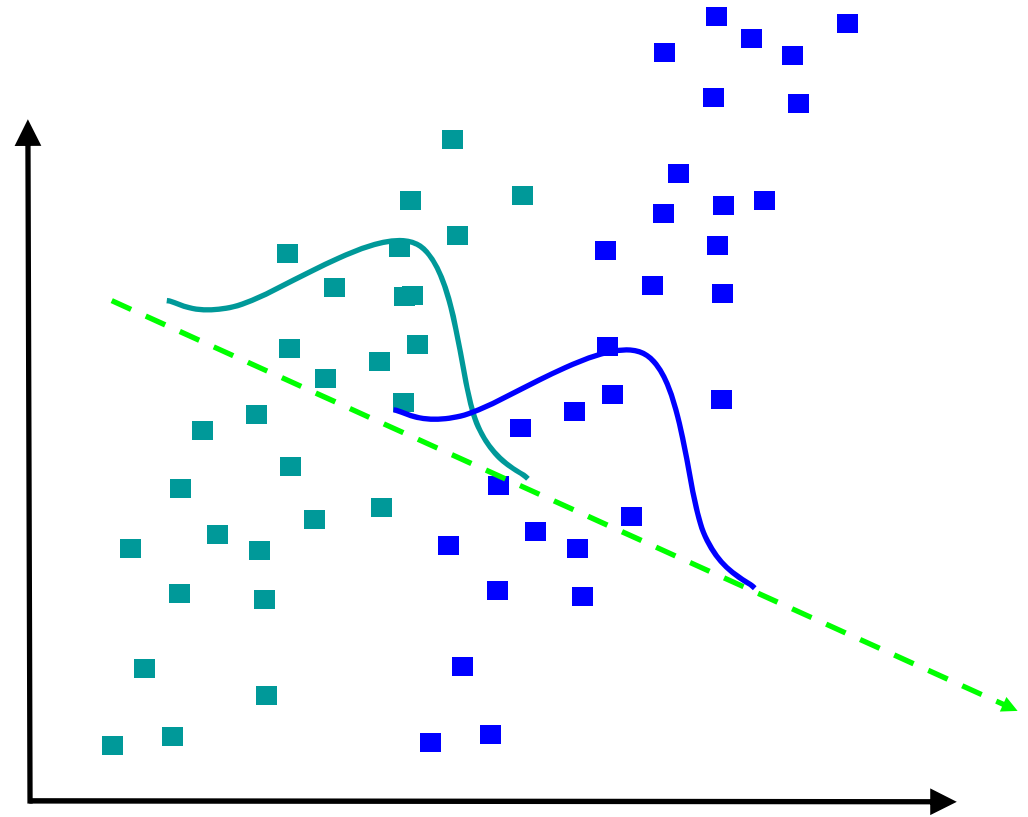
$$(PP^T)v = v\lambda$$

- $U$  are the eigen vectors (image basis)
- Most important thing to notice: Distance in the eigen-space is an approximation to the correlation in the original space

$$\|x_i - x_j\| \approx \|c_i - c_j\|$$

# LDA - Linear Discriminant Analysis

- Also known as Fisher's linear discriminant analysis:
  - Projection that best separates the data in a least-squares sense.
  - Projection of n-dimensional data onto a line.





---

# Fisher's linear discriminants

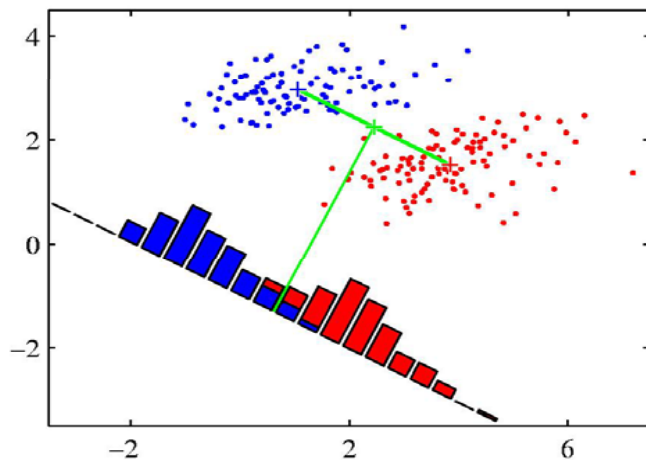
---

- A simple linear discriminant function is a projection of the data down to 1-D.
  - So choose the projection that gives the best separation of the classes.  
What do we mean by “best separation”?
- An obvious direction to choose is the direction of the line joining the class means.
  - But if the main direction of variance in each class is not orthogonal to this line, this will not give good separation (see the next figure).
- LDA chooses the direction that maximizes the ratio of between class variance to within class variance.
  - This is the direction in which the projected points contain the most information about class membership (under Gaussian assumptions)

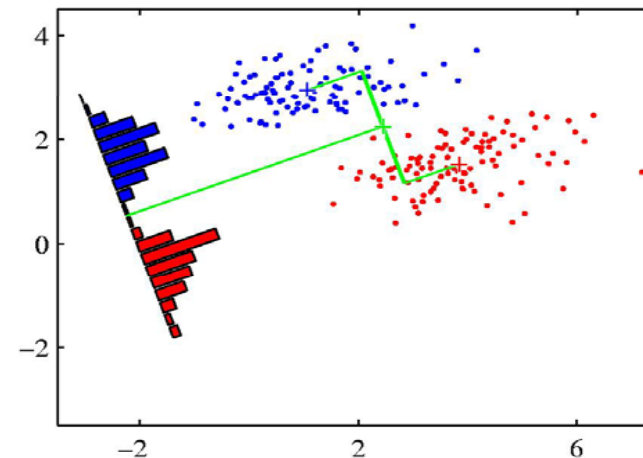
---

# Illustration of the advantage of Fisher's linear discriminants

---



When projected onto the line joining the class means, the classes are not well separated.



Fisher chooses a direction that makes the projected classes much tighter, even though their projected means are less far apart.

# Math of Fisher's linear discriminants

- What linear transformation is best for discrimination?

$$y = \mathbf{w}^T \mathbf{x}$$

- The projection onto the vector separating the class means seems sensible:

$$\mathbf{w} \propto \mathbf{m}_2 - \mathbf{m}_1$$

- But we also want small variance within each class:

$$s_1^2 = \sum_{n \in C_1} (y_n - m_1)^2$$

$$s_2^2 = \sum_{n \in C_2} (y_n - m_2)^2$$

- Fisher's objective function is:

$$J(\mathbf{w}) = \frac{(m_2 - m_1)^2}{s_1^2 + s_2^2}$$

← between  
← within

---

## More math of Fisher's linear discriminants

---

$$J(\mathbf{w}) = \frac{(m_2 - m_1)^2}{s_1^2 + s_2^2} = \frac{\mathbf{w}^T \mathbf{S}_B \mathbf{w}}{\mathbf{w}^T \mathbf{S}_W \mathbf{w}}$$

$$\mathbf{S}_B = (\mathbf{m}_2 - \mathbf{m}_1) (\mathbf{m}_2 - \mathbf{m}_1)^T$$

$$\mathbf{S}_W = \sum_{n \in C_1} (\mathbf{x}_n - \mathbf{m}_1) (\mathbf{x}_n - \mathbf{m}_1)^T + \sum_{n \in C_2} (\mathbf{x}_n - \mathbf{m}_2) (\mathbf{x}_n - \mathbf{m}_2)^T$$

$$\text{Optimal solution : } \mathbf{w} \propto \mathbf{S}_W^{-1} (\mathbf{m}_2 - \mathbf{m}_1)$$

# Feature space selection

- PCA vs LDA examples:

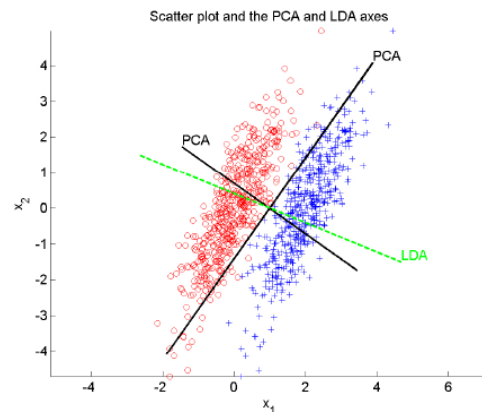


Figure: Scatter plot.

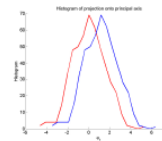


Figure: Projection onto the first PCA axis.

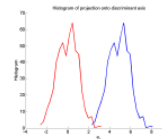


Figure: Projection onto the first LDA axis.

Scatter plot and the PCA and LDA axes for a bivariate sample with two classes. Histogram of the projection onto the first LDA axis shows better separation than the projection onto the first PCA axis. ➡

# Feature space selection

- PCA vs LDA examples:

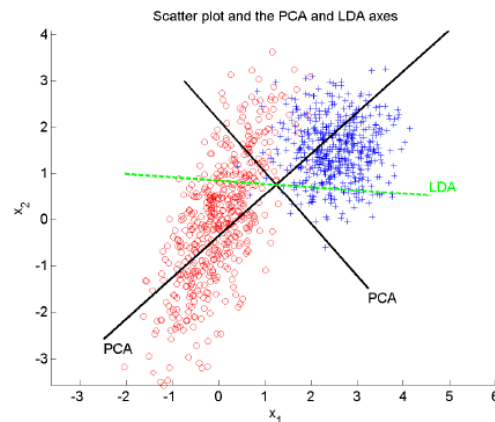


Figure: Scatter plot.

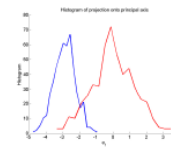


Figure: Projection onto the first PCA axis.

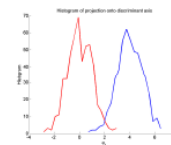


Figure: Projection onto the first LDA axis.

Scatter plot and the PCA and LDA axes for a bivariate sample with two classes. Histogram of the projection onto the first LDA axis shows better separation than the projection onto the first PCA axis. ➡

---

# Non-Bayesian Classifiers

---

- Distance-based classifiers:
  - Nearest neighbor classifier
- Decision boundary-based classifiers:
  - Decision trees
  - Neural networks
  - Support vector machines

---

# Non-Bayesian Classifiers

---

- The k-Nearest neighbor Classifier:

- ▶ Given the training data  $\mathcal{D} = \{\mathbf{x}_1, \dots, \mathbf{x}_n\}$  as a set of  $n$  labeled examples, the nearest neighbour classifier assigns a test point  $\mathbf{x}$  the label associated with its closest neighbour in  $\mathcal{D}$ .

The k-nearest neighbour classifier classifies  $\mathbf{x}$  by assigning it the label most frequently represented among the  $k$  nearest samples.

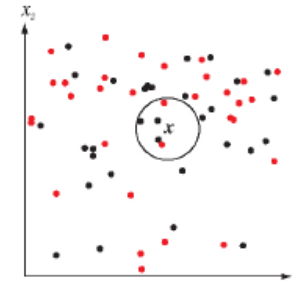


Figure: Classifier for  $k = 5$ .

- ▶ Closeness is defined using a distance function.



# Non-Bayesian Classifiers

- Distance Functions:

- ▶ A general class of metrics for  $d$ -dimensional patterns is the **Minkowski metric**

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

also referred to as the  $L_p$  norm.

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

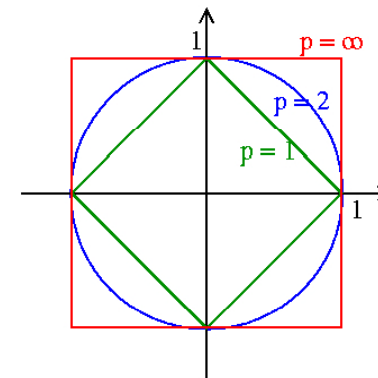
$$L_2(\mathbf{x}, \mathbf{y}) = \left( \sum_{i=1}^d |x_i - y_i|^2 \right)^{1/2}$$

- ▶ The **Manhattan or city block distance** is the  $L_1$  norm

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

The  $L_\infty$  norm is the maximum of the distances along individual coordinate axes

$$L_\infty(\mathbf{x}, \mathbf{y}) = \max_{i=1}^d |x_i - y_i|$$



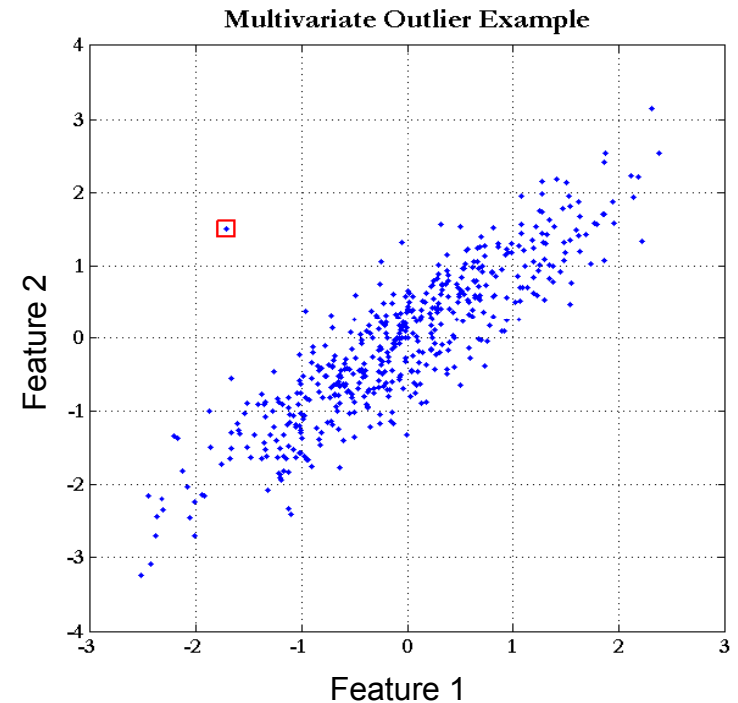
**Figure:** Each colored shape consists of points at a distance 1.0 from the origin, measured using different values of  $p$  in the Minkowski  $L_p$  metric.

# Non-Bayesian Classifiers

- Mahalanobis distance:
  - Based on the covariance of each feature with the class examples.
    - Based on the assumption that distances in the direction of high variance are less important
    - Highly dependent on a good estimate of covariance.
  - Superior to the Euclidean distance.

$$D_M(x) = \sqrt{(x - \mu)^T P^{-1} (x - \mu)}.$$

where  $P$  is the covariance matrix



# Non-Bayesian Classifiers

- Linear Discriminant Functions:

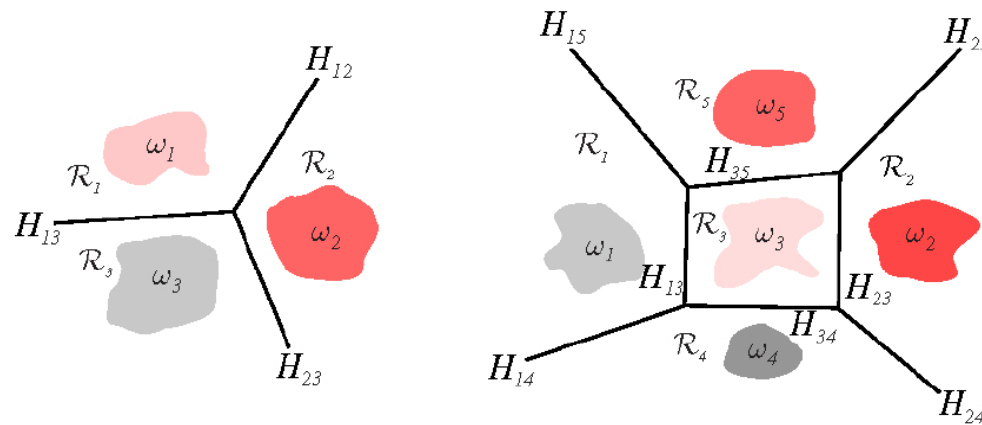


Figure: Linear decision boundaries produced by using one linear discriminant for each class.

---

# Non-Bayesian Classifiers

---

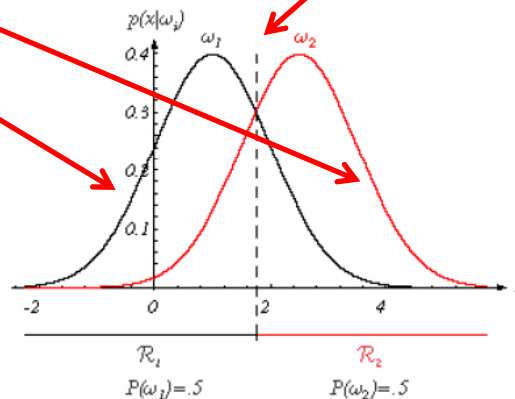
- **Discriminative vs Generative models:**
  - **Discriminative models** are a class of models used in machine learning for modelling the dependence of an unobserved variable  $y$  on an observed variable  $x$ . Within a statistical framework, this is done by modelling the conditional probability distribution  $P(y | x)$ , which can be used for predicting  $y$  from  $x$ .
  - **Generative models** can randomly generate observable data, typically given some hidden parameters. It specifies a joint probability distribution over observation and label sequences. Generative models are used in machine learning for either modelling data directly or as an intermediate step to forming a conditional probability density function. A conditional distribution can be formed from a generative model through the use of Bayes' rule.
- Discriminative models differ from generative models in that they do not allow one to generate samples from the joint distribution of  $x$  and  $y$ . However, for tasks such as classification and regression that do not require the joint distribution, discriminative models generally yield superior performance. On the other hand, generative models are typically more flexible than discriminative models in expressing dependencies in complex learning tasks.

# Non-Bayesian Classifiers

- Discriminative vs Generative models:

Generative models  
estimate the  
distributions

Discriminative models  
go right to the point and  
define a decision  
boundary



# Non-Bayesian Classifiers

- Decision trees:

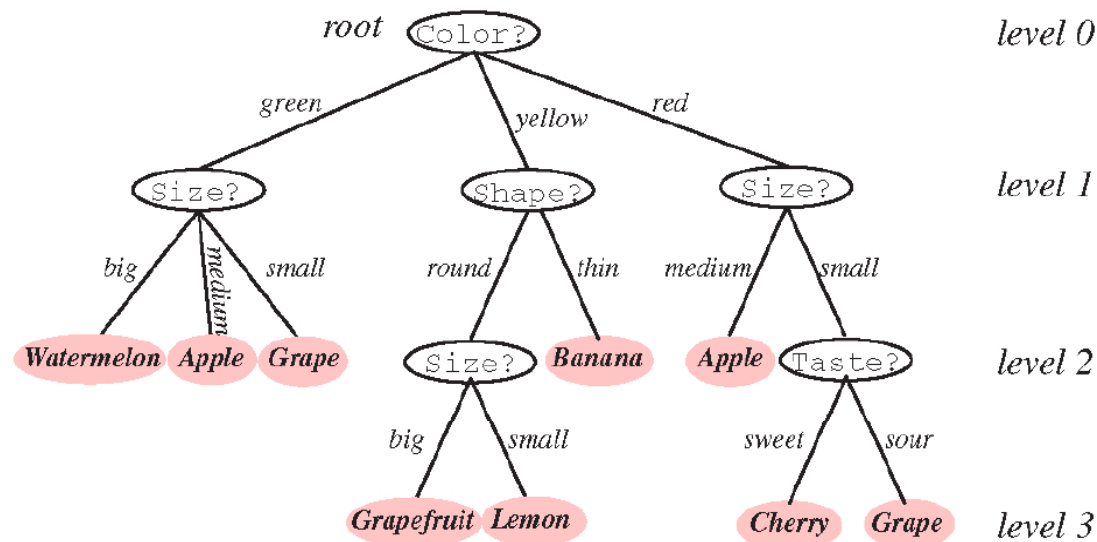


Figure: Decision trees classify a pattern through a sequence of questions, in which the next question asked depends on the answer to the current question.

---

# Non-Bayesian Classifiers

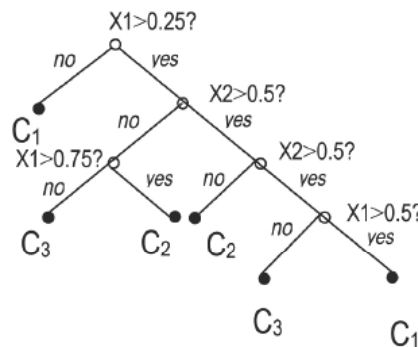
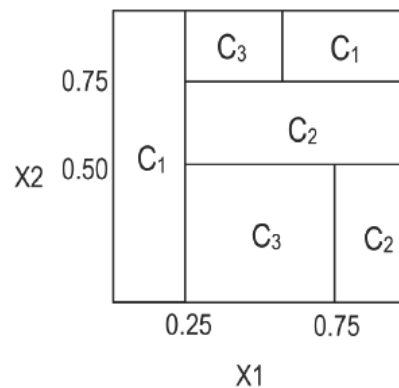
---

- Decision trees:
  - ▶ Decision trees are hierarchical decision systems in which conditions are sequentially tested until a class is accepted
  - ▶ To this end, the feature space is split into unique regions, corresponding to the classes, in a sequential manner
  - ▶ Upon the arrival of a feature vector, the searching of the region to which the feature vector will be assigned is achieved via a sequence of decisions along a path of nodes of an appropriately constructed tree

# Non-Bayesian Classifiers

- Decision trees:

- ▶ The most popular schemes among decision trees are those that split the space into hyperrectangles with sides parallel to the axes
- ▶ The sequence of decisions is applied to individual features, and the questions to be answered are of the form “is the feature  $x_k \leq \alpha$ ?”, where  $\alpha$  is a threshold value
- ▶ Such trees are known as ordinary binary classification trees (OBCTs).



```
proc growtree(data)
  if (data not perfectly classified)
    find 'best' splitting attribute A
    for each a in A
      create child a
      data_a = data restricted to A = a
      growtree(data_a)
    endfor
  endif
endproc
```



---

# Non-Bayesian Classifiers

---

- **Decision trees:**

In order to develop a binary decision tree, the following design elements have to be considered in the training phase:

1. At each node, the set of candidate questions to be asked has to be decided. Each question corresponds to a specific binary split into two descendant nodes. Each node  $t$  is associated with a specific subset  $S_t$  of the training set  $S$ . The splitting of a node is equivalent to the split of the subset  $S_t$  into two disjoint descendants subsets  $S_{t,Y}$  and  $S_{t,N}$ . The first of the two consists of the examples in  $S_t$  that correspond to the answer "Yes" of the question and those of the second to the "No" answer. The first node (root) of the tree is associated with the training set  $S$ .
2. A splitting criterion must be adopted according to which the best split from the set of candidate ones is chosen.
3. A stop-splitting rule is required that controls the growth of the tree and a node is declared as a terminal (leaf).
4. A rule is required that assigns each leaf to a specific class.

---

# Non-Bayesian Classifiers

---

- **Decision trees:**

- Set of questions**

- For the OBCT type of trees the questions are of the form “Is  $x_i \leq \alpha$ ?”. For each feature, every possible value of the threshold  $\alpha$  defines a specific split of the subset  $S_t$ . Thus in theory, an infinite set of questions has to be asked if  $\alpha$  varies in an interval in  $\mathbf{R}$ . In practice, only a finite set of questions need to be considered. Since the number of training examples  $N$  is finite, any of the features  $x_i$  can take at most  $N_i \leq N$  different values. Thus, for feature  $x_i$ , one can use as possible  $\alpha$  values the midvalues of two consecutive distinct values of  $x_i$ . The same has to be repeated for all features. Thus, in such case, the total number of candidate questions is majored by  $\sum_{i=1}^d N_i$ . However, only one of them has to be chosen to provide the binary split at the current node of the tree. This is selected to be the one that leads to the best split of the associated subset  $S_t$ . The best split is decided according to a splitting criterion.

---

# Non-Bayesian Classifiers

---

- Decision trees:

## Splitting criterion I

Every binary split of a node  $t$  generates two descendant nodes, be them  $t_L$  and  $t_R$ , each one associated with two new subsets,  $S_{t,L}$  and  $S_{t,R}$ , respectively. There is a class of impurity measures that quantify how impure each node  $t$  of the tree is, where purity occurs when all cases in a node belong to just one class. From the root node to the leaves, every split must generate subsets that are more homogeneous compared to the ancestor set  $S_t$ .

One can define the goodness of the split  $s$  (comprising a threshold  $\alpha$  and a feature  $x_i$ ) as the decrease of impurity from the ancestor node to the descendants, reaching more “class homogeneous” descendants subsets.

---

# Non-Bayesian Classifiers

---

- **Decision trees:**

- Splitting criterion II**

- If one then adopt a split  $s$  at the node  $t$ , with the proportion of examples going into node  $t_L$  being  $p_L$  and the proportion going into node  $t_R$  being  $p_R$ , using the impurity function  $I$ , one can measure the change of impurity originated by that split as:

- $$\Delta I(s, t) = I(t) - [p_L I_L + p_R I_R]$$

- The greater this difference, the greater the decrease of impurity and purer nodes are reached. Therefore, one chooses the split that maximizes this expression, which is equivalent to minimize

- $$p_L I_L + p_R I_R$$

- while the difference remains positive; otherwise the split must not be performed.

- Impurity measures commonly used in classification trees include:

- ▶ Entropy index:  $I(s, t) = - \sum_{i=1}^K p(C_i|t) \log_2 p(C_i|t)$
    - ▶ Gini index:  $I(s, t) = \sum_{i=1, j=1, i \neq j}^K p(C_i|t) p(C_j|t)$

- where  $p(C_i|t)$  denotes the probability that a vector in the subset  $S_t$ , associated with a node  $t$ , belongs to class  $C_i$ ,  $i = 1, \dots, K$ .

---

# Non-Bayesian Classifiers

---

- **Decision trees:**

- Stop-splitting rule**

The natural question that now arises is when one decides to stop splitting a node and declares it as a leaf of the tree. A possibility is to stop splitting when the purity improvement of the best split is below an adopted threshold. Other alternatives are to stop splitting either if the cardinality of the subset  $S_t$  is small enough or if  $S_t$  is pure, in the sense that all points in it belong to a single class. Experience has shown that the use of a threshold value for the impurity decrease as a stop-splitting rule does not lead to satisfactory results. Many times it stops tree growing either too early or too late. The most commonly used approach is to grow the tree up to a large size first and then prune nodes according to a pruning criterion. A number of pruning criteria have been suggested. A commonly approach is to combine an estimate of the error probability with a complexity measuring term (e.g. number of terminal nodes)

---

# Non-Bayesian Classifiers

---

- **Decision trees:**

## **Class assignment rule**

Once a node is declared to be a leaf, then it has to be given a class label.

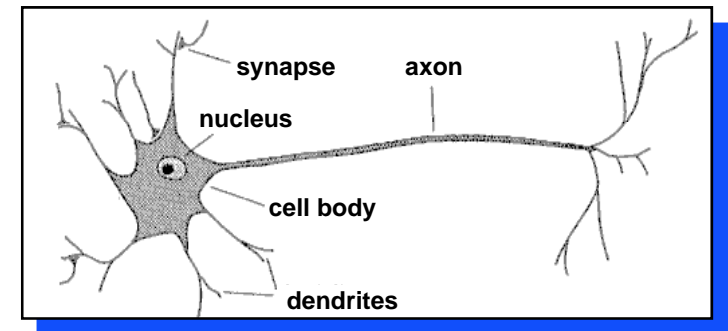
- ▶ For regression, a commonly used rule is to minimize the mean square error, leading to predict as the average of the  $y_i$  in the leaf.
- ▶ For classification of nominal classes, a commonly used rule is the majority rule, i.e., the leaf is labelled as the class most represented in the leaf (mode).
- ▶ For classification of ordinal classes, a better rule is to select the median of the values in the leaf.

---

# Non-Bayesian Classifiers

---

- **MLP:**
  - Formulated from loose biological principles
  - Popularized mid 1980s
    - Rumelhart, Hinton & Williams 1986
    - Werbos 1974, Ho 1964
- “learn” pre-processing stage from data
- **layered, feed-forward structure**
  - sigmoidal pre-processing
  - task-specific output



**non-linear model**

---

# Perceptron

---

- “Perceptrons” describes a whole family of learning machines, but the standard type consisted of a layer of fixed non-linear basis functions followed by a simple linear discriminant function.
  - They were introduced in the late 1950’s and they had a simple online learning procedure.
  - Grand claims were made about their abilities. This led to lots of controversy.
  - Researchers in symbolic AI emphasized their limitations (as part of an ideological campaign against real numbers, probabilities, and learning)

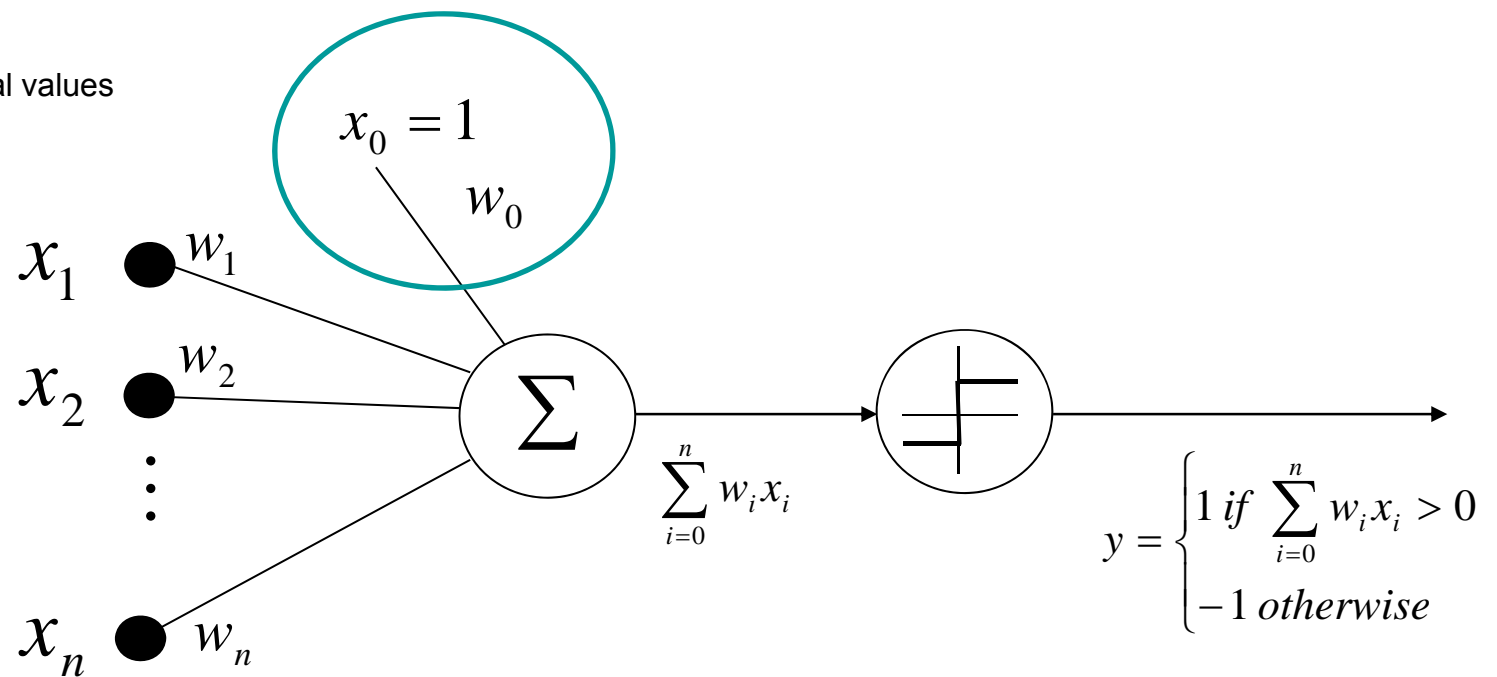
$$y(\mathbf{x}, \mathbf{w}) = f \left( \sum_{j=1}^M w_j \phi_j(\mathbf{x}) \right)$$



# Perceptron

- **Model:**

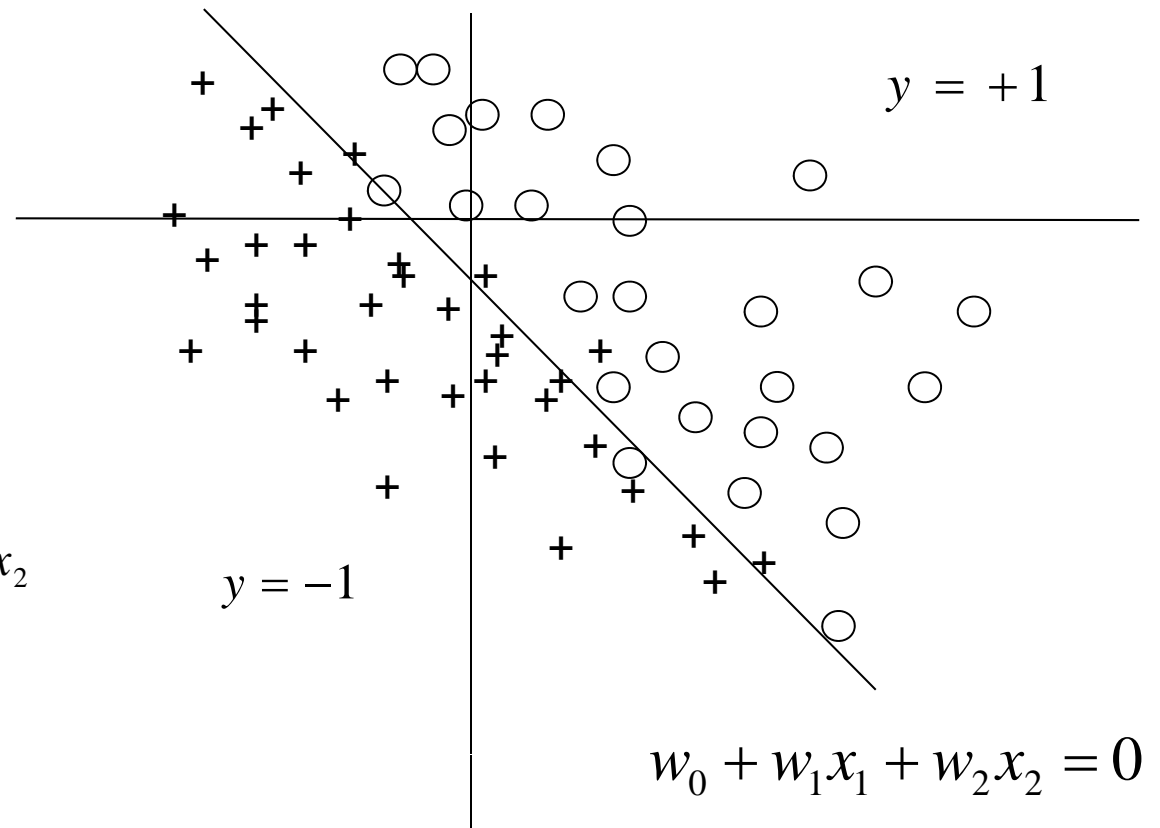
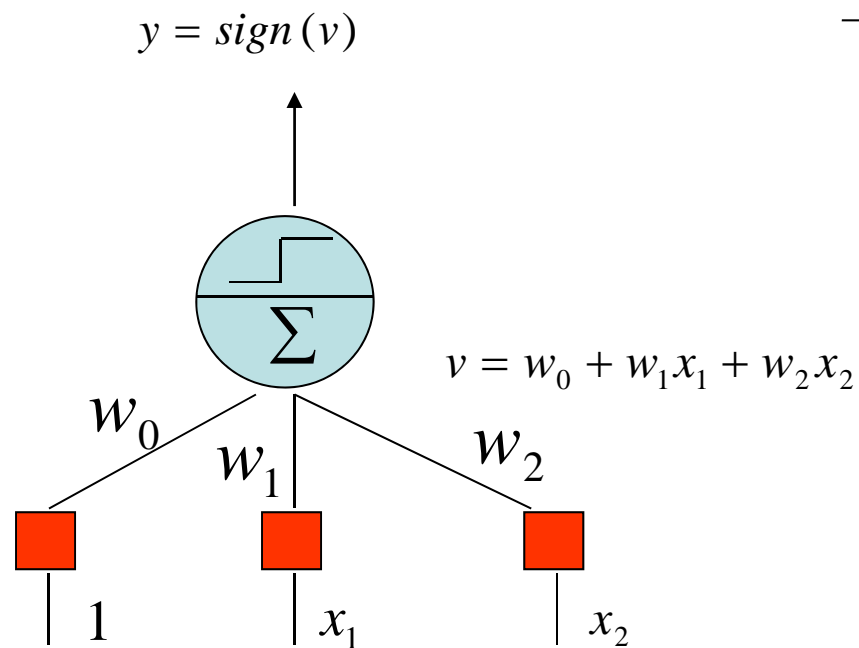
- Rosenblatt (1962)
- Linear separation
- Inputs : Vector of real values
- Outputs : 1 or -1



$$y(x, w) = \text{sgn}(W^T X)$$

# Perceptron

- Defines a plane that linearly separates the feature space.



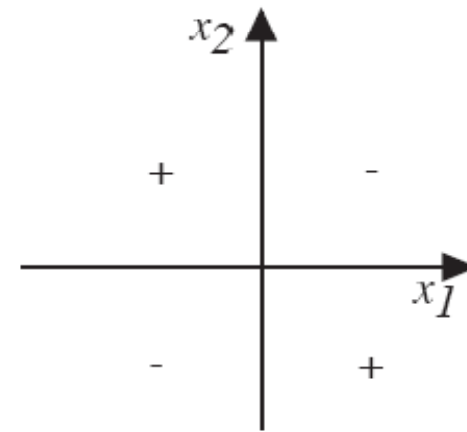
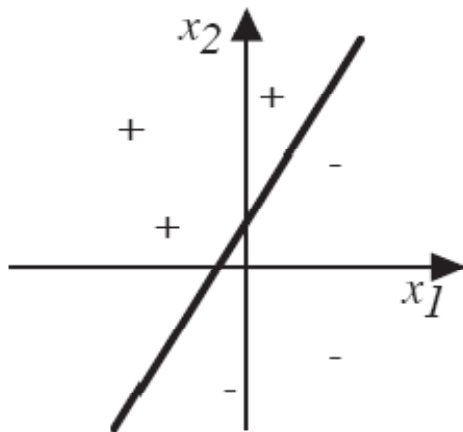
---

# Perceptron

---

- **Decision surface:**

- The decision surface for a single perceptron is a line.
- Can represent some functions (AND( $x_1, x_2$ ) for example).
- Can only work on linearly separable problems (fails on the XOR).
- For non-linearly separable problems perceptrons must be organized in networks.



---

# Perceptron

---

- Error function

- One possible error measure is the number of misclassified examples.
- Suppose  $Y = \{-1, 1\}$  and the estimated classifier  $y(x)$  also outputs a result which is either +1 or -1.
- An example  $\langle X(n), Y(n) \rangle$  is misclassified if  $Y(n) \cdot y(X(n))$  is negative.
- So a reasonable error function is just counting the number of examples correctly classified:

$$E(W) = - \sum_{i \in \text{misclassified}}^n Y(i) y(X(i))$$

- This is called 0-1 loss
- How can we find the best weights  $W$ ?

# Learning (The perceptron rule)

- Minimization of the cost function :

- This is an error counting cost function ->

$$J(w) = \sum_{k \in M} -y_p^k v^k$$

- $J(w)$  is always  $\geq 0$  ( $M$  is the ensemble of bad classified examples)

- $y_p^k$  is the target value

- Partial cost

- If  $x^k$  is not well classified :  $J^k(w) = -y_p^k v^k$
- If  $x^k$  is well classified:  $J^k(w) = 0$

- Partial cost gradient



$$\frac{\partial J^k(w)}{\partial w} = -y_p^k x^k$$

- Perceptron algorithm (updates  $w$  to reduce  $J$ ):

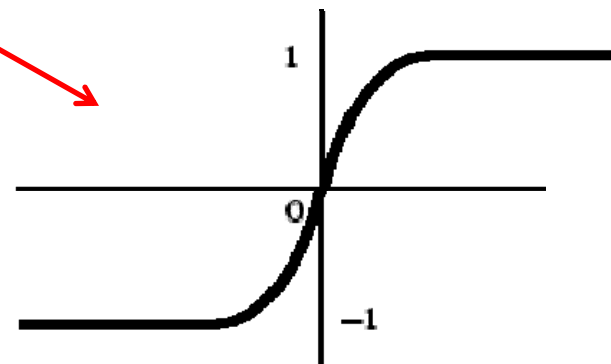
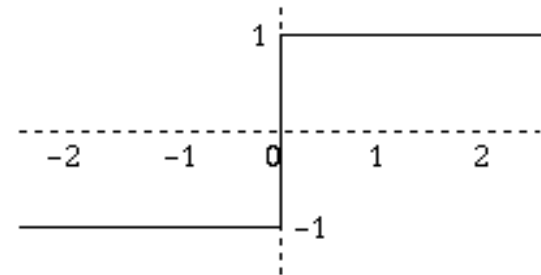
if  $y_p^k v^k > 0$  ( $x^k$  is well classified) :  $w(k) = w(k-1)$

if  $y_p^k v^k < 0$  ( $x^k$  is not well classified) :  $w(k) = w(k-1) + y_p^k x^k$

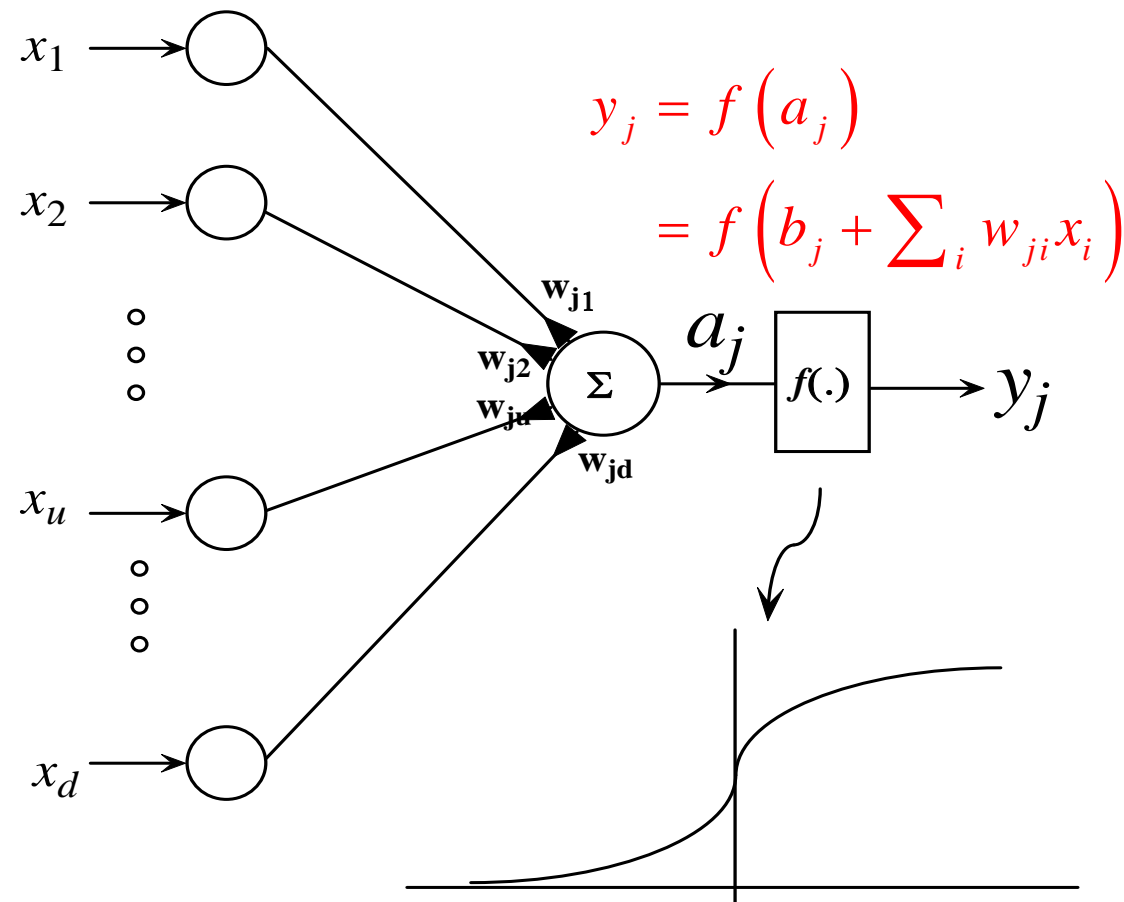
# Perceptron

- **Activation Functions**

- Controls when unit is “active” or “inactive”
- Threshold function (sign) outputs 1 when input is positive and 0 otherwise
- Sigmoid function =  $1 / (1 + e^{-x})$
- Sigmoid function behaves very closely to the threshold function but enable backwards error propagation since it is continuously derivable.

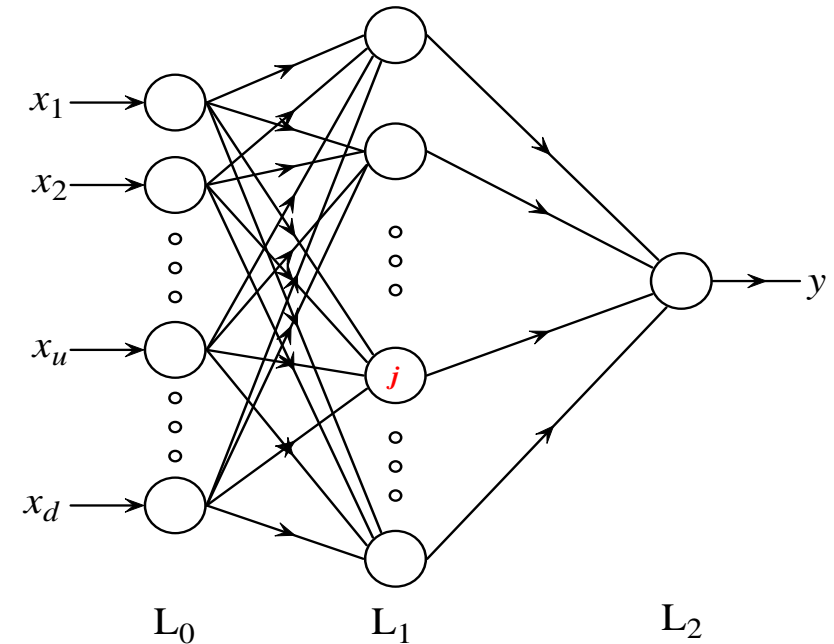


# A Sigmoidal Unit



# Combining multiple Perceptrons

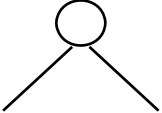
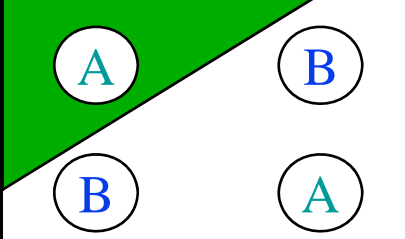
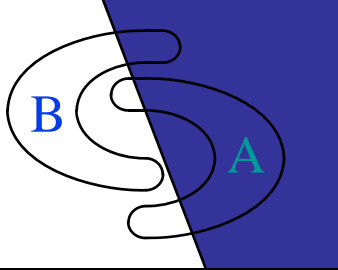
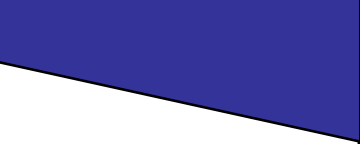
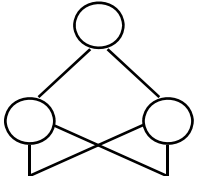
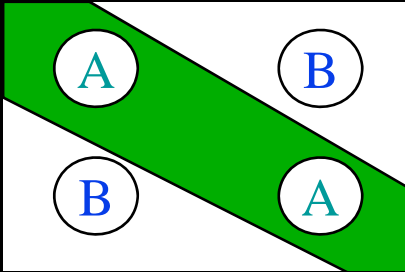
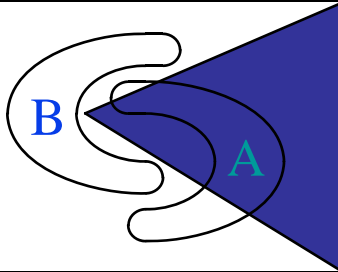
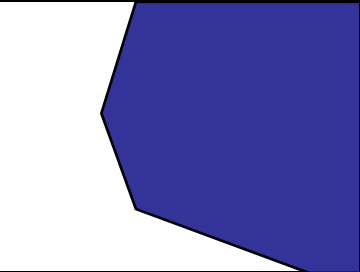
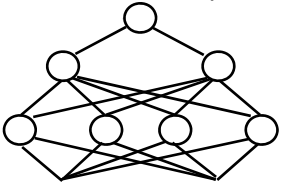
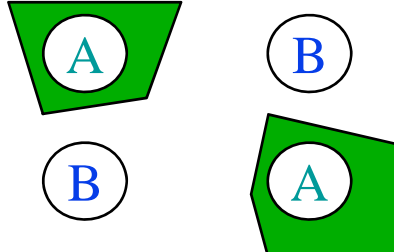
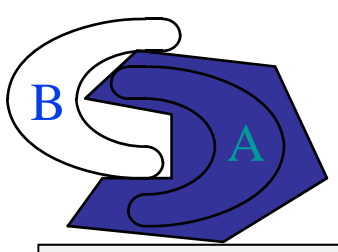
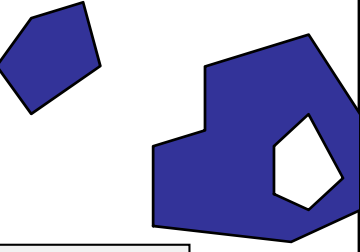
- Adding layers to the MLPs and combining individual perceptrons:
  - To handle more complex problems (than linearly separable ones) we need multiple layers.
  - Each layer receives its inputs from the previous layer and forwards its outputs to the next layer
  - The result is the combination of linear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



$$y = \theta(\underline{w}^T \underline{v} + b)$$
$$v_j = \sigma(\underline{w}_j^T \underline{x} + b_j)$$



# Different non linearly separable problems

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

Neural Networks – An Introduction Dr. Andrew Hunter

---

# Neural Network Learning

---

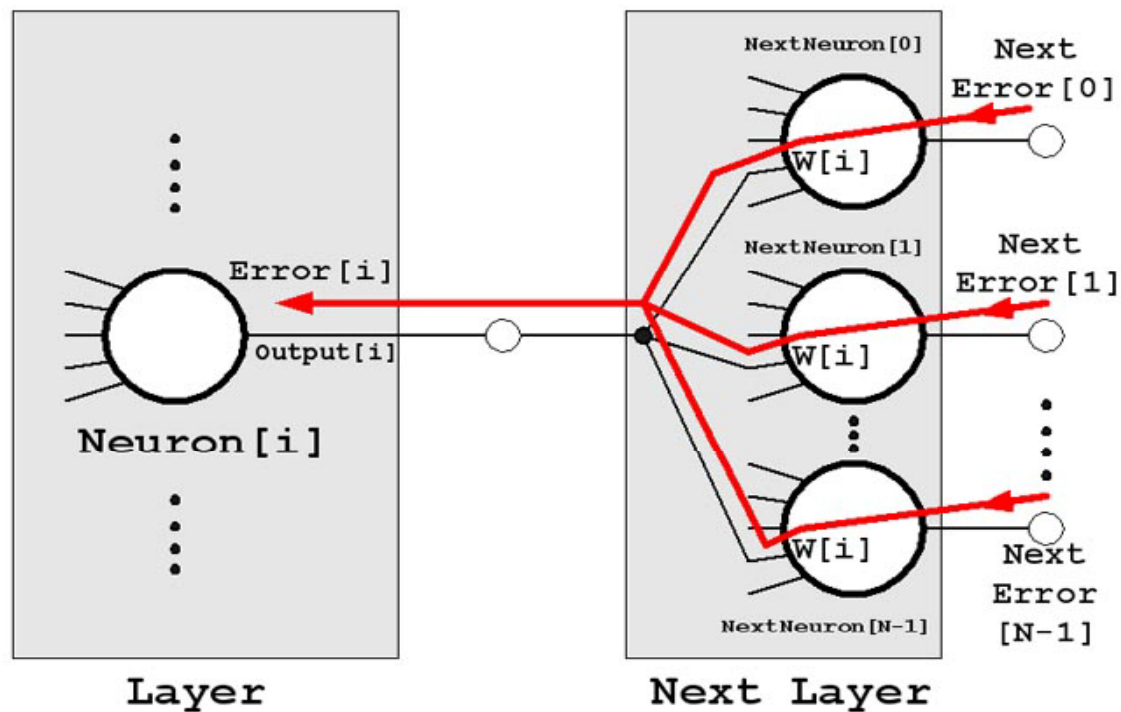
## Back-Propagation Algorithm:

```
function BACK-PROP-LEARNING(examples, network) returns a neural network
  inputs: examples, a set of examples, each with input vector x and output vector y
           network, a multilayer network with  $L$  layers, weights  $W_{j,i}$ , activation function  $g$ 

  repeat
    for each  $e$  in examples do
      for each node  $j$  in the input layer do  $a_j \leftarrow x_j[e]$ 
      for  $l = 2$  to  $M$  do
         $in_i \leftarrow \sum_j W_{j,i} a_j$ 
         $a_i \leftarrow g(in_i)$ 
      for each node  $i$  in the output layer do
         $\Delta_j \leftarrow g'(in_j) \sum_i W_{ji} \Delta_i$ 

        for  $l = M - 1$  to  $1$  do
          for each node  $j$  in layer  $l$  do
             $\Delta_j \leftarrow g'(in_j) \sum_i W_{j,i} \Delta_i$ 
            for each node  $i$  in layer  $l + 1$  do
               $W_{j,i} \leftarrow W_{j,i} + \alpha \times a_j \times \Delta_i$ 
      until some stopping criterion is satisfied
  return NEURAL-NET-HYPOTHESIS(network)
```

# Back-Propagation Illustration

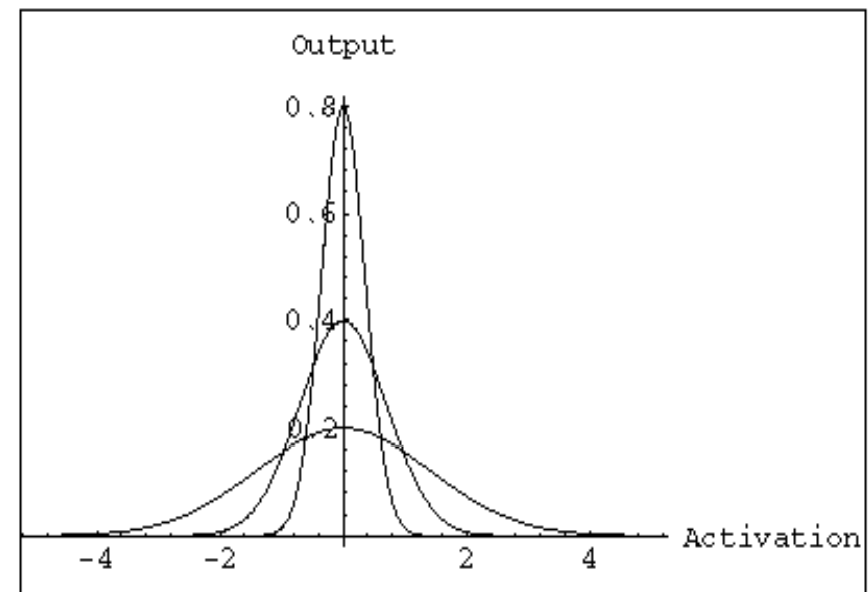
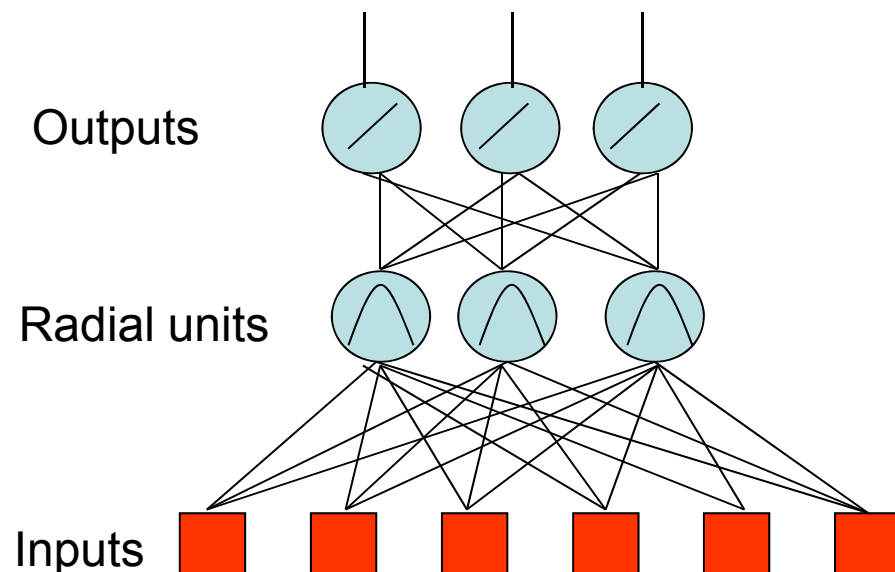


ARTIFICIAL NEURAL NETWORKS Colin Fahey's Guide (Book CD)

# Radial Basis Functions (RBFs)

- **Features**

- One hidden layer
- The activation of a hidden unit is determined by the distance between the input vector and a prototype vector



---

# Radial Basis Functions (RBFs)

---

- RBF hidden layer units have a receptive field which has a centre
- Generally, the hidden unit function is Gaussian
- The output Layer is linear
- Realized function

$$s(x) = \sum_{j=1}^K W_j \Phi(\|x - c_j\|)$$

$$\Phi(\|x - c_j\|) = \exp - \left( \frac{\|x - c_j\|}{\sigma_j} \right)^2$$

---

# Radial Basis Functions (RBFs)

---

- **Learning**
  - The training is performed by deciding on
    - How many hidden nodes there should be
    - The centers and the sharpness of the Gaussians
  - 2 steps
    - In the 1st stage, the input data set is used to determine the parameters of the basis functions
    - In the 2nd stage, functions are kept fixed while the second layer weights are estimated ( Simple BP algorithm like for MLPs)

# MLPs versus RBFs

- **Classification**

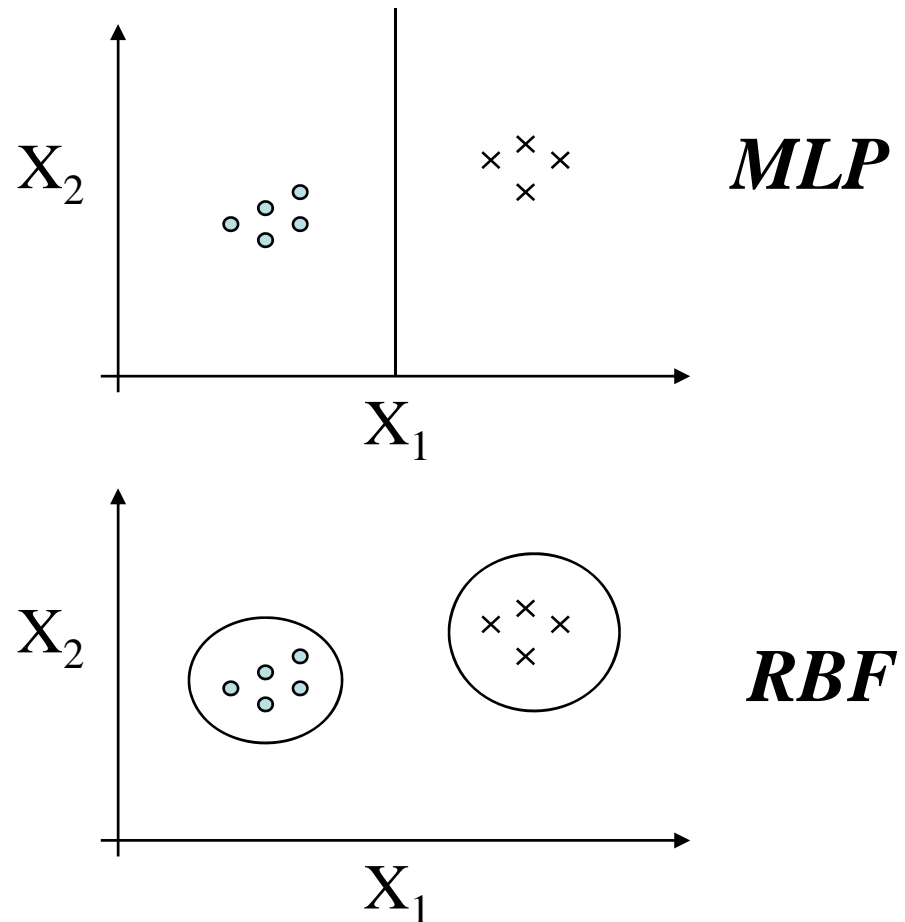
- MLPs separate classes via hyperplanes
- RBFs separate classes via hyperspheres

- **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

---

- Decision surface is a hyperplane (line in 2D) in feature space (similar to the Perceptron)
- Arguably, the most important recent discovery in machine learning
- 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

**Slides from:** *Constantin F. Aliferis & Ioannis Tsamardinos*



---

# Support Vector Machines

---

- Three main ideas:
  1. Define what an optimal hyperplane is (in way that can be identified in a computationally efficient way): maximize margin
  2. Extend the above definition for non-linearly separable problems: have a penalty term for misclassifications
  3. Map data to high dimensional space where it is easier to classify with linear decision surfaces: reformulate problem so that data is mapped implicitly to this space

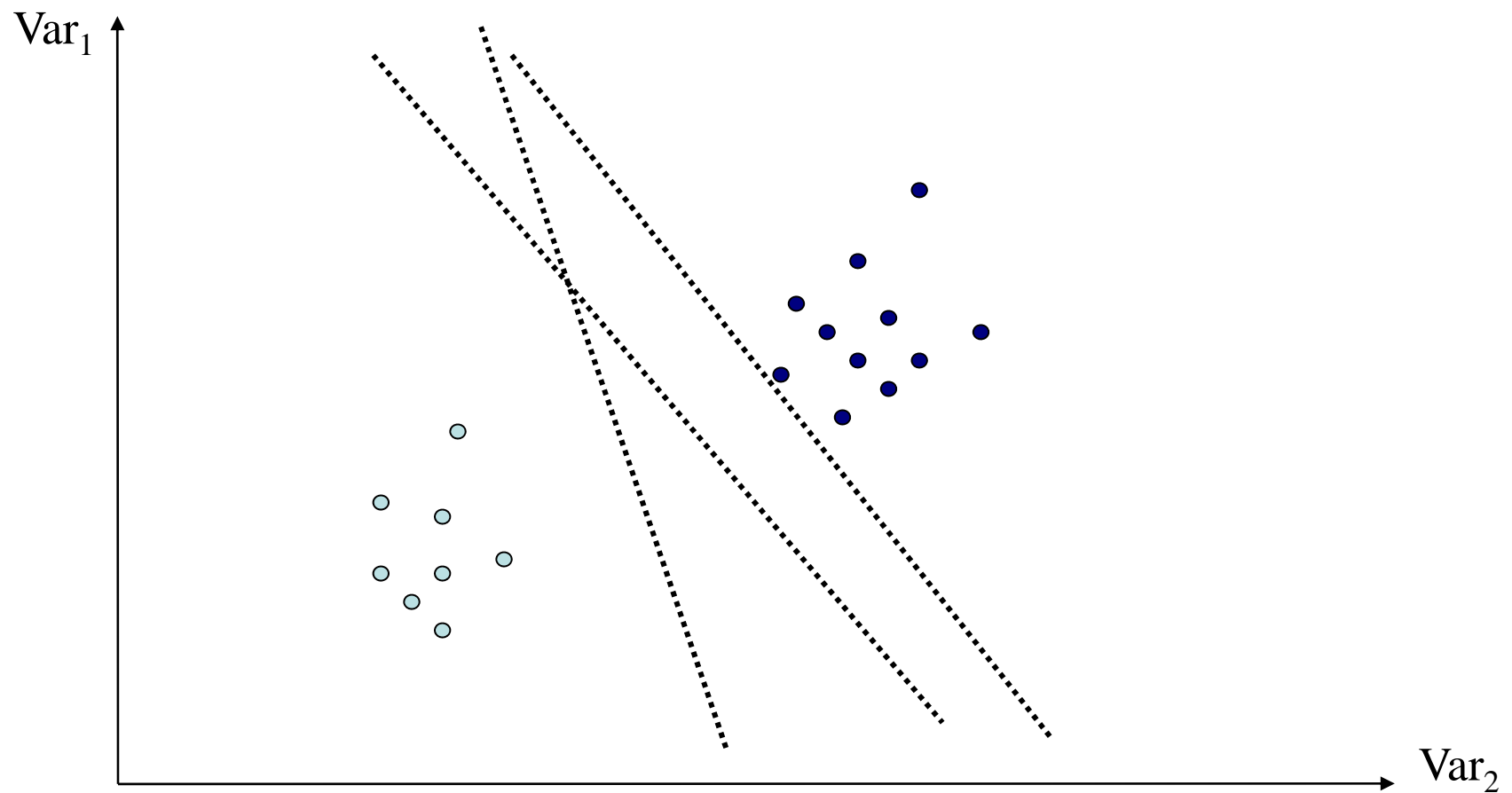
---

# Support Vector Machines

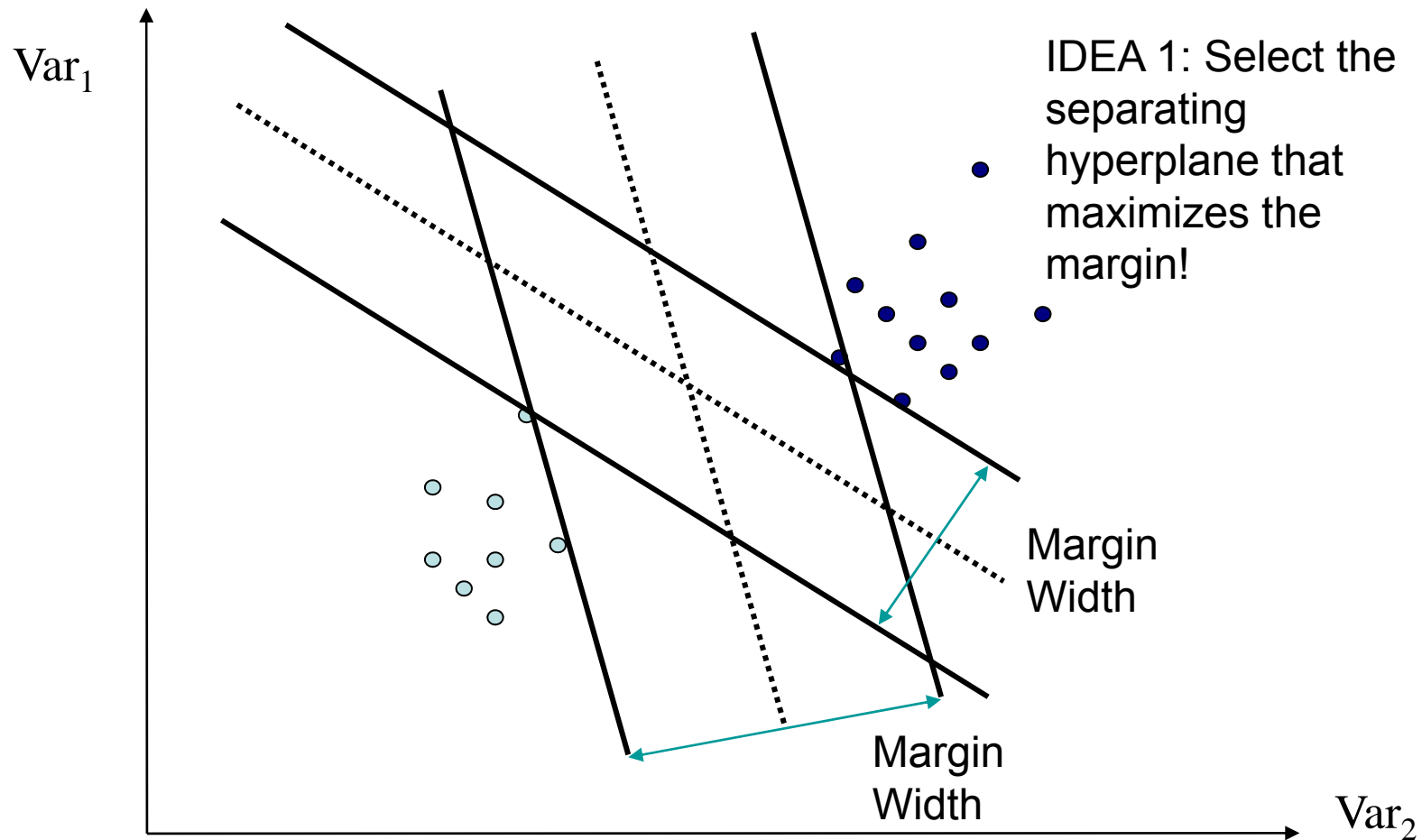
---

- Three main ideas:
  1. Define what an optimal hyperplane is (in way that can be identified in a computationally efficient way): maximize margin
  2. Extend the above definition for non-linearly separable problems: have a penalty term for misclassifications
  3. Map data to high dimensional space where it is easier to classify with linear decision surfaces: reformulate problem so that data is mapped implicitly to this space

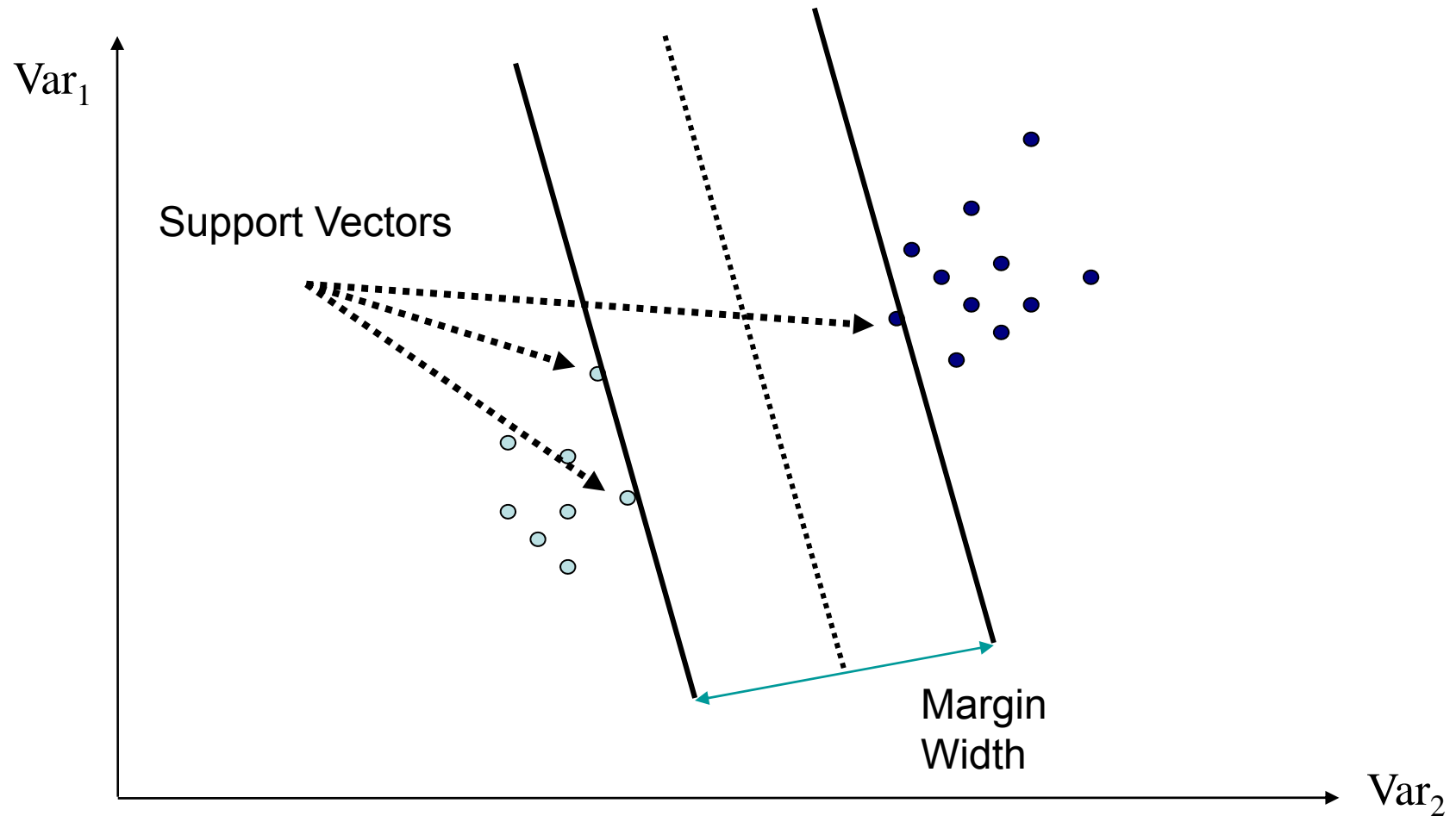
# Which Separating Hyperplane to Use?



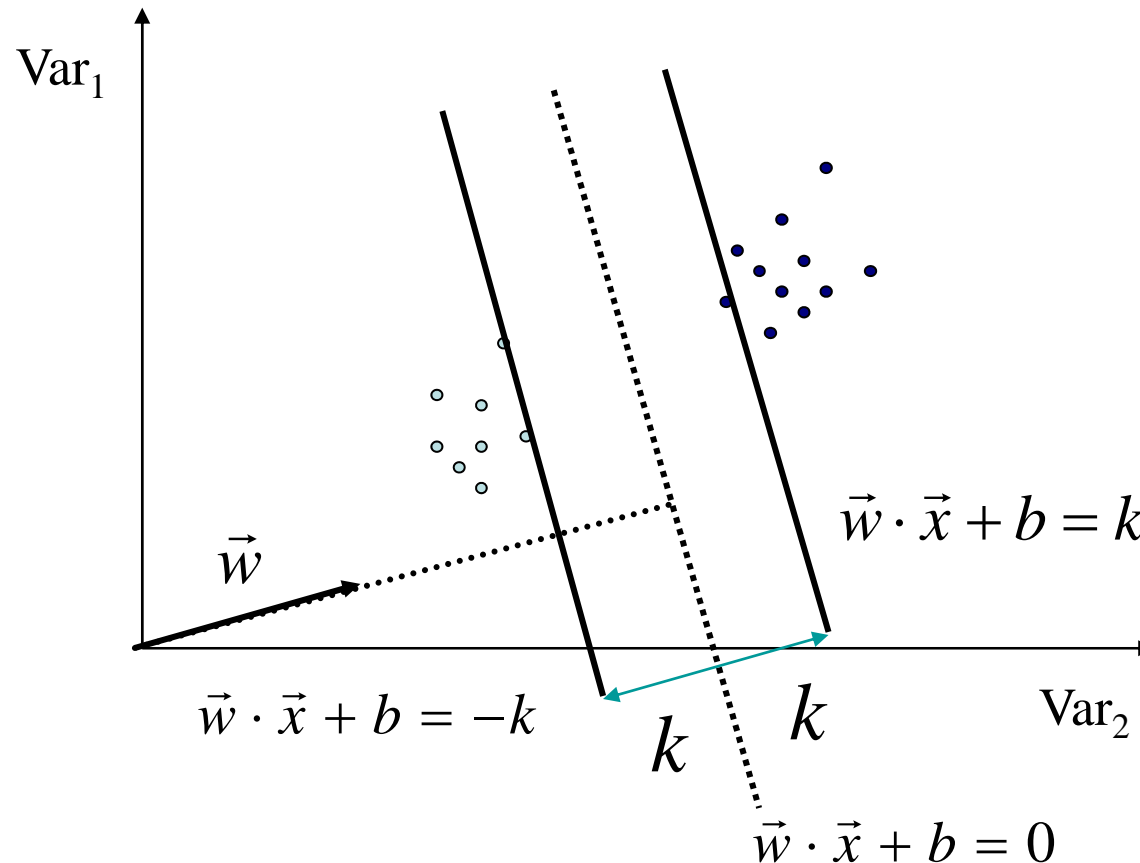
# Maximizing the Margin



# Support Vectors



# Setting Up the Optimization Problem



The width of the margin is:

$$\frac{2|k|}{\|\vec{w}\|}$$

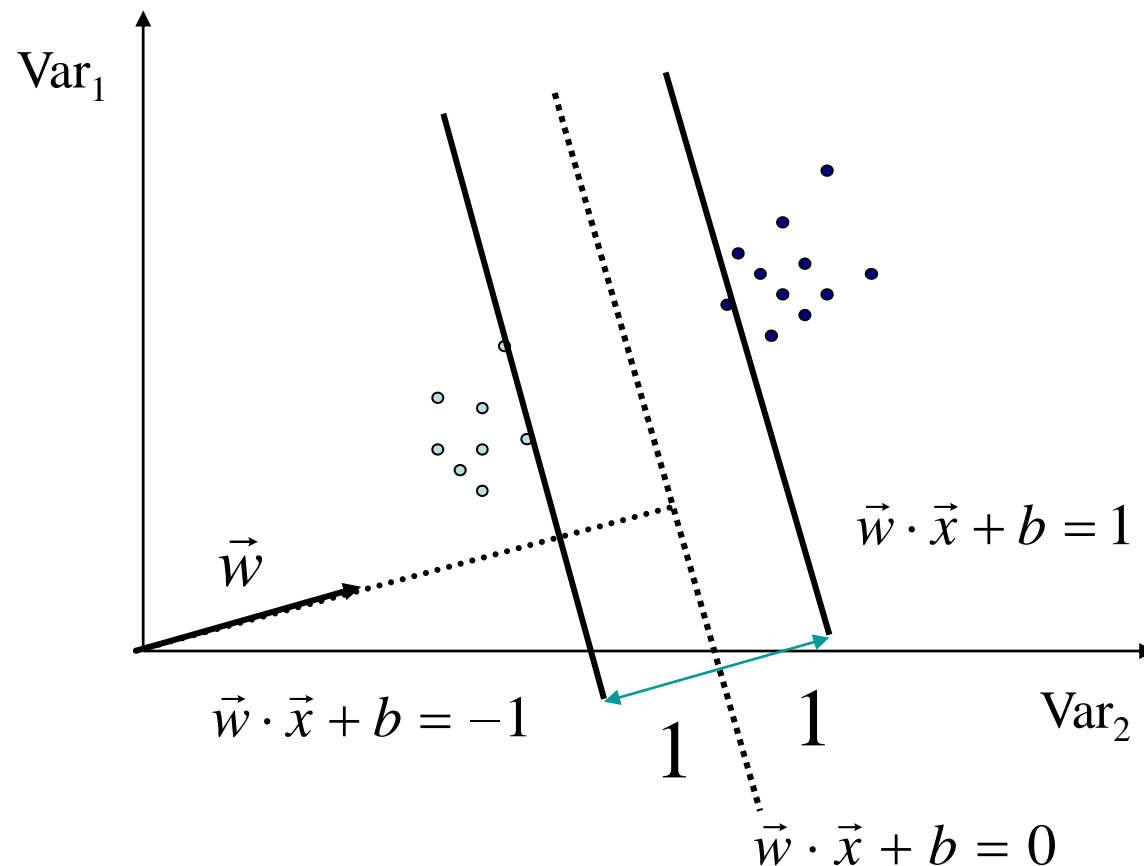
So, the problem is:

$$\max \frac{2|k|}{\|\vec{w}\|}$$

*s.t.*  $(\vec{w} \cdot \vec{x} + b) \geq k, \forall x \text{ of class 1}$

$(\vec{w} \cdot \vec{x} + b) \leq -k, \forall x \text{ of class 2}$

# Setting Up the Optimization Problem



There is a scale and unit for data so that  $k=1$ . Then problem becomes:

$$\begin{aligned} \max \quad & \frac{2}{\|\vec{w}\|} \\ \text{s.t.} \quad & (\vec{w} \cdot \vec{x} + b) \geq 1, \quad \forall x \text{ of class 1} \\ & (\vec{w} \cdot \vec{x} + b) \leq -1, \quad \forall x \text{ of class 2} \end{aligned}$$

---

# Setting Up the Optimization Problem

---

- If class 1 corresponds to 1 and class 2 corresponds to -1, we can rewrite

$$(w \cdot x_i + b) \geq 1, \quad \forall x_i \text{ with } y_i = 1$$

$$(w \cdot x_i + b) \leq -1, \quad \forall x_i \text{ with } y_i = -1$$

- as

$$y_i (w \cdot x_i + b) \geq 1, \quad \forall x_i$$

- So the problem becomes:

$$\max \frac{2}{\|w\|}$$

$$s.t. \ y_i (w \cdot x_i + b) \geq 1, \quad \forall x_i$$

or

$$\min \frac{1}{2} \|w\|^2$$

$$s.t. \ y_i (w \cdot x_i + b) \geq 1, \quad \forall x_i$$



---

# Linear, Hard-Margin SVM Formulation

---

- Find  $w, b$  that solves

$$\min \frac{1}{2} \|w\|^2$$

$$s.t. y_i(w \cdot x_i + b) \geq 1, \quad \forall x_i$$

- Problem is convex so, there is a unique global minimum value (when feasible)
- There is also a unique minimizer, i.e. weight and  $b$  value that provides the minimum
- Non-solvable if the data is not linearly separable
- Quadratic Programming
  - Very efficient computationally with modern constraint optimization engines (handles thousands of constraints and training instances).

# Max Margin Loss Function

Primal

$$L(\vec{w}, b) = \frac{1}{2} \vec{w} \cdot \vec{w} - \sum_{i=0}^{N-1} \alpha_i [t_i ((\vec{w} \cdot \vec{x}_i) + b) - 1]$$

$$\vec{w} = \sum_{i=0}^{N-1} \alpha_i t_i \vec{x}_i$$

Dual

$$W(\alpha) = \sum_{i=0}^{N-1} \alpha_i - \frac{1}{2} \sum_{i,j=0}^{N-1} \alpha_i \alpha_j t_i t_j (\vec{x}_i \cdot \vec{x}_j)$$

where  $\alpha_i \geq 0$

$$\sum_{i=0}^{N-1} \alpha_i t_i = 0$$

---

# Support Vector Expansion

---

New decision Function

$$\begin{aligned} D(\vec{x}) &= \text{sign}(\vec{w}^T \vec{x} + b) \\ &= \text{sign} \left( \left[ \sum_{i=0}^{N-1} \alpha_i t_i \vec{x}_i \right]^T \vec{x} + b \right) \\ &= \text{sign} \left( \left[ \sum_{i=0}^{N-1} \alpha_i t_i (\vec{x}_i^T \vec{x}) \right] + b \right) \end{aligned}$$

$$\vec{w} = \sum_{i=0}^{N-1} \alpha_i t_i \vec{x}_i$$

- When  $\alpha_i$  is non-zero then  $x_i$  is a support vector
- When  $\alpha_i$  is zero  $x_i$  is not a support vector

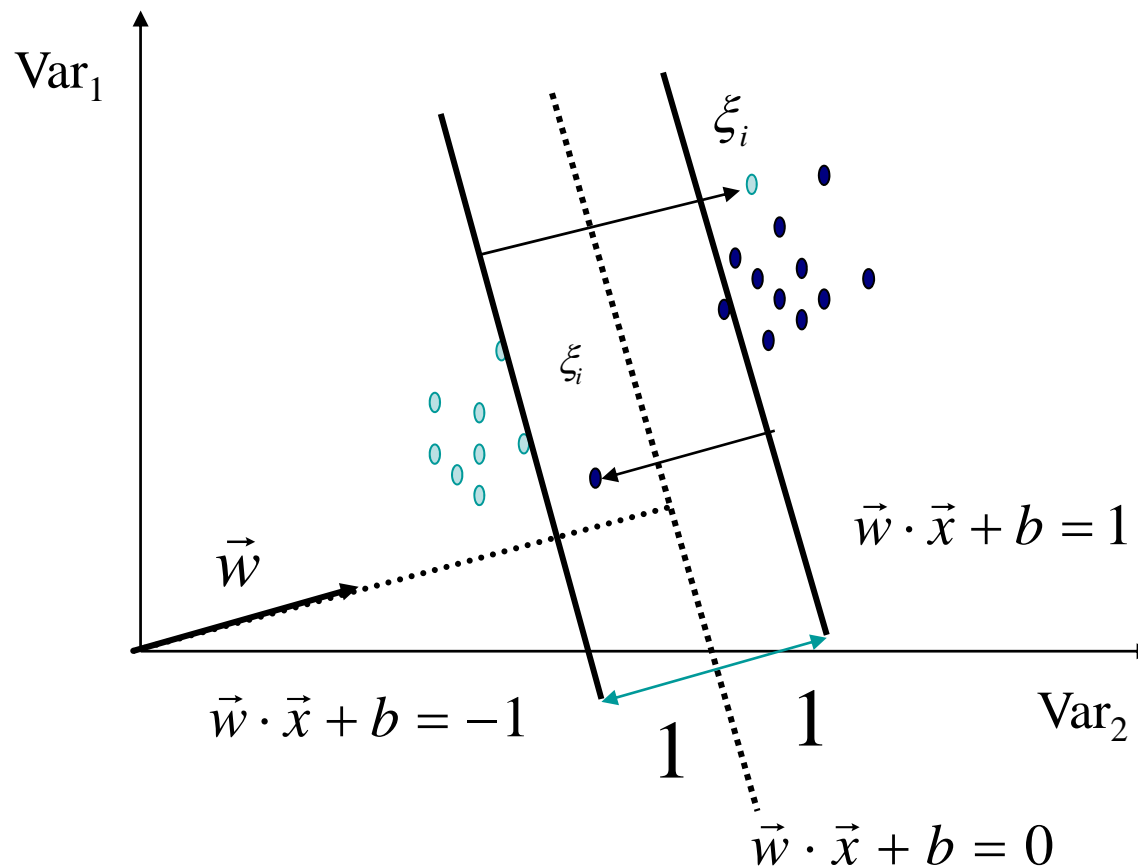
---

# Support Vector Machines

---

- Three main ideas:
  1. Define what an optimal hyperplane is (in way that can be identified in a computationally efficient way): maximize margin
  2. Extend the above definition for non-linearly separable problems: have a penalty term for misclassifications
  3. Map data to high dimensional space where it is easier to classify with linear decision surfaces: reformulate problem so that data is mapped implicitly to this space

# Non-Linearly Separable Data

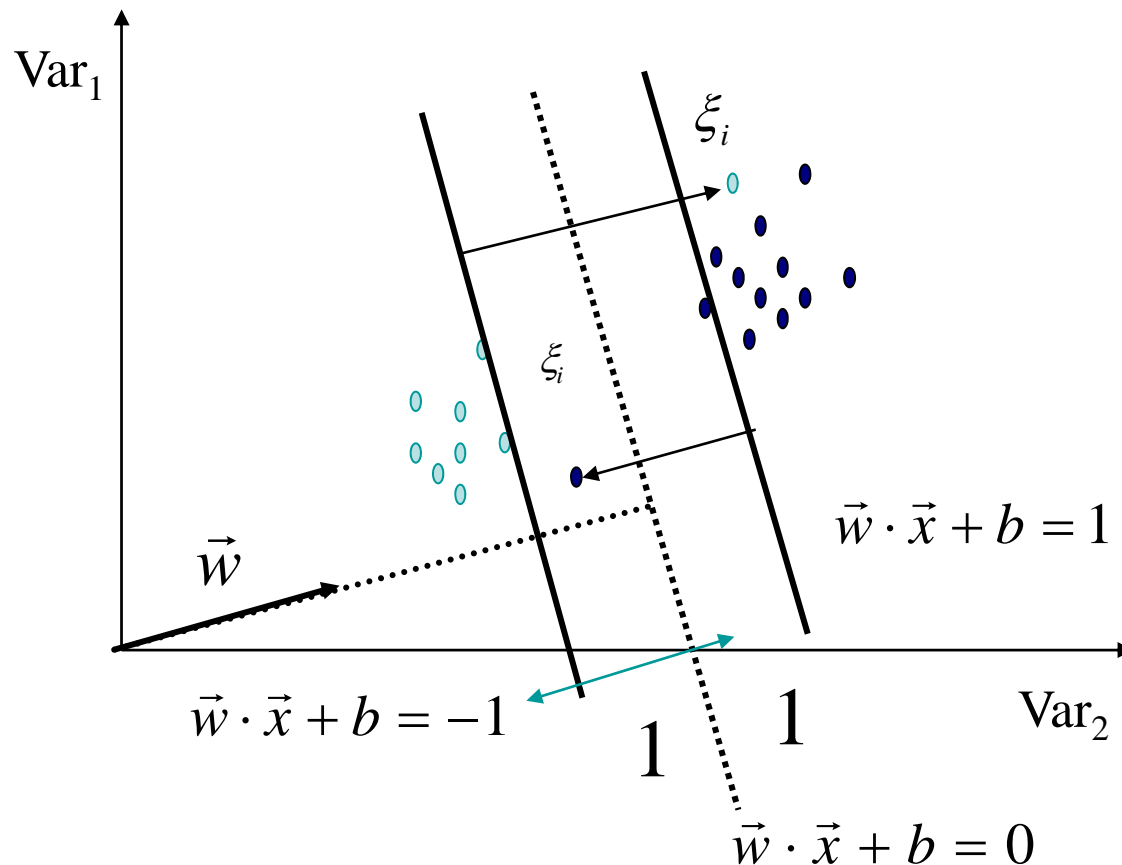


Introduce slack variables

 $\xi_i$ 

Allow some instances to fall within the margin, but penalize them

# Non-Linearly Separable Data



Constraint becomes :

$$y_i(w \cdot x_i + b) \geq 1 - \xi_i, \quad \forall x_i$$

$$\xi_i \geq 0$$

Objective function penalizes for misclassified instances and those within the margin

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

C trades-off margin width and misclassifications

---

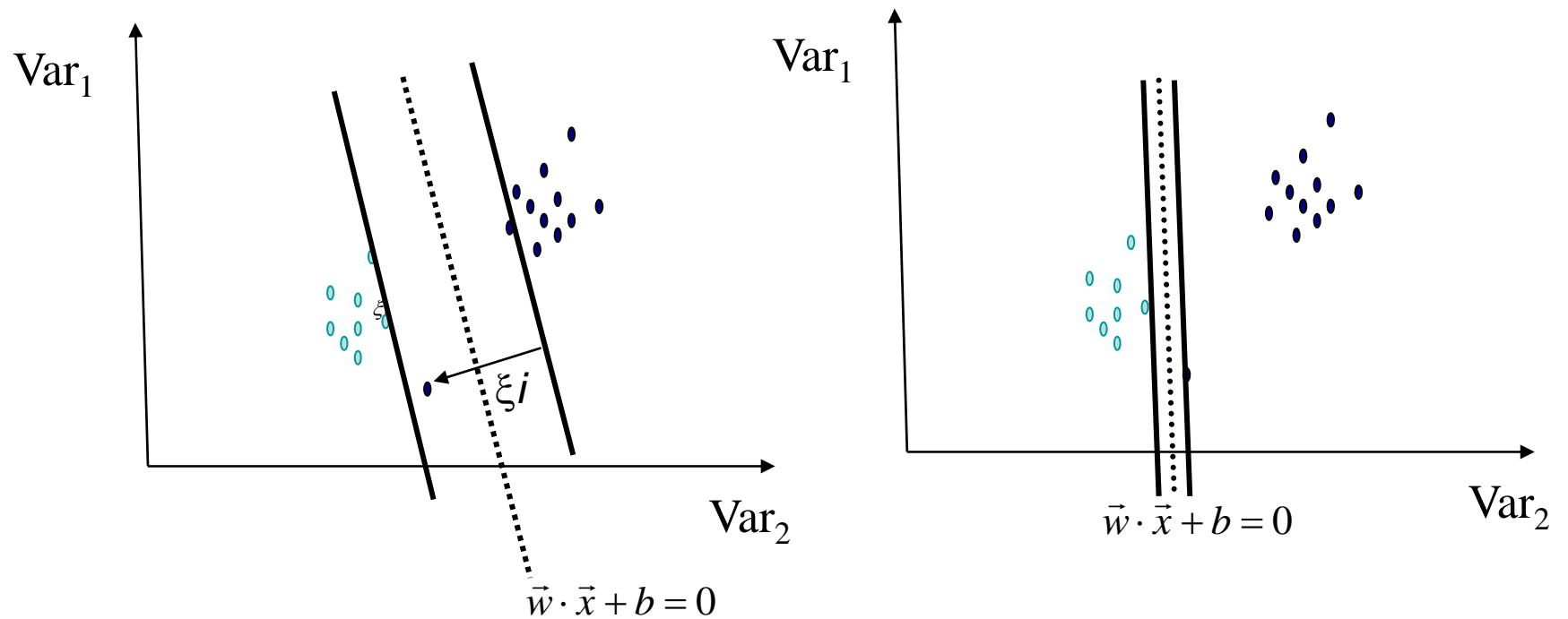
# Linear, Soft-Margin SVMs

---

$$\min \frac{1}{2} \|w\|^2 + C \sum_i \xi_i \quad \begin{array}{l} y_i (w \cdot x_i + b) \geq 1 - \xi_i, \quad \forall x_i \\ \xi_i \geq 0 \end{array}$$

- Algorithm tries to maintain  $\xi_i$  to zero while maximizing margin
- Notice: algorithm does not minimize the *number* of misclassifications (NP-complete problem) but the sum of distances from the margin hyperplanes
- Other formulations use  $\xi_i^2$  instead
- As  $C \rightarrow \infty$ , we get closer to the hard-margin solution

# Robustness of Soft vs Hard Margin SVMs



Soft Margin SVN

Hard Margin SVN



---

# Soft vs Hard Margin SVM

---

- Soft-Margin always have 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)

---

# Support Vector Machines

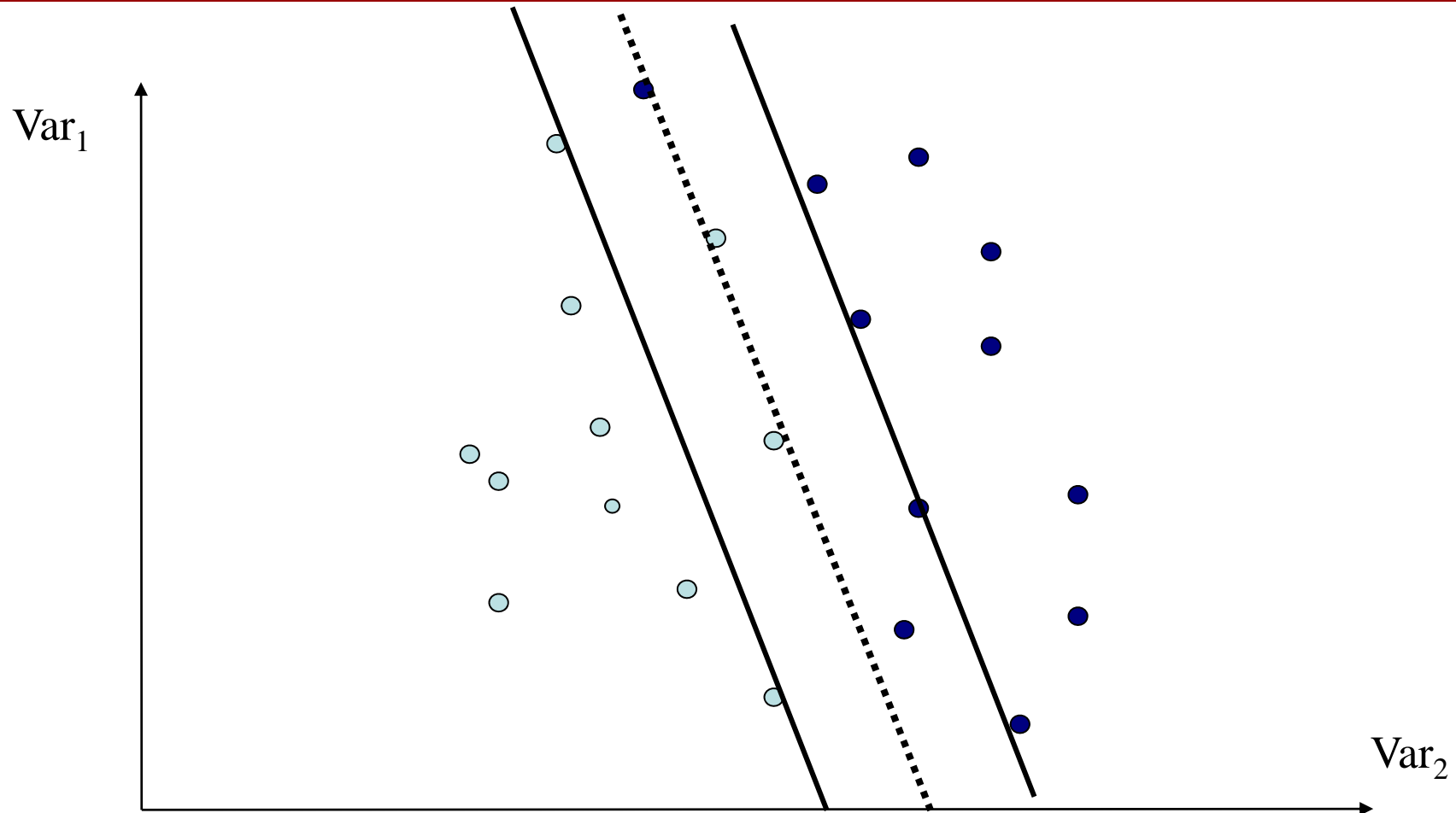
---

- Three main ideas:
  1. Define what an optimal hyperplane is (in way that can be identified in a computationally efficient way): maximize margin
  2. Extend the above definition for non-linearly separable problems: have a penalty term for misclassifications
  3. Map data to high dimensional space where it is easier to classify with linear decision surfaces: reformulate problem so that data is mapped implicitly to this space

---

# Disadvantages of Linear Decision Surfaces

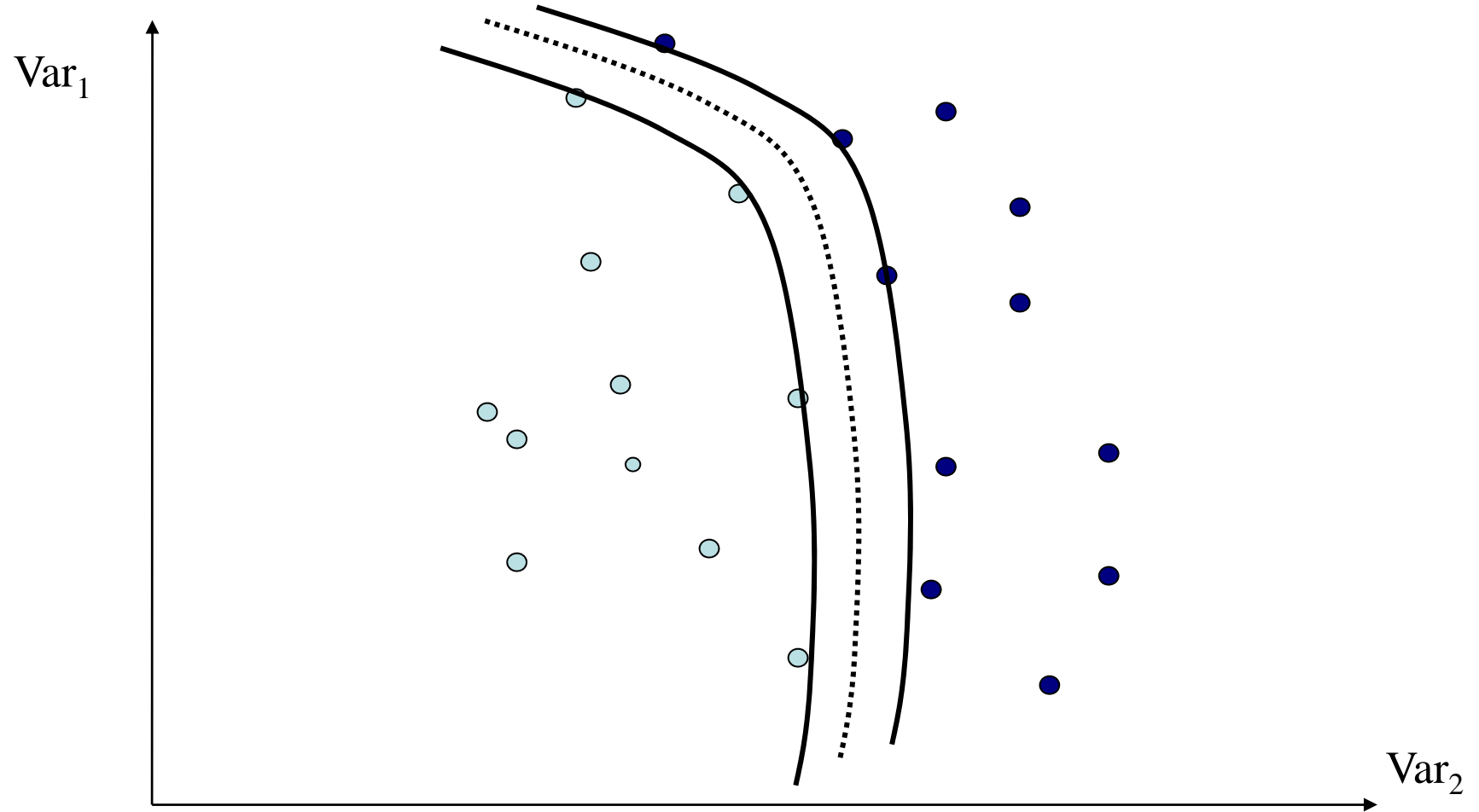
---



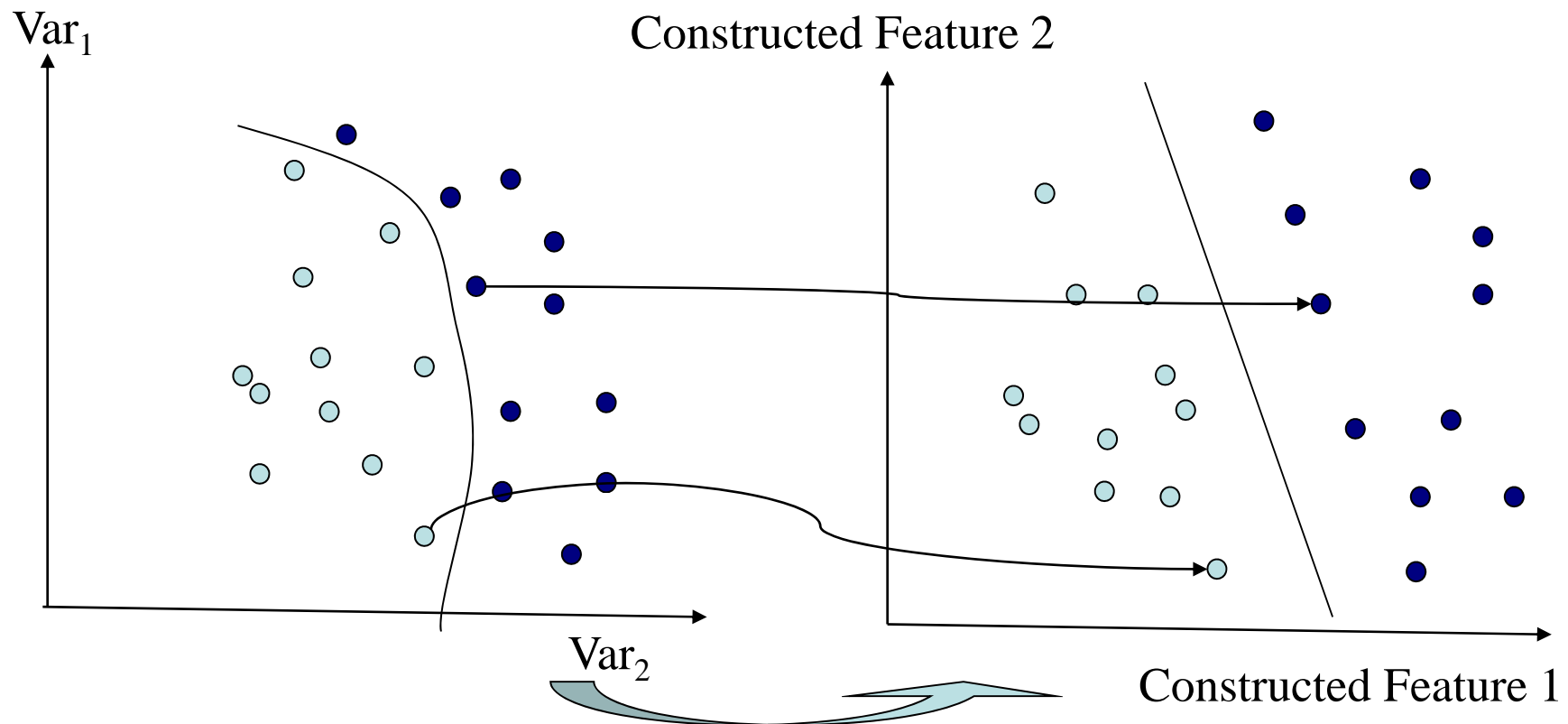
---

# Advantages of Non-Linear Surfaces

---



# Linear Classifiers in High-Dimensional Spaces



Find function  $\Phi(x)$  to map to a different space

---

# Mapping Data to a High-Dimensional Space

---

- Find function  $\Phi(x)$  to map to a different space, then SVM formulation becomes:

$$\min \frac{1}{2} \|w\|^2 + C \sum_i \xi_i \quad s.t. \quad y_i (w \cdot \Phi(x) + b) \geq 1 - \xi_i, \forall x_i \\ \xi_i \geq 0$$

- Data appear as  $\Phi(x)$ , weights  $w$  are now weights in the new space
- Explicit mapping expensive if  $\Phi(x)$  is very high dimensional
- Solving the problem without explicitly mapping the data is desirable

# The Dual of the SVM Formulation

- Original SVM formulation
  - $n$  inequality constraints
  - $n$  positivity constraints
  - $n$  number of  $\xi$  variables
- The (Wolfe) dual of this problem
  - one equality constraint
  - $n$  positivity constraints
  - $n$  number of  $\alpha$  variables (Lagrange multipliers)
  - Objective function more complicated
- NOTICE: Data only appear as  $\Phi(x_i) \cdot \Phi(x_j)$

$$\min_{w,b} \frac{1}{2} \|w\|^2 + C \sum_i \xi_i$$

$$s.t. \quad y_i (w \cdot \Phi(x) + b) \geq 1 - \xi_i, \forall x_i \\ \xi_i \geq 0$$

$$\min_{\alpha_i} \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j (\Phi(x_i) \cdot \Phi(x_j)) - \sum_i \alpha_i$$

$$s.t. \quad C \geq \alpha_i \geq 0, \forall x_i$$

$$\sum_i \alpha_i y_i = 0$$

---

# Incorporating Kernels in SVMs

---

$$W(\alpha) = \sum_{i=0}^{N-1} \alpha_i - \frac{1}{2} \sum_{i,j=0}^{N-1} l_i l_j \alpha_i \alpha_j (\phi(x_i) \cdot \phi(x_j))$$
$$W(\alpha) = \sum_{i=0}^{N-1} \alpha_i - \frac{1}{2} \sum_{i,j=0}^{N-1} l_i l_j \alpha_i \alpha_j K(x_i, x_j)$$

- Optimize  $\alpha_i$ 's and bias w.r.t. kernel
- Decision function:

$$D(\vec{x}) = \text{sign} \left( \left[ \sum_{i=0}^{N-1} \alpha_i l_i (\vec{x}_i^T \vec{x}) \right] + b \right)$$
$$D(\vec{x}) = \text{sign} \left( \left[ \sum_{i=0}^{N-1} \alpha_i l_i K(\vec{x}_i, \vec{x}) \right] + b \right)$$



---

# Polynomial Kernels

---

$$K(\vec{x}, \vec{z}) = (\vec{x}^T \vec{z} + c)^d$$

where  $c \geq 0$

- The dot product is related to a polynomial power of the original dot product.
- if  $c$  is large then focus on linear terms
- if  $c$  is small focus on higher order terms
- Very fast to calculate

---

# Radial Basis Functions

---

$$K(\vec{x}, \vec{z}) = \exp \left\{ \frac{\|\vec{x} - \vec{z}\|^2}{2\sigma^2} \right\}$$

- The inner product of two points is related to the distance in space between the two points.
- Placing a bump on each point.

