# Statistical Pattern Recognition



Ricardo Gutierrez-Osuna
Wright State University

# Outline

- Introduction to Pattern Recognition
- Dimensionality reduction
- Classification
- Validation
- Clustering



# SECTION I: Introduction

- Features, patterns and classifiers
- Components of a PR system
- An example
- Probability definitions
- Bayes Theorem
- Gaussian densities



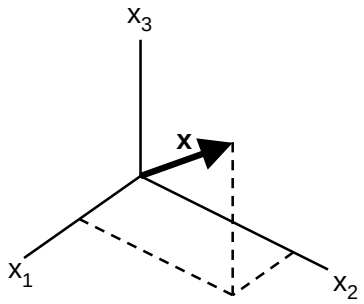
# Features, patterns and classifiers

## Feature

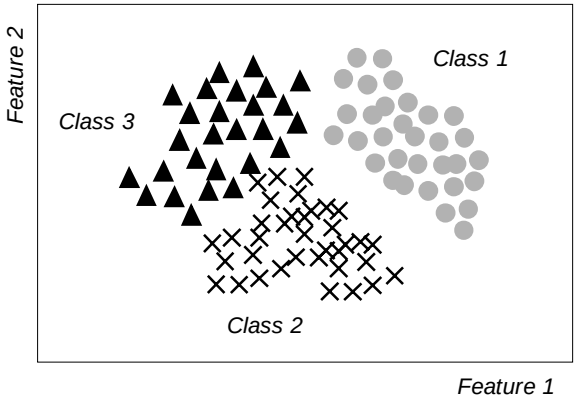
- Feature is any distinctive aspect, quality or characteristic
  - Features may be symbolic (i.e., color) or numeric (i.e., height)
- The combination of *d* features is represented as a *d*-dimensional column vector called a **feature vector**
  - The d-dimensional space defined by the feature vector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

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

**Feature vector**




**Feature space (3D)**

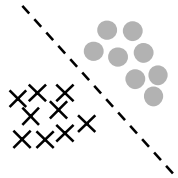



**Scatter plot (2D)**



# Features, patterns and classifiers

- **Pattern**
  - Pattern is a composite of traits or features characteristic of an individual
  - In classification, a pattern is a pair of variables $\{x, \omega\}$ where
    - $\boldsymbol{x}$ is a collection of observations or features (feature vector)
    - $\omega$ is the concept behind the observation (label)
- **What makes a "good" feature vector?**
  - The quality of a feature vector is related to its ability to discriminate examples from different classes
    - Examples from the same class should have similar feature values
    - Examples from different classes have different feature values

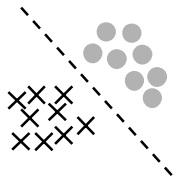


***"Good" features***

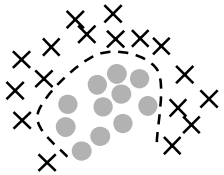
***"Bad" features***

# Features, patterns and classifiers

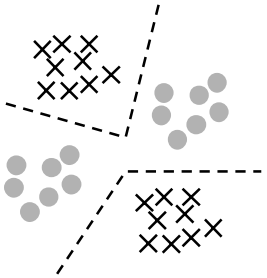
## More feature properties



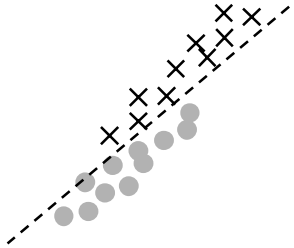
*Linear separability*



*Non-linear separability*



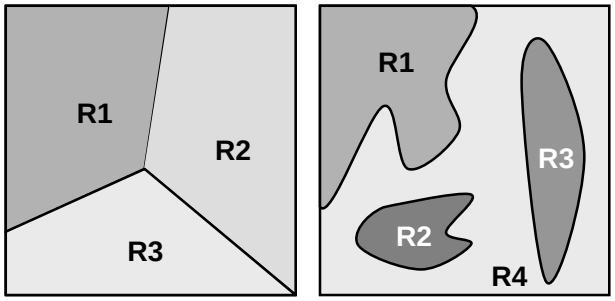
*Multi-modal*



*Highly correlated features*

## Classifiers

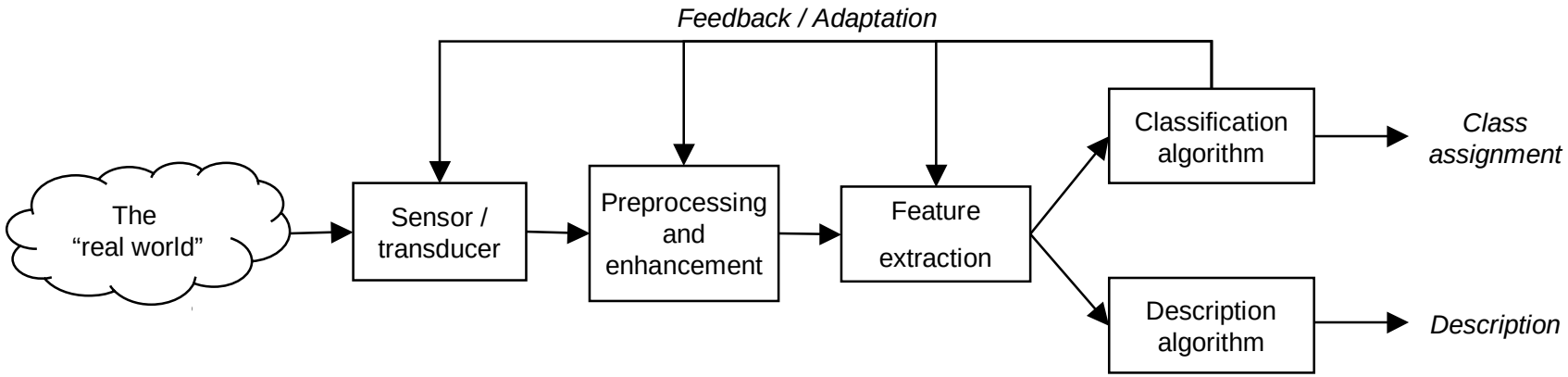
- The goal of a classifier is to partition feature space into class-labeled **decision regions**
- Borders between decision regions are called **decision boundaries**






# Components of a pattern recognition system

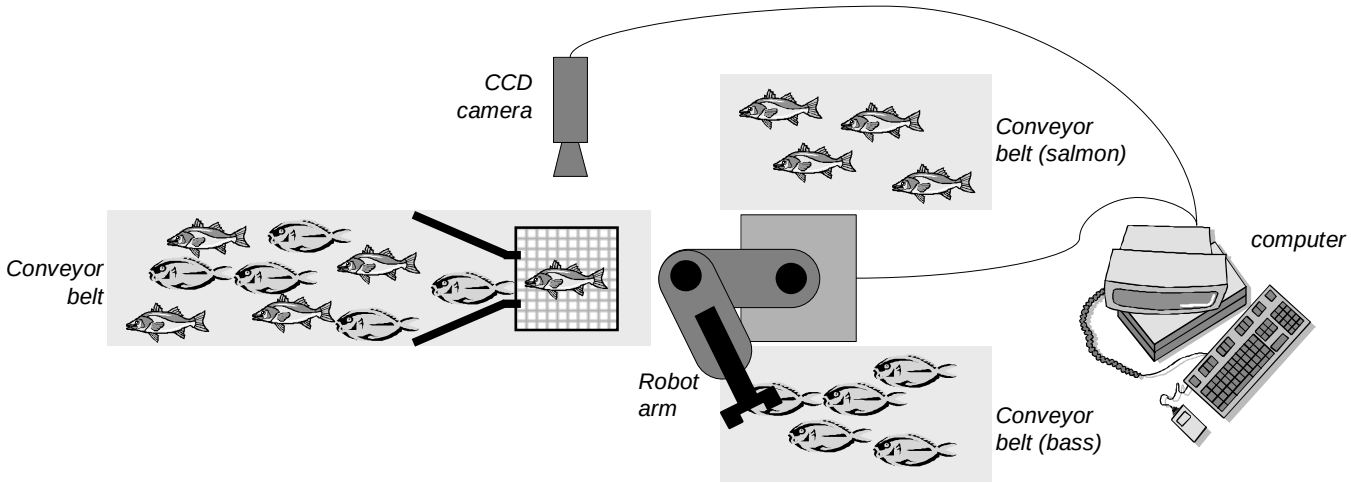
- **A typical pattern recognition system contains**
  - A sensor
  - A preprocessing mechanism
  - A feature extraction mechanism (manual or automated)
  - A classification or description algorithm
  - A set of examples (training set) already classified or described




[Duda, Hart and Stork, 2001]

# An example

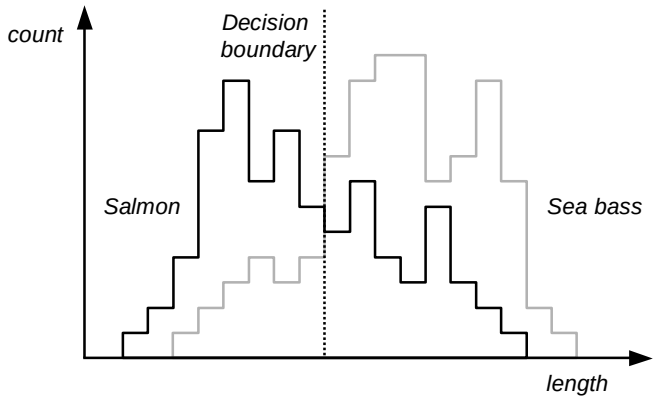
- **Consider the following scenario***
  - A fish processing plan wants to automate the process of sorting incoming fish according to species (salmon or sea bass)
  - The automation system consists of
    - a conveyor belt for incoming products
    - two conveyor belts for sorted products
    - a pick-and-place robotic arm
    - a vision system with an overhead CCD camera
    - a computer to analyze images and control the robot arm




***Adapted from [Duda, Hart and Stork, 2001]**

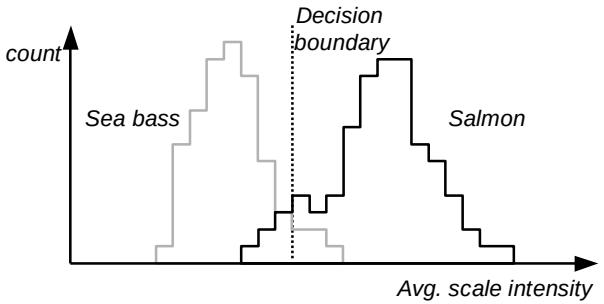
# An example

- **Sensor**
  - The camera captures an image as a new fish enters the sorting area
- **Preprocessing**
  - Adjustments for average intensity levels
  - Segmentation to separate fish from background
- **Feature Extraction**
  - Suppose we know that, on the average, sea bass is larger than salmon
- **Classification**
  - Collect a set of examples from both species
    - Plot a distribution of lengths for both classes
  - Determine a decision boundary (threshold) that minimizes the classification error
    - We estimate the system's probability of error and obtain a discouraging result of 40%
  - **What is next?**

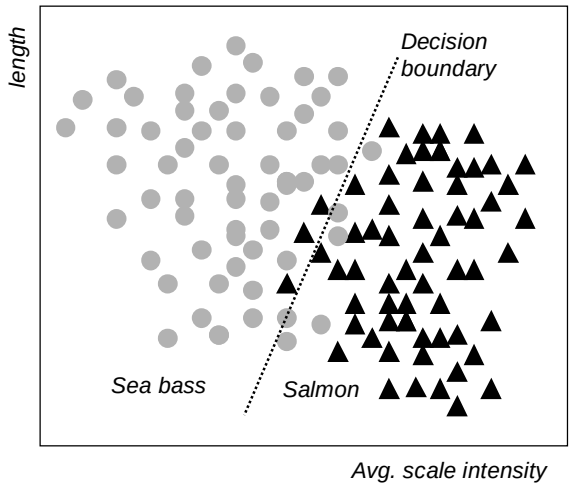


# An example

- **Improving the performance of our PR system**
  - Committed to achieve a recognition rate of 95%, we try a number of features
    - Width, Area, Position of the eyes w.r.t. mouth...
    - only to find out that these features contain no discriminatory information
  - Finally we find a “good” feature: average intensity of the scales




  - We combine “*length*” and “*average intensity of the scales*” to improve class separability
  - We compute a linear discriminant function to separate the two classes, and obtain a classification rate of 95.7%

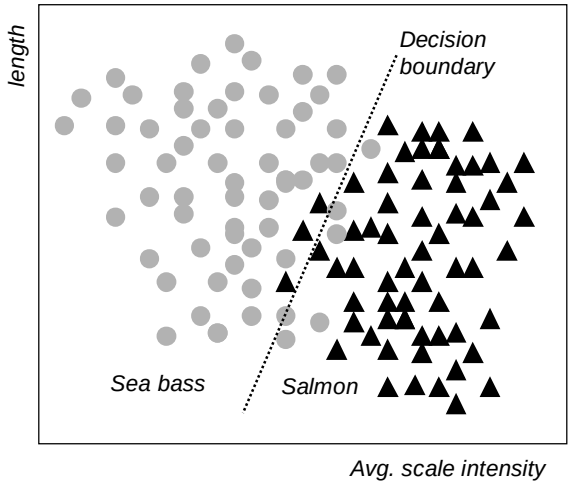


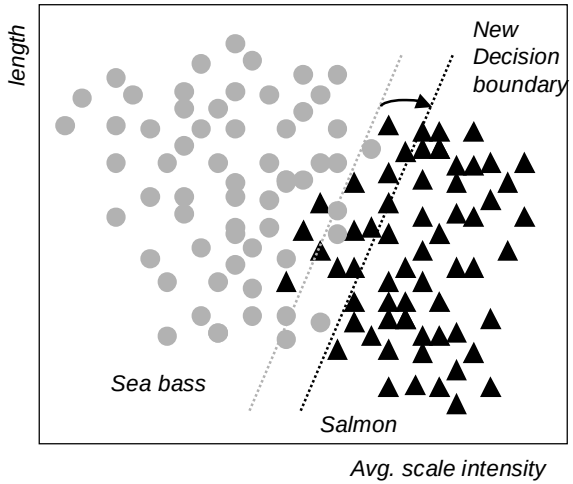



# An example

## Cost Versus Classification rate

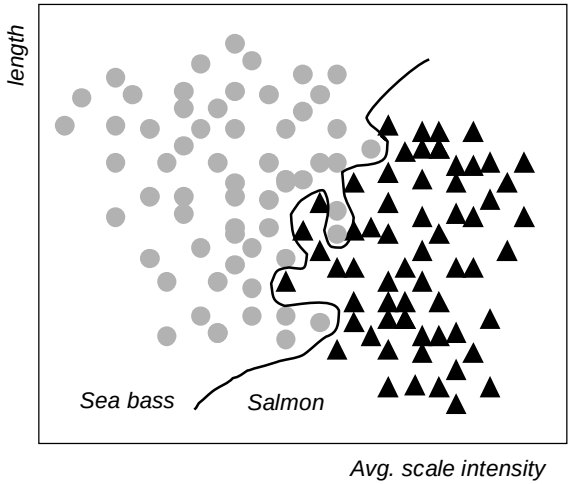
- Is classification rate the best objective function for this problem?
  - The **cost** of misclassifying salmon as sea bass is that the end customer will occasionally find a tasty piece of salmon when he purchases sea bass
  - The **cost** of misclassifying sea bass as salmon is a customer upset when he finds a piece of sea bass purchased at the price of salmon
- We could intuitively shift the decision boundary to minimize an alternative cost function



# An example

- **The issue of generalization**
  - The recognition rate of our linear classifier (95.7%) met the design specs, but we still think we can improve the performance of the system
    - *We then design an artificial neural network with five hidden layers, a combination of logistic and hyperbolic tangent activation functions, train it with the Levenberg-Marquardt algorithm and obtain an impressive classification rate of 99.9975% with the following decision boundary*

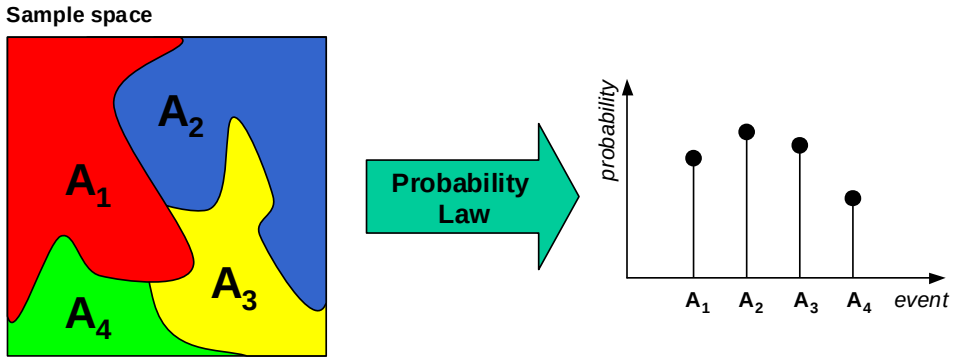



  - Satisfied with our classifier, we integrate the system and deploy it to the fish processing plant
    - A few days later the plant manager calls to complain that the system is misclassifying an average of 25% of the fish
    - **What went wrong?**

# Review of probability theory

## Probability

- Probabilities are numbers assigned to events that indicate "***how likely***" it is that the event will occur when a random experiment is performed




## Conditional Probability

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

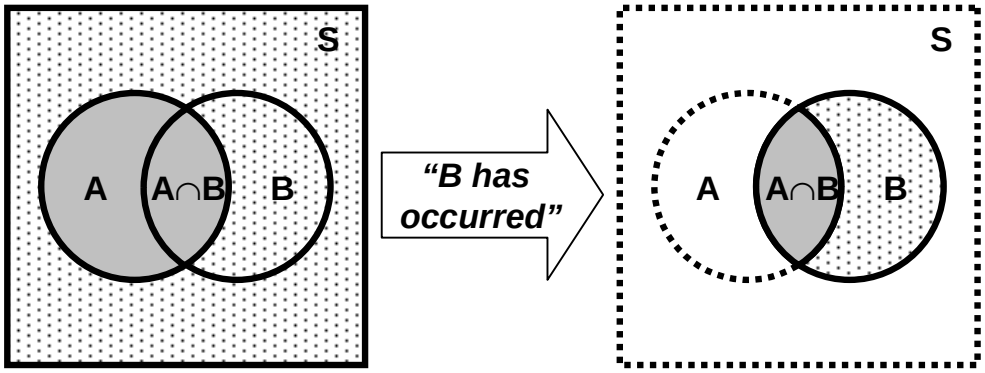
$$P[A \mid B] = \frac{P[A \cap B]}{P[B]} \quad \text{for} \quad P[B] > 0$$

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



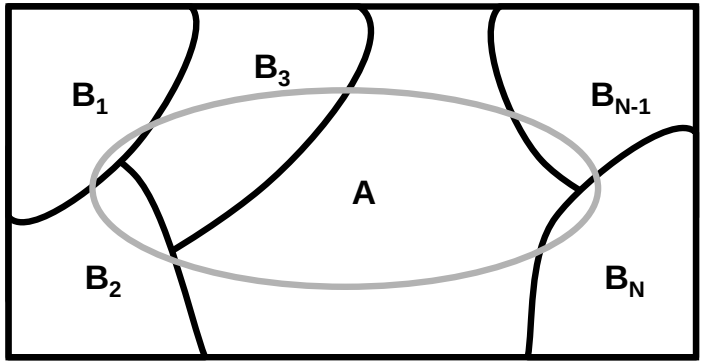
# Review of probability theory

- **Conditional probability: graphical interpretation**




- **Theorem of Total Probability**
  - Let $B_1, B_2, \ldots, B_N$ be mutually exclusive events, then

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



# Review of probability theory

## Bayes Theorem

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

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

- Bayes Theorem is definitely the fundamental relationship in Statistical Pattern Recognition

**Rev. Thomas Bayes (1702-1761)**

# Review of probability theory

- **For pattern recognition, Bayes Theorem can be expressed as**

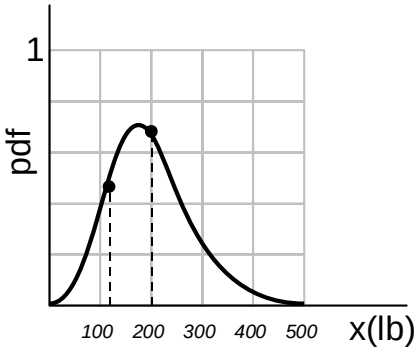
$$P(\omega_j \mid x) = \frac{P(x \mid \omega_j) \cdot P(\omega_j)}{\sum_{k=1}^{N} P(x \mid \omega_k) \cdot P(\omega_k)} = \frac{P(x \mid \omega_j) \cdot P(\omega_j)}{P(x)}$$

  - where $\omega_j$ is the $i^{th}$ class and ***x*** is the feature vector
- **Each term in the Bayes Theorem has a special name, which you should be familiar with**
  - **$P(\omega_i)$** *Prior probability* (of class $\omega_i$)
  - **$P(\omega_i|x)$** *Posterior Probability* (of class $\omega_i$ given the observation ***x***)
  - **$P(x|\omega_i)$** *Likelihood* (conditional prob. of ***x*** given class $\omega_i$)
  - **P(x)** A normalization constant that does not affect the decision
- **Two commonly used decision rules are**
  - Maximum A Posteriori (MAP): choose the class $\omega_i$ with highest $P(\omega_i|x)$
  - Maximum Likelihood (ML): choose the class $\omega_i$ with highest $P(x|\omega_i)$
    - ML and MAP are equivalent for non-informative priors ($P(\omega_i)$=constant)

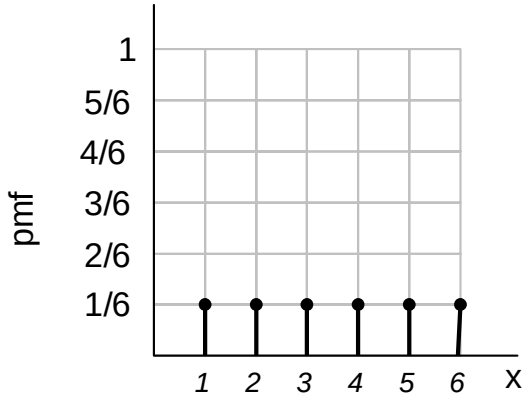
# Review of probability theory

## Characterizing features/vectors

- Complete: Probability mass/density function




*pdf for a person's weight*




*pmf for rolling a (fair) dice*

- Partial: Statistics
  - Expectation
    - The expectation represents the center of mass of a density
  - Variance
    - The variance represents the spread about the mean
  - Covariance (only for random vectors)
    - The tendency of each pair of features to vary together, i.e., to co-vary



# Review of probability theory

## The covariance matrix (cont.)

$$COV[X] = \Sigma = E[(X-\mu)(X-\mu)^T]$$

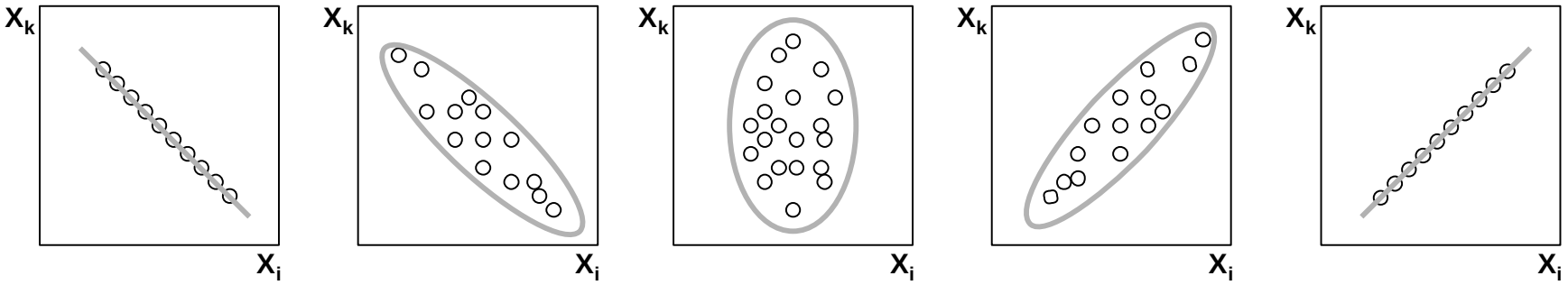
$$= \begin{bmatrix} E[(x_1-\mu_1)(x_1-\mu_1)] & \cdots & E[(x_1-\mu_1)(x_N-\mu_N)] \\ \cdots & \ddots & \\ E[(x_N-\mu_N)(x_1-\mu_1)] & \cdots & E[(x_N-\mu_N)(x_N-\mu_N)] \end{bmatrix} = \begin{bmatrix} \sigma_1^2 & \cdots & c_{1N} \\ \cdots & \cdots & \\ c_{1N} & \cdots & \sigma_N^2 \end{bmatrix}$$

- The covariance terms can be expressed as

$$c_{ii} = \sigma_i^2 \quad \text{and} \quad c_{ik} = \rho_{ik}\sigma_i\sigma_k$$

  - where $\rho_{ik}$ is called the correlation coefficient

## Graphical interpretation



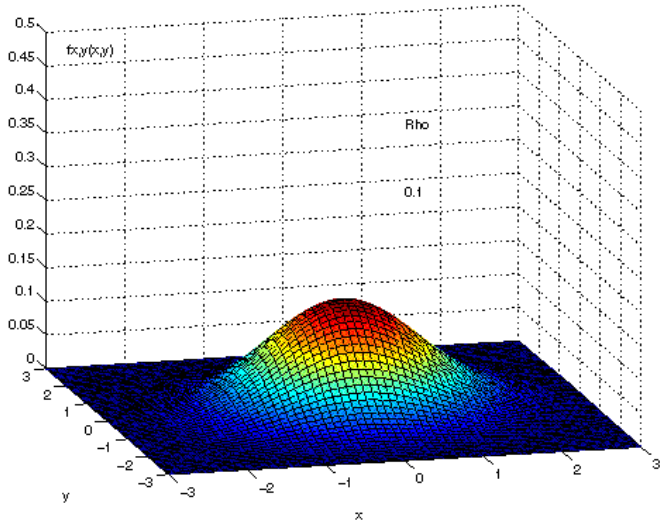
# Review of probability theory

- **Meet the multivariate Normal or Gaussian density N(μ,Σ):**

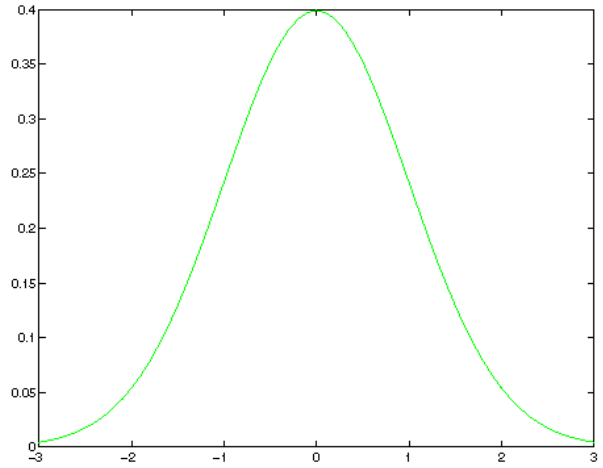
$$f_X(x) = \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} \exp\left[-\frac{1}{2}(X-\mu)^T \Sigma^{-1}(X-\mu)\right]$$

  - For a single dimension, this formula reduces to the familiar expression

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




- **Gaussian distributions are very popular**
  - The parameters (μ,Σ) are **sufficient** to uniquely characterize the normal distribution
  - If the $\mathbf{x_i}$'s are mutually **uncorrelated** ($\mathbf{c_{ik}}$=0), then they are also **independent**
    - The covariance matrix becomes diagonal, with the individual variances in the main diagonal
  - Marginal and conditional densities
  - Linear transformations

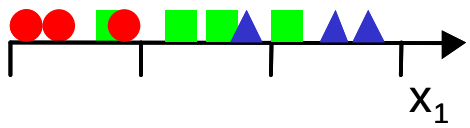


# SECTION II: Dimensionality reduction

- The “curse of dimensionality”
- Feature extraction vs. feature selection
- Feature extraction criteria
- Principal Components Analysis (briefly)
- Linear Discriminant Analysis
- Feature Subset Selection


# Dimensionality reduction

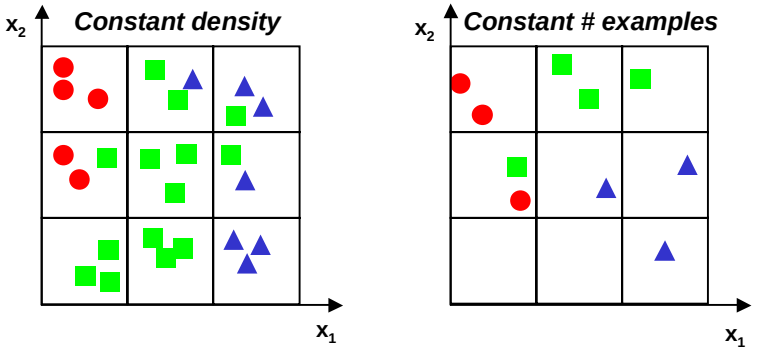
- **The "curse of dimensionality" [Bellman, 1961]**
  - Refers to the problems associated with multivariate data analysis as the dimensionality increases
- **Consider a 3-class pattern recognition problem**
  - A simple (Maximum Likelihood) procedure would be to
    - Divide the feature space into uniform bins
    - Compute the ratio of examples for each class at each bin and,
    - For a new example, find its bin and choose the predominant class in that bin
  - We decide to start with one feature and divide the real line into 3 bins



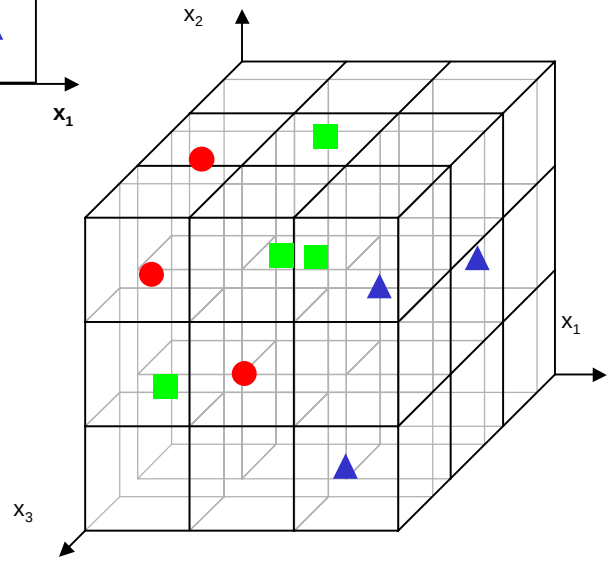

    - Notice that there exists a large overlap between classes $\Rightarrow$ to improve discrimination, we decide to incorporate a second feature

# Dimensionality reduction

- **Moving to two dimensions increases the number of bins from 3 to $3^2=9$**
  - QUESTION: Which should we maintain constant?
    - The density of examples per bin? This increases the number of examples from 9 to 27
    - The total number of examples? This results in a 2D scatter plot that is very sparse



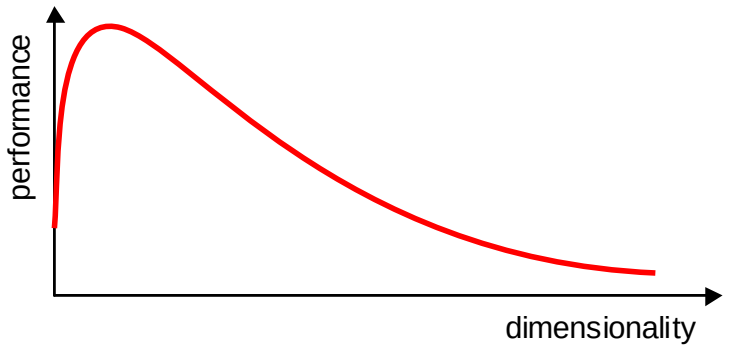

- **Moving to three features …**
  - The number of bins grows to $3^3=27$
  - To maintain the initial density of examples, the number of required examples grows to 81
  - For the same number of examples the 3D scatter plot is almost empty






# Dimensionality reduction

- **Implications of the curse of dimensionality**
  - Exponential growth with dimensionality in the number of examples required to accurately estimate a function
- **How do we beat the curse of dimensionality?**
  - By incorporating prior knowledge
  - By providing increasing smoothness of the target function
  - By reducing the dimensionality
- **In practice, the curse of dimensionality means that**
  - For a given sample size, there is a maximum number of features above which the performance of our classifier will degrade rather than improve
    - In most cases, the information that was lost by discarding some features is compensated by a more accurate mapping in lower-dimensional space



# The curse of dimensionality

- **Implications of the curse of dimensionality**
  - Exponential growth in the number of examples required to maintain a given sampling density
    - For a density of N examples/bin and D dimensions, the total number of examples grows as $N^D$
  - Exponential growth in the complexity of the target function (a density) with increasing dimensionality
    - "A function defined in high-dimensional space is likely to be much more complex than a function defined in a lower-dimensional space, and those complications are harder to discern" –Jerome Friedman
      - This means that a more complex target function requires denser sample points to learn it well!
  - What to do if it ain't Gaussian?
    - For one dimension, a large number of density functions are available
    - For most multivariate problems, only the Gaussian density is employed
    - For high-dimensional data, the Gaussian density can only be handled in a simplified form!
  - Human performance in high dimensions
    - Humans have an extraordinary capacity to discern patterns in 1-3 dimensions, but these capabilities degrade drastically for 4 or higher dimensions

# Dimensionality reduction

- **Two approaches to perform dim. reduction $\Re^N \rightarrow \Re^M$ (M<N)**
  - **Feature selection**: choosing a subset of all the features

$$[x_1\ x_2 \ldots x_N] \xrightarrow{\text{feature selection}} [x_{i_1}\ x_{i_2} \ldots x_{i_M}]$$

  - **Feature extraction**: creating new features by combining existing ones

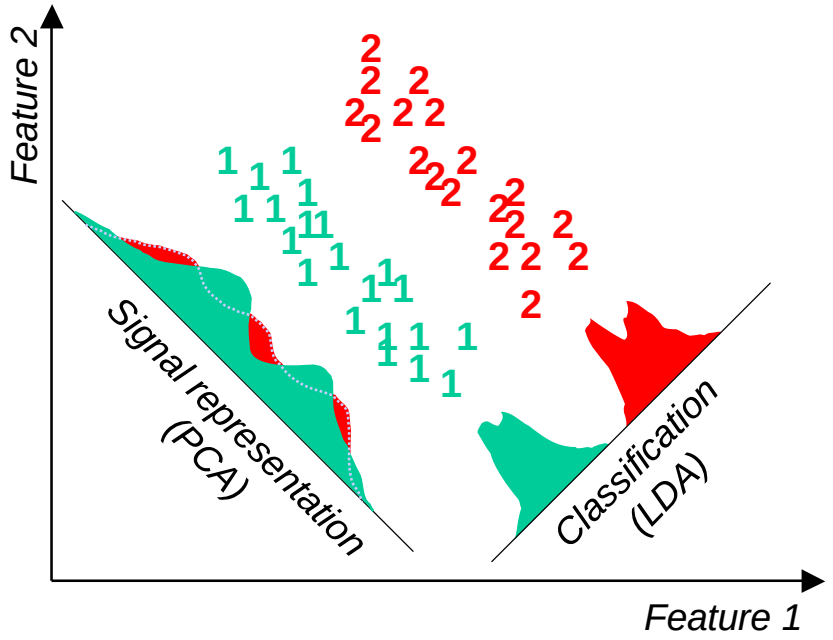
$$[x_1\ x_2 \ldots x_N] \xrightarrow{\text{feature extraction}} [y_1\ y_2 \ldots y_M] = f([x_{i_1}\ x_{i_2} \ldots x_{i_M}])$$

    - In either case, the goal is to find a low-dimensional representation of the data that preserves (most of) the information or structure in the data
- **Linear feature extraction**
  - The "optimal" mapping y=f(x) is, in general, a non-linear function whose form is problem-dependent
    - Hence, feature extraction is commonly limited to linear projections **y=Wx**

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix} \xrightarrow{\text{linear feature extraction}} \begin{bmatrix} y_1 \\ y_2 \\ \\ y_M \end{bmatrix} = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1N} \\ w_{21} & w_{22} & \cdots & w_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ w_{M1} & w_{M2} & & w_{MN} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix}$$

# Signal representation versus classification

- **Two criteria can be used to find the "optimal" feature extraction mapping y=f(x)**
  - **Signal representation**: The goal of feature extraction is to represent the samples accurately in a lower-dimensional space
  - **Classification**: The goal of feature extraction is to enhance the class-discriminatory information in the lower-dimensional space
- **Within the realm of linear feature extraction, two techniques are commonly used**
  - Principal Components (PCA)
    - Based on signal representation
  - Fisher's Linear Discriminant (LDA)
    - Based on classification



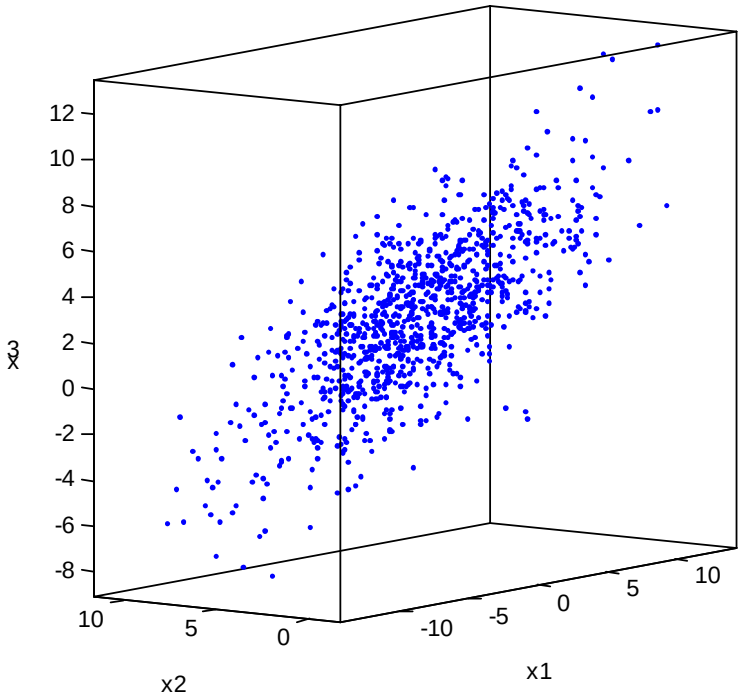
# Principal Components Analysis

- **Summary**
  - The optimal* approximation of a random vector $x \in \Re^N$ by a linear combination of M (M<N) independent vectors is obtained by projecting the random vector x onto the eigenvectors $\varphi_i$ corresponding to the largest eigenvalues $\lambda_i$ of the covariance matrix $\Sigma_x$
    - *optimality is defined as the minimum of the sum-square magnitude of the approximation error
  - Since PCA uses the eigenvectors of the covariance matrix $\Sigma_x$, it is able to find the independent axes of the data under the unimodal Gaussian assumption
    - However, for non-Gaussian or multi-modal Gaussian data, PCA simply de-correlates the axes
  - The main limitation of PCA is that it does not consider class separability since it does not take into account the class label of the feature vector
    - PCA simply performs a coordinate rotation that aligns the transformed axes with the directions of maximum variance
    - **There is no guarantee that the directions of maximum variance will contain good features for discrimination!!**

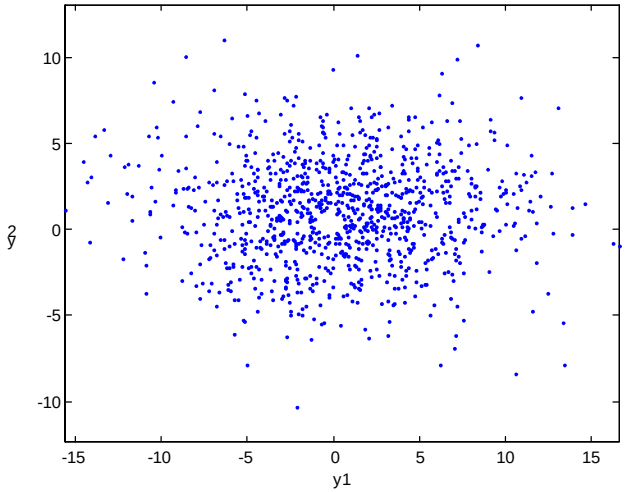
# PCA example

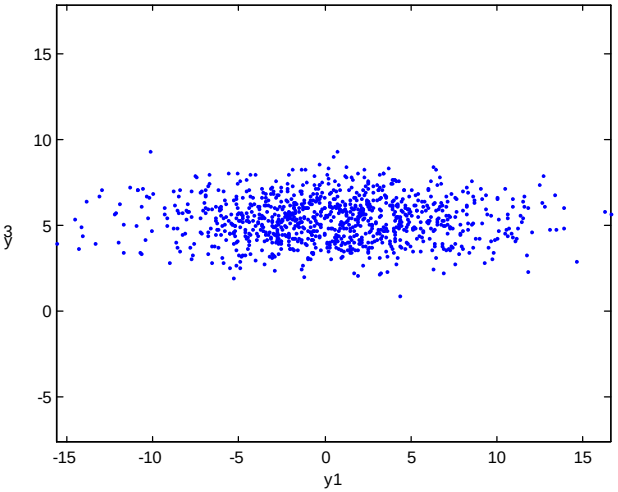
- **3D Gaussian distribution**
  - Mean and covariance are

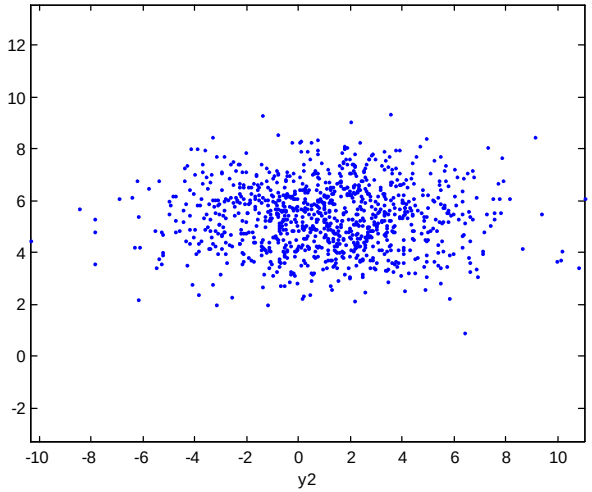
$$\mu = [0 \;\; 5 \;\; 2]^T \text{ and } \Sigma = \begin{bmatrix} 25 & -1 & 7 \\ -1 & 4 & -4 \\ 7 & -4 & 10 \end{bmatrix}$$




- **RESULTS**
  - First PC has largest variance
  - PCA projections are de-correlated

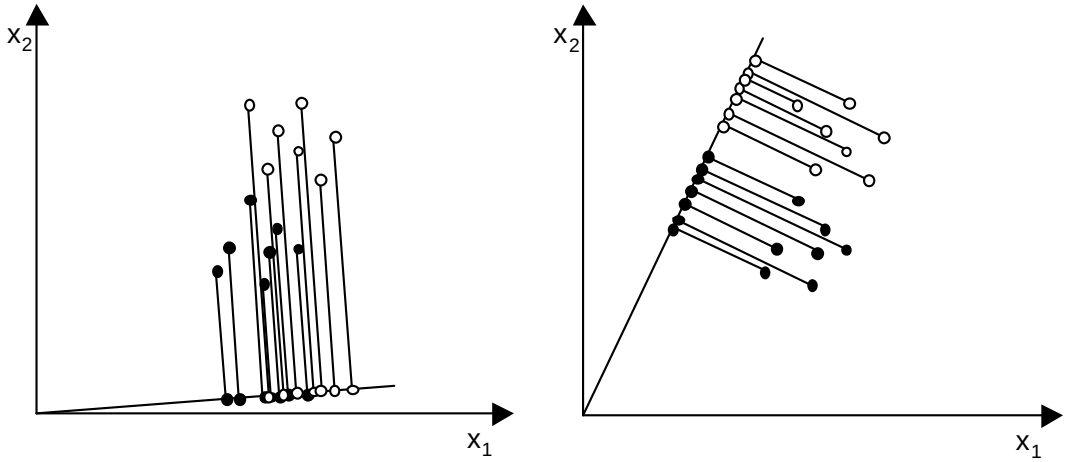


# Linear Discriminant Analysis, two-classes

- **The objective of LDA is to perform dimensionality reduction while preserving as much of the class discriminatory information as possible**
  - Assume a set of D-dimensional samples $\{x_1, x_2, \ldots, x_N\}$, $N_1$ of which belong to class $\omega_1$, and $N_2$ to class $\omega_2$
  - We seek to obtain a scalar $y$ by projecting the samples $x$ onto a line $y=w^Tx$
  - Of all possible lines we want the one that maximizes the separability of the scalars



# Linear Discriminant Analysis, two-classes

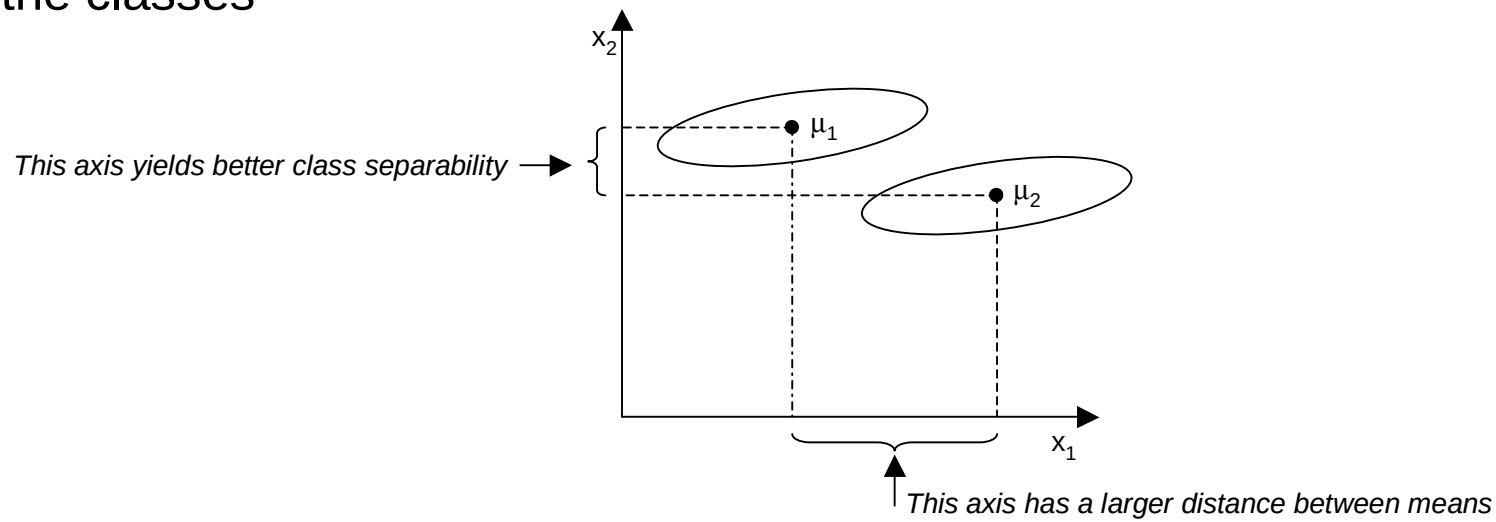
- **In order to find a good projection vector, we need to define a measure of separation between the projections**
  - We could choose the distance between the projected means

$$J(w) = \left| \tilde{\mu}_1 - \tilde{\mu}_2 \right| = \left| w^T (\mu_1 - \mu_2) \right|$$

    - Where the mean values of the *x* and *y* examples are

$$\mu_i = \frac{1}{N_i} \sum_{x \in \omega_i} x \quad \text{and} \quad \tilde{\mu}_i = \frac{1}{N_i} \sum_{y \in \omega_i} y = \frac{1}{N_i} \sum_{y \in \omega_i} w^T x = w^T \mu_i$$

  - However, the distance between projected means is not a very good measure since it does not take into account the standard deviation within the classes



# Linear Discriminant Analysis, two-classes

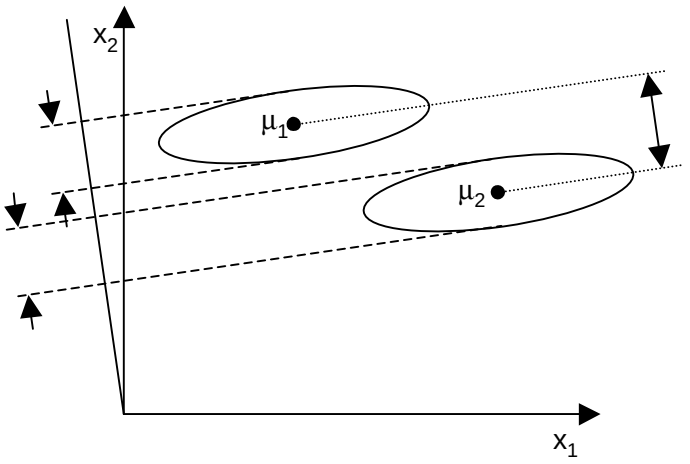
- The solution proposed by Fisher is to normalize the difference between the means by a measure of the within-class variance
  - For each class we define the scatter, an equivalent of the variance, as

$$\tilde{s}_i^2 = \sum_{y \in \omega_i} \left(y - \tilde{\mu}_i\right)^2$$

  - and the quantity $\left(\tilde{s}_1^2 + \tilde{s}_2^2\right)$ is called the within-class scatter of the projected examples
- The Fisher linear discriminant is defined as the linear function $\mathbf{w}^T\mathbf{x}$ that maximizes the criterion function

$$J(w) = \frac{\left|\tilde{\mu}_1 - \tilde{\mu}_2\right|^2}{\tilde{s}_1^2 + \tilde{s}_2^2}$$

- In a nutshell: we look for a projection where examples from the same class are projected very close to each other and, at the same time, the projected means are as farther apart as possible



# Linear Discriminant Analysis, two-classes

- Noticing that

$$(\tilde{\mu}_1 - \tilde{\mu}_2)^2 = (w^T\mu_1 - w^T\mu_2)^2 = w^T \underbrace{(\mu_1 - \mu_2)(\mu_1 - \mu_2)^T}_{\text{BETWEEN-CLASS SCATTER } S_B} w = w^T S_B w$$

$$\tilde{s}_i^2 = \sum_{y \in \omega_i} (y - \tilde{\mu}_i)^2 = \sum_{x \in \omega_i} (w^T x - w^T \mu_i)^2 = w^T \underbrace{\sum_{x \in \omega_i} (x - \mu_i)(x - \mu_i)^T}_{\text{CLASS SCATTER } S_i} w = w^T S_i w$$

$$\tilde{s}_1^2 + \tilde{s}_2^2 = w^T \underbrace{(S_1 + S_2)}_{\text{WITHIN-CLASS SCATTER } S_W} w = w^T S_W w$$

- We can express J(w) in terms of x and w as

$$J(w) = \frac{w^T S_B w}{w^T S_W w}$$

- It can be shown that the projection vector w* which maximizes J(w) is

$$w^* = \underset{w}{\text{argmax}} \left\{ \frac{w^T S_B w}{w^T S_W w} \right\} = S_W^{-1}(\mu_1 - \mu_2)$$

# Linear Discriminant Analysis, C-classes

- **Fisher's LDA generalizes for C-class problems very gracefully**
  - For the C class problem we will seek (C-1) projection vectors $w_i$, which can be arranged by columns into a projection matrix $W=[w_1|w_2|\ldots|w_{C-1}]$
- **The solution uses a slightly different formulation**
  - The within-class scatter matrix is similar as the two-class problem

$$S_W = \sum_{i=1}^{C} S_i = \sum_{i=1}^{C} \sum_{x \in \omega_i} (x - \mu_i)(x - \mu_i)^T$$

  - The generalization of the between-class scatter matrix is

$$S_B = \sum_{i=1}^{C} N_i (\mu_i - \mu)(\mu_i - \mu)^T$$

    - Where $\mu$ is the mean of the entire dataset
  - The objective function becomes

$$J(W) = \frac{\left|\tilde{S}_B\right|}{\left|\tilde{S}_W\right|} = \frac{\left|W^T S_B W\right|}{\left|W^T S_W W\right|}$$

    - Recall that we are looking for a projection that, in some sense, maximizes the ratio of between-class to within-class scatter:
      - Since the projection is not scalar (it has C-1 dimensions), we have used the determinant of the scatter matrices to obtain a scalar criterion function

# Linear Discriminant Analysis, C-classes

- It can be shown that the optimal projection matrix W* is the one whose columns are the eigenvectors corresponding to the largest eigenvalues of the following generalized eigenvalue problem

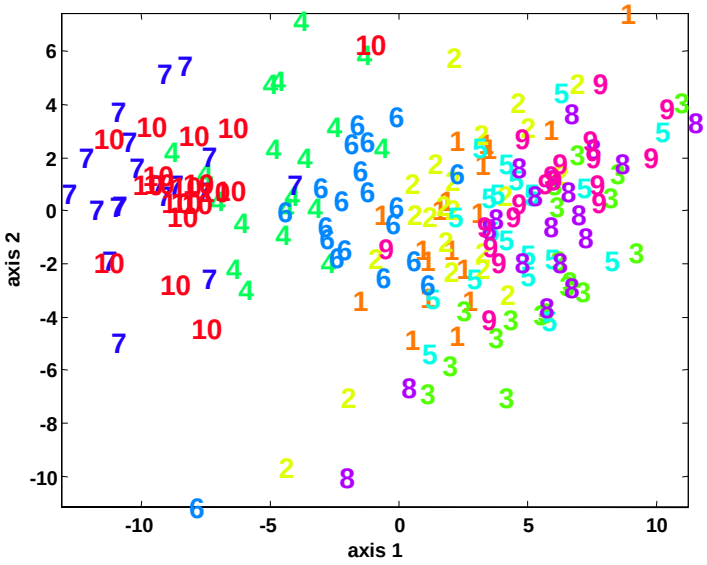
$$W^* = \left[ w_1^* \mid w_2^* \mid \cdots \mid w_{C-1}^* \right] = \arg\max \left\{ \frac{W^T S_B W}{W^T S_W W} \right\} \Rightarrow (S_B - \lambda_i S_W) w_i^* = 0$$

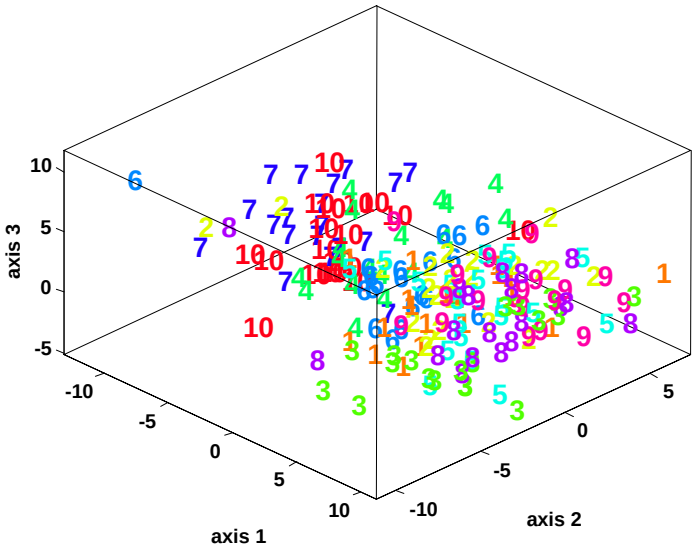
## NOTE

- $S_B$ is the sum of C matrices of rank one or less and the mean vectors are constrained by $\Sigma N_i \mu_i = N\mu$
  - Therefore, $S_B$ will be at most of rank (C-1)
  - This means that only (C-1) of the eigenvalues $\lambda_i$ will be non-zero
- Therefore, LDA produces at most C-1 feature projections
  - If the classification error estimates establish that more features are needed, some other method must be employed to provide those additional features

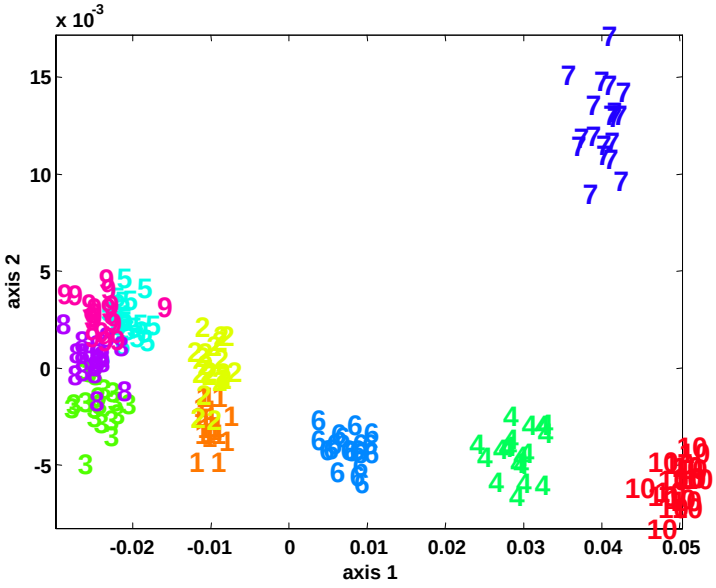
# PCA Versus LDA

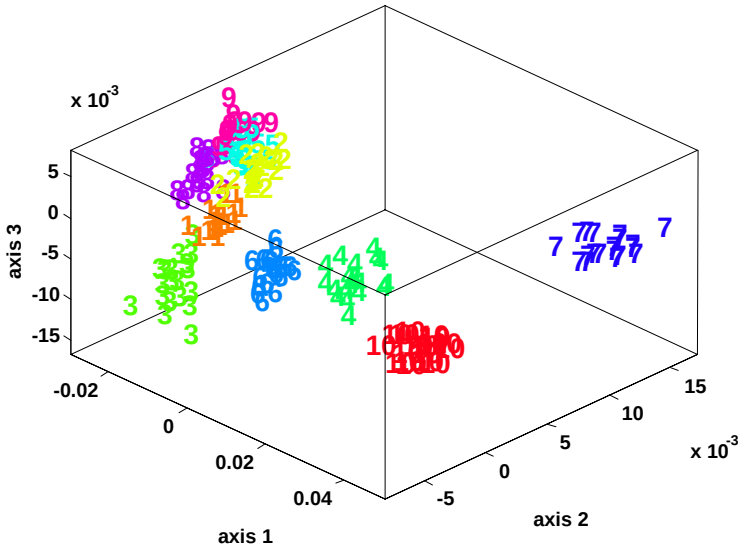
**PCA**




**LDA**

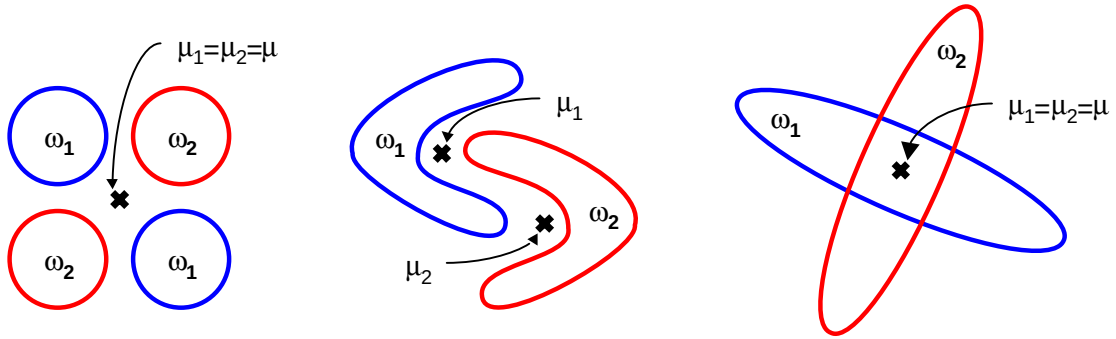


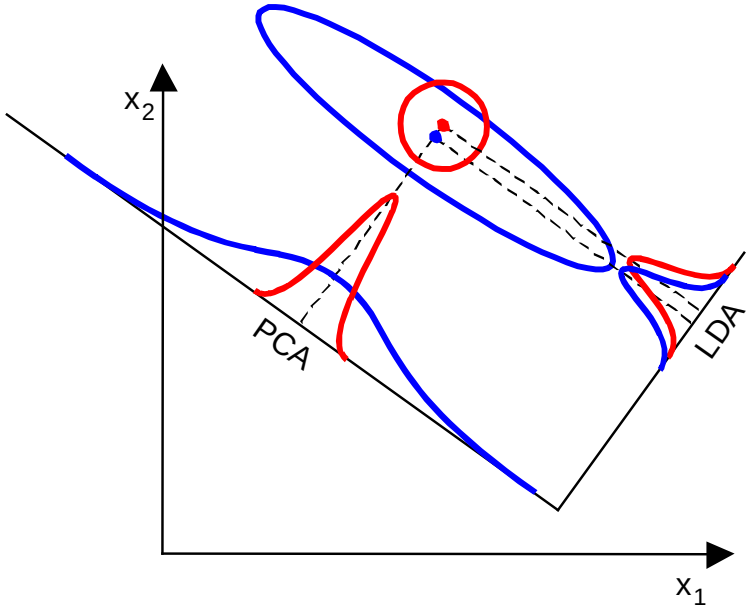



# Limitations of LDA

- **LDA assumes unimodal Gaussian likelihoods**
  - If the densities are significantly non-Gaussian, LDA may not preserve any complex structure of the data needed for classification




- **LDA will fail when the discriminatory information is not in the mean but rather in the variance of the data**

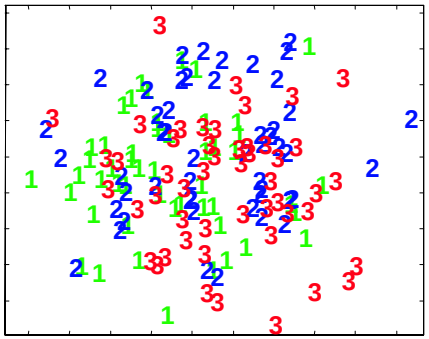




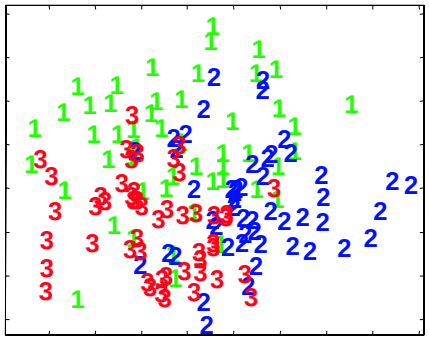

# Limitations of LDA

- **LDA has a tendency to overfit training data**
  - To illustrate this problem, we generate an artificial dataset
    - Three classes, 50 examples per class, with the exact same likelihood: a multivariate Gaussian with zero mean and identity covariance
    - As we arbitrarily increase the number of dimensions, classes appear to separate better, even though they come from the same distribution

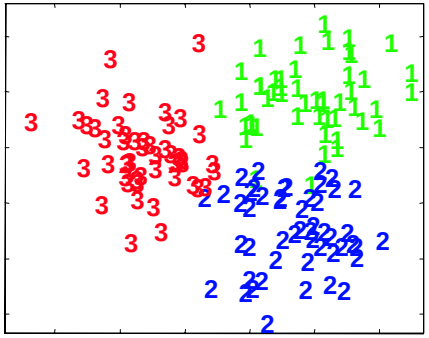
**10 dimensions**



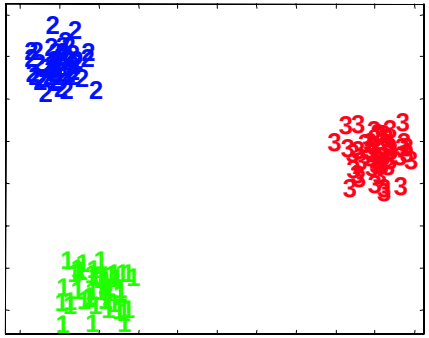
**50 dimensions**



**100 dimensions**



**150 dimensions**



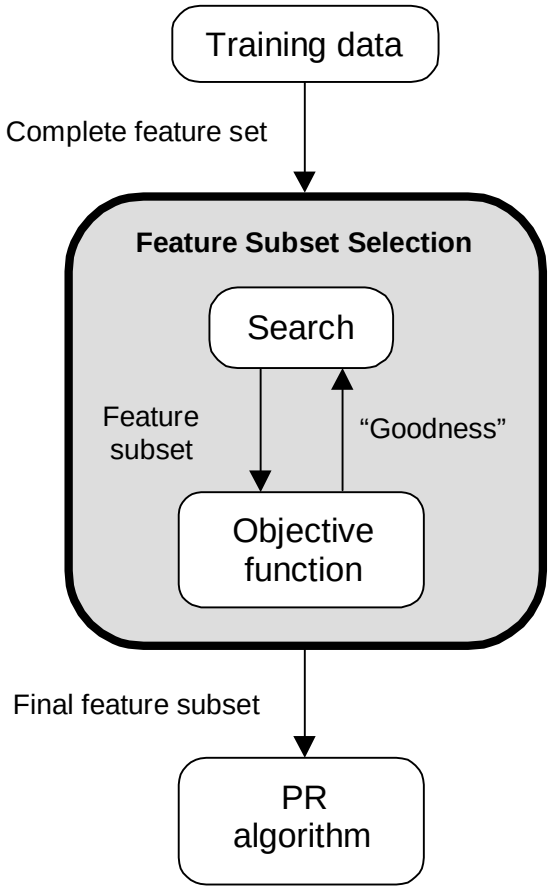


# Feature Subset Selection

- **Why Feature Subset Selection?**
  - Feature Subset Selection is necessary in a number of situations
    - Features may be expensive to obtain
      - You evaluate a large number of features (sensors) in the test bed and select only a few for the final implementation
    - You may want to extract meaningful rules from your classifier
      - When you transform or project, the measurement units (length, weight, etc.) of your features are lost
    - Features may not be numeric
      - A typical situation in the machine learning domain
- **Although FSS can be thought of as a special case of feature extraction (think of an identity projection matrix with a few diagonal zeros), in practice it is a quite different problem**
  - FSS looks at dimensionality reduction from a different perspective
  - FSS has a unique set of methodologies

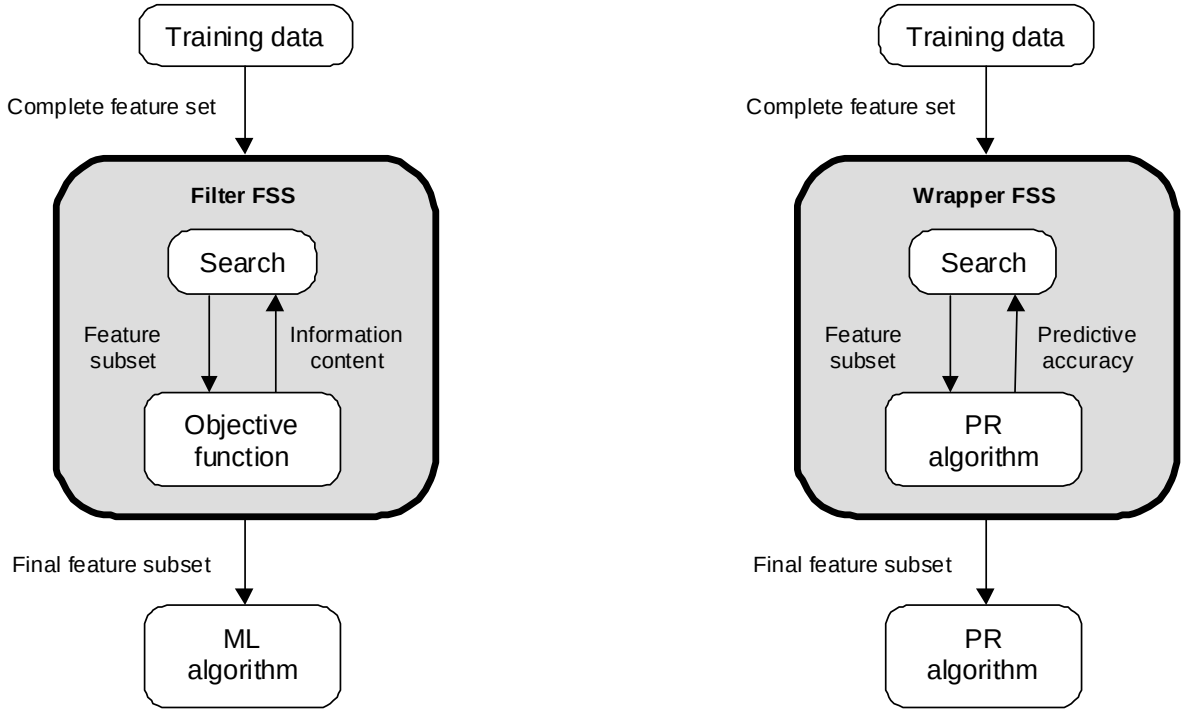
# Search strategy and objective function

- **Feature Subset Selection requires**
  - A search strategy to select candidate subsets
  - An objective function to evaluate candidates
- **Search Strategy**
  - Exhaustive evaluation involves $\binom{N}{M}$ feature subsets for a fixed value of M, and $2^N$ subsets if M must be optimized as well
    - This number of combinations is unfeasible, even for moderate values of M and N
      - For example, exhaustive evaluation of 10 out of 20 features involves 184,756 feature subsets; exhaustive evaluation of 10 out of 20 involves more than $10^{13}$ feature subsets
  - A search strategy is needed to explore the space of all possible feature combinations
- **Objective Function**
  - The objective function evaluates candidate subsets and returns a measure of their "goodness", a feedback that is used by the search strategy to select new candidates



# Objective function

- **Objective functions are divided in two groups**
  - **Filters**: The objective function evaluates feature subsets by their information content, typically interclass distance, statistical dependence or information-theoretic measures
  - **Wrappers**: The objective function is a pattern classifier, which evaluates feature subsets by their predictive accuracy (recognition rate on test data) by statistical resampling or cross-validation



# Filter types

- **Distance or separability measures**
  - These methods use distance metrics to measure class separability:
    - Distance between classes: Euclidean, Mahalanobis, etc.
    - Determinant of $S_W^{-1}S_B$ (LDA eigenvalues)
- **Correlation and information-theoretic measures**
  - These methods assume that "good" subsets contain features highly correlated with (predictive of) the class, yet uncorrelated with each other
    - Linear relation measures
      - Linear relationship between variables can be measured using the correlation coefficient $J(Y_M)$
        - where $\rho$ is a correlation coefficient

$$J(Y_M) = \frac{\sum_{i=1}^{M} \rho_{ic}}{\sum_{i=1}^{M} \sum_{j=i+1}^{M} \rho_{ij}}$$

    - Non-Linear relation measures
      - Correlation is only capable of measuring linear dependence. A more powerful method is the mutual information $I(Y_k;C)$

$$J(Y_M) = I(Y_M;C) = H(C) - H(C \mid Y_M) = \sum_{c=1}^{C} \int_{Y_M} P(Y_M, \omega_c) \lg \frac{P(Y_M, \omega_c)}{P(Y_M)\,P(\omega_c)} dx$$

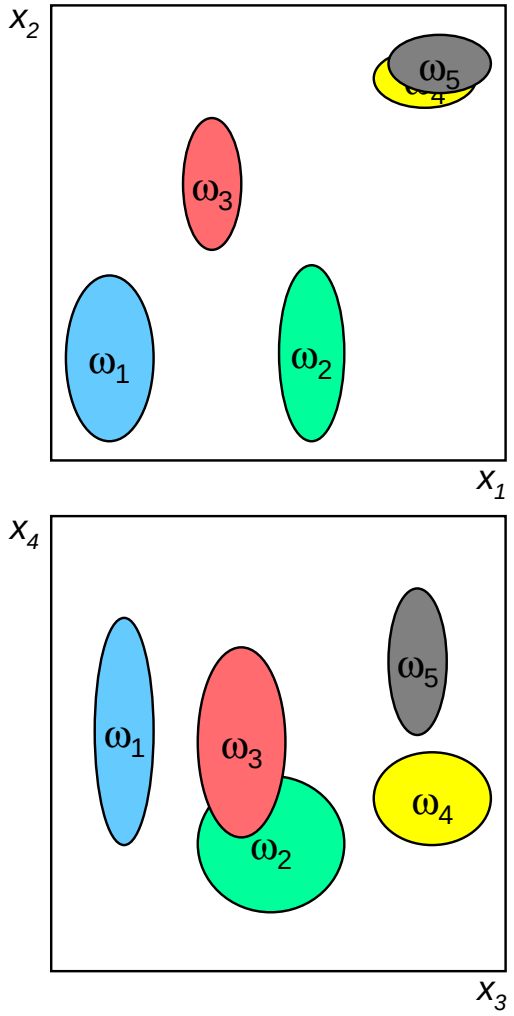
        - The mutual information between feature vector and class label $I(Y_M;C)$ measures the amount by which the uncertainty in the class $H(C)$ is decreased by knowledge of the feature vector $H(C|Y_M)$, where $H(\cdot)$ is the entropy function

# Filters Vs. Wrappers

- **Wrappers**
  - Advantages
    - **Accuracy**: wrappers generally achieve better recognition rates than filters since they are tuned to the specific interactions between the classifier and the dataset
    - **Ability to generalize**: wrappers have a mechanism to avoid overfitting, since they typically use cross-validation measures of predictive accuracy
  - Disadvantages
    - **Slow execution**: since the wrapper must train a classifier for each feature subset (or several classifiers if cross-validation is used), the method is unfeasible for computationally intensive classifiers
    - **Lack of generality**: the solution lacks generality since it is tied to the bias of the classifier used in the evaluation function
- **Filters**
  - Advantages
    - **Fast execution**: Filters generally involve a non-iterative computation on the dataset, which can execute much faster than a classifier training session
    - **Generality**: Since filters evaluate the intrinsic properties of the data, rather than their interactions with a particular classifier, their results exhibit more generality: the solution will be “good” for a larger family of classifiers
  - Disadvantages
    - **Tendency to select large subsets**: Since the filter objective functions are generally monotonic, the filter tends to select the full feature set as the optimal solution. This forces the user to select an arbitrary cutoff on the number of features to be selected

# Naïve sequential feature selection

- **One may be tempted to evaluate each individual feature separately and select those M features with the highest scores**
  - Unfortunately, this strategy will very rarely work since it does not account for feature dependence
- **An example will help illustrate the poor performance of this naïve approach**
  - The scatter plots show a 4-dimensional pattern recognition problem with 5 classes
    - The objective is to select the best subset of 2 features using the naïve sequential FSS procedure
  - A reasonable objective function will generate the following feature ranking: $J(x_1)>J(x_2)\approx J(x_3)>J(x_4)$
    - The optimal feature subset turns out to be $\{x_1, x_4\}$, because $x_4$ provides the only information that $x_1$ needs: discrimination between classes $\omega_4$ and $\omega_5$
    - If we were to choose features according to the individual scores $J(x_k)$, we would choose $x_1$ and either $x_2$ or $x_3$, leaving classes $\omega_4$ and $\omega_5$ non separable



# Search strategies

- **FSS search strategies can be grouped in three categories***
  - Sequential
    - These algorithms add or remove features sequentially, but have a tendency to become trapped in local minima
      - Sequential Forward Selection
      - Sequential Backward Selection
      - Sequential Floating Selection
  - Exponential
    - These algorithms evaluate a number of subsets that grows exponentially with the dimensionality of the search space
      - Exhaustive Search (already discussed)
      - Branch and Bound
      - Beam Search
  - Randomized
    - These algorithms incorporating randomness into their search procedure to escape local minima
      - Simulated Annealing
      - Genetic Algorithms

***The subsequent description of FSS is borrowed from [Doak, 1992]**

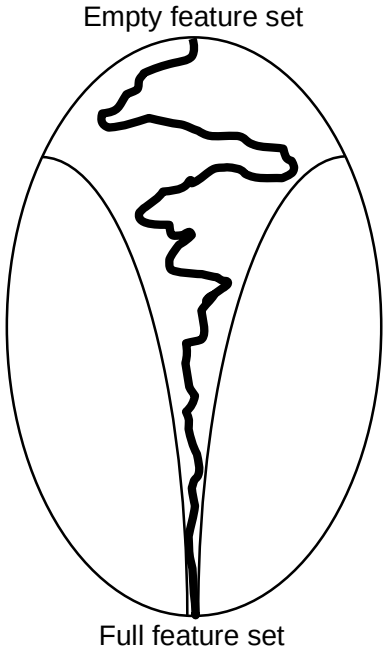


# Sequential Forward Selection (SFS)

- **Sequential Forward Selection is a simple greedy search**

1. Start with the empty set $Y=\{\varnothing\}$
2. Select the next best feature $x^+ = \arg\max_{x \in X - Y_k} \left[ J(Y_k + x) \right]$
3. Update $Y_{k+1}=Y_k+x$; $k=k+1$
4. Go to 2

- **Notes**
  - SFS performs best when the optimal subset has a small number of features
    - When the search is near the empty set, a large number of states can be potentially evaluated
    - Towards the full set, the region examined by SFS is narrower since most of the features have already been selected
    - The main disadvantage of SFS is that it is unable to remove features that become obsolete with the addition of new features

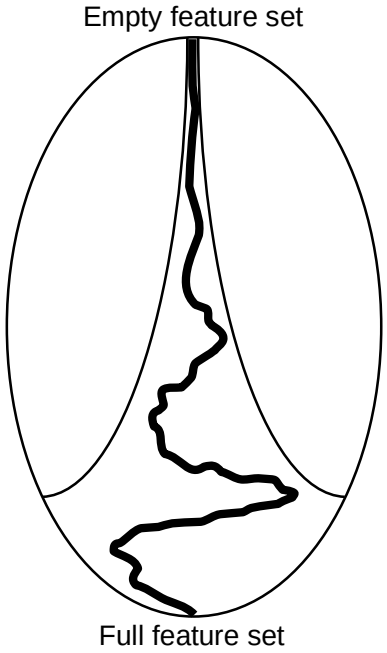


# Sequential Backward Selection (SBS)

- **Sequential Backward Selection works in the opposite manner as SFS**

1. Start with the full set Y=X
2. Remove the worst feature $x^- = \arg\max_{x \in Y_k} [J(Y_k - x)]$
3. Update $Y_{k+1}=Y_k-x$; k=k+1
4. Go to 2

- **Notes**
  - SBS works best when the optimal feature subset has a large number of features, since SBS spends most of its time visiting large subsets
  - The main limitation of SBS is its inability to reevaluate the usefulness of a feature after it has been discarded

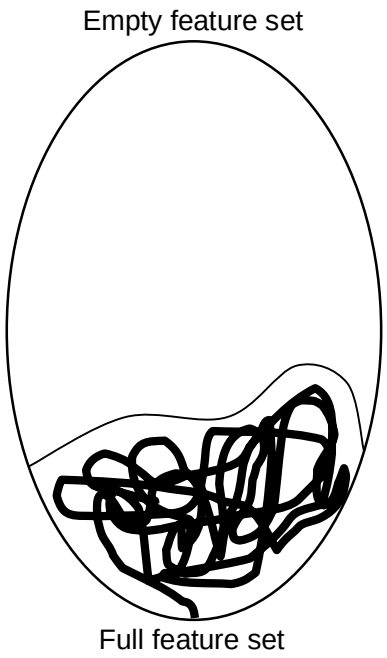


# Branch and Bound

- **The Branch and Bound algorithm is guaranteed to find the optimal feature subset under the monotonicity assumption**
  - The monotonicity assumption states that the addition of features can only increase the value of the objective function, this is

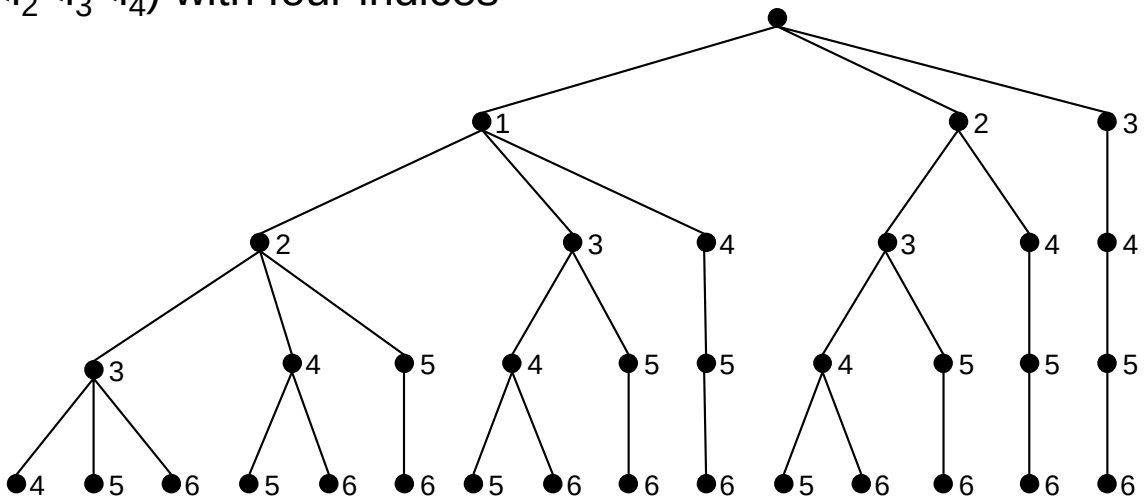
$$J(x_{i_1}) < J(x_{i_1}, x_{i_2}) < J(x_{i_1}, x_{i_2}, x_{i_3}) < \cdots < J(x_{i_1}, x_{i_2}, \cdots, x_{i_N})$$

  - Branch and Bound starts from the full set and removes features using a depth-first strategy
    - Nodes whose objective function are lower than the current best are not explored since the monotonicity assumption ensures that their children will not contain a better solution



# Branch and Bound

- **Algorithm**
  - The algorithm is better explained by considering the subsets of M'=N-M features already discarded, where N is the dimensionality of the state space and M is the desired number of features
  - Since the order of the features is irrelevant, we will only consider an increasing ordering $i_1<i_2<...i_{M'}$ of the feature indices, this will avoid exploring states that differ only in the ordering of their features
  - The Branch and Bound tree for N=6 and M=2 is shown below (numbers indicate features that are being removed)
    - Notice that at the level directly below the root we only consider removing features 1, 2 or 3, since a higher number would not allow sequences ($i_1<i_2<i_3<i_4$) with four indices

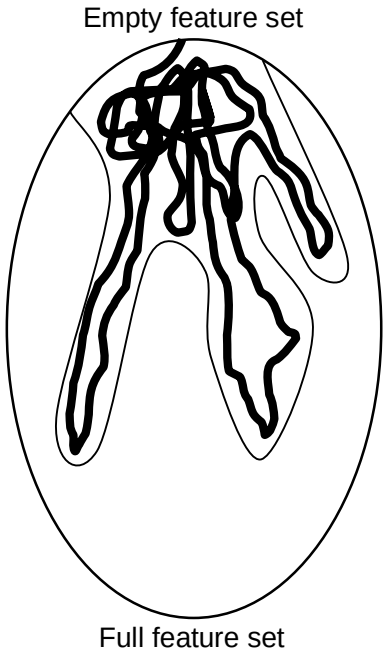


# Branch and Bound

1. Initialize: $\alpha=-\infty$, k=0
2. Generate successors of the current node and store them in LIST(k)
3. Select new node
   if LIST(k) is empty
      go to Step 5
   else
      $i_k = \arg\max_{j \in \text{LIST}(k)} \left[ J(x_{i_1}, x_{i_2}, \ldots x_{i_{k-1}}, j) \right]$
      delete $i_k$ from LIST(k)
4. Check bound
   if $J(x_{i_1}, x_{i_2}, \ldots x_{i_k}) < \alpha$
      go to 5
   else if k=M' (we have the desired number of features)
      go to 6
   else
      k=k+1
      go to 2
5. Backtrack to lower level
   set k=k-1
   if k=0
      terminate algorithm
   else
      go to 3
6. Last level
   Set $\alpha = J(x_{i_1}, x_{i_2}, \ldots x_{i_{k-1}}, j)$ and $Y_{M'}^{*} = \{x_{i_1}, x_{i_2}, \ldots x_{i_k}\}$
   go to 5

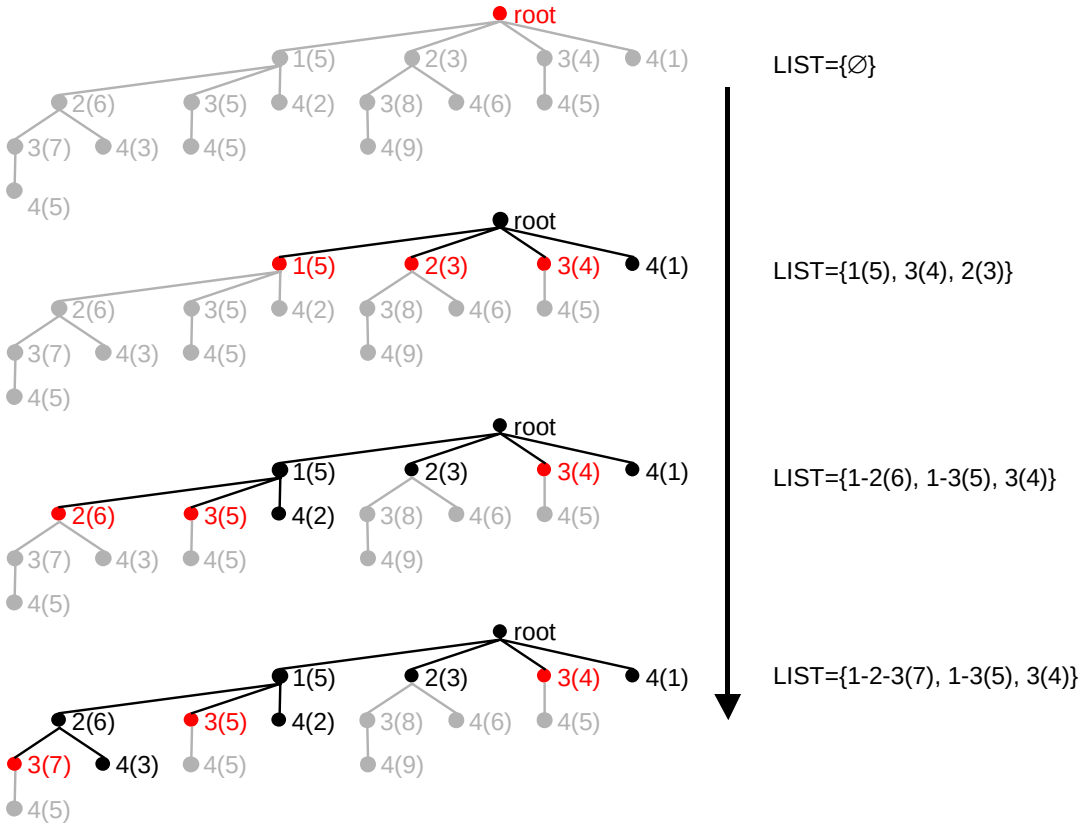
# Beam Search

- **Beam Search is a variation of best-first search with a bounded queue to limit the scope of the search**
  - The queue organizes states from best to worst, with the best states placed at the head of the queue
  - At every iteration, BS evaluates all possible states that result from adding a feature to the feature subset, and the results are inserted into the queue in their proper locations
    - It is trivial to notice that BS degenerates to Exhaustive search if there is no limit on the size of the queue. Similarly, if the queue size is set to one, BS is equivalent to Sequential Forward Selection



# Beam Search

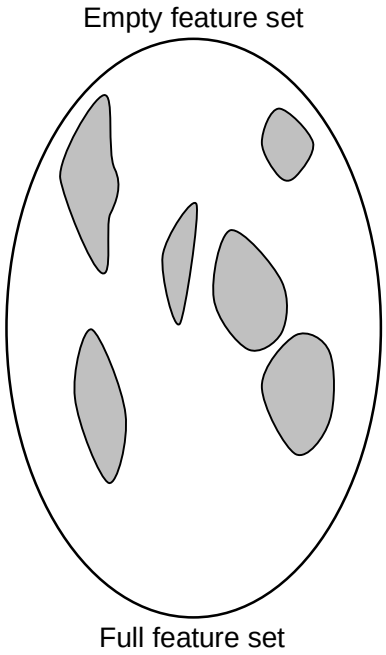
- **This example illustrates BS for a 4D space and a queue of size 3**
  - BS cannot guarantee that the optimal subset is found:
    - in the example, the optimal is 2-3-4(9), which is never explored
    - however, with the proper queue size, Beam Search can avoid local minimal by preserving solutions from varying regions in the search space



# Genetic Algorithms

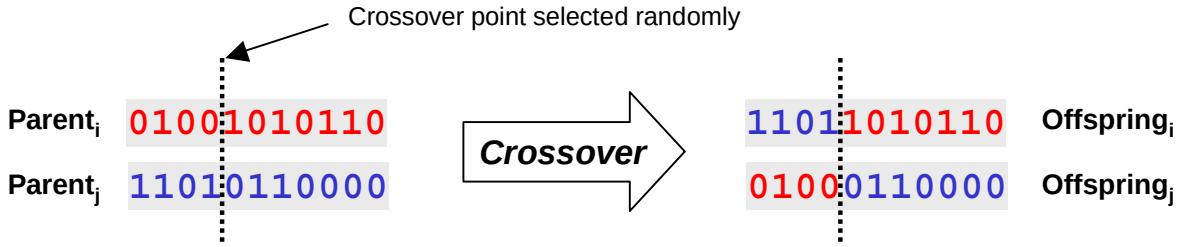
- **Genetic algorithms are optimization techniques that mimic the evolutionary process of "survival of the fittest"**
  - Starting with an initial random population of solutions, evolve new populations by mating (crossover) pairs of solutions and mutating solutions according to their fitness (an objective function)
  - The better solutions are more likely to be selected for the mating and mutation operators and, therefore, carry their "genetic code" from generation to generation
  - In FSS, individual solutions are represented with a binary number (1 if the feature is selected, 0 otherwise)

1. Create an initial random population
2. Evaluate initial population
3. Repeat until convergence (or a number of generations)
   - 3a. Select the fittest individuals in the population
   - 3b. Perform crossover on selected individuals to create offspring
   - 3c. Perform mutation on selected individuals
   - 3d. Create new population from old population and offspring
   - 3e. Evaluate the new population

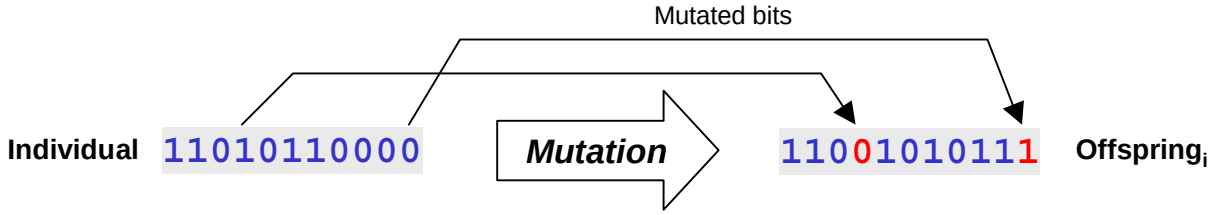


# Genetic operators

- **Single-point crossover**
  - Select two individuals (parents) according to their fitness
  - Select a crossover point
  - With probability $P_c$ (0.95 is reasonable) create two offspring by combining the parents

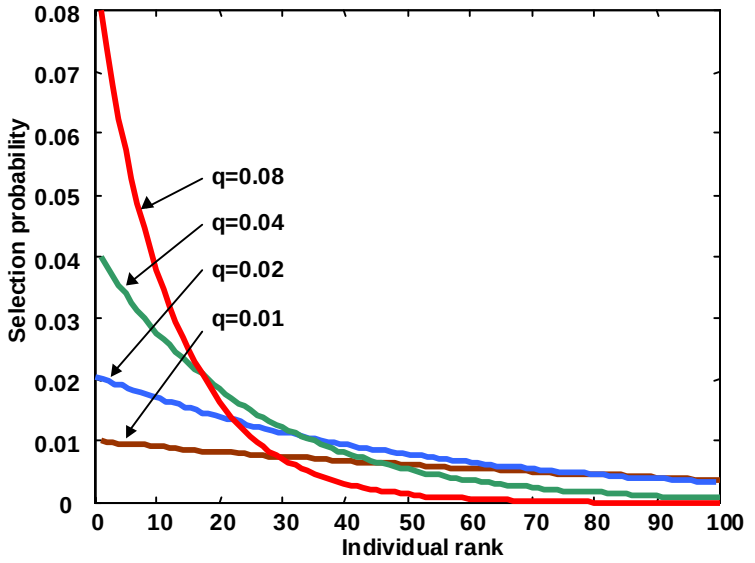



- **Binary mutation**
  - Select an individual according to its fitness
  - With probability $P_M$ (0.01 is reasonable) mutate each one of its bits



# Selection methods

- **The selection of individuals is based on their fitness**
- **We will describe a selection method called Geometric selection**
  - Several methods are available: Roulette Wheel, Tournament Selection…
- **Geometric selection**
  - The probability of selecting the $r^{th}$ best individual is given by the geometric probability mass function $P(r) = q(1-q)^{r-1}$
    - q is the probability of selecting the best individual (0.05 is a reasonable value)
  - The geometric distribution assigns higher probability to individuals ranked better, but also allows unfit individuals to be selected
- **In addition, it is typical to carry the best individual of each population to the next one**
  - This is called the Elitist Model



# GAs, parameter choices for Feature selection

- **The choice of crossover rate $P_C$ is not critical**
  - You will want a value close to 1.0 to have a large number of offspring
- **The choice of mutation rate $P_M$ is very critical**
  - An optimal choice of $P_M$ will allow the GA to explore the more promising regions while avoiding getting trapped in local minima
    - A large value (i.e., $P_M$>0.25) will not allow the search to focus on the better regions, and the GA will perform like random search
    - A small value (i.e., close to 0.0) will not allow the search to escape local minima
- **The choice of 'q', the probability of selecting the best individual is also critical**
  - An optimal value of 'q' will allow the GA to explore the most promising solution, and at the same time provide sufficient diversity to avoid early convergence of the algorithm
- **In general, poorly selected control parameters will result in sub-optimal solutions due to early convergence**

# Search Strategies, summary

| | Accuracy | Complexity | Advantages | Disadvantages |
|---|---|---|---|---|
| **Exhaustive** | Always finds the optimal solution | Exponential | High accuracy | High complexity |
| **Sequential** | Good if no backtracking needed | Quadratic $O(N_{EX}^2)$ | Simple and fast | Cannot backtrack |
| **Randomized** | Good with proper control parameters | Generally low | Designed to escape local minima | Difficult to choose good parameters |

- **A highly recommended review of this material**

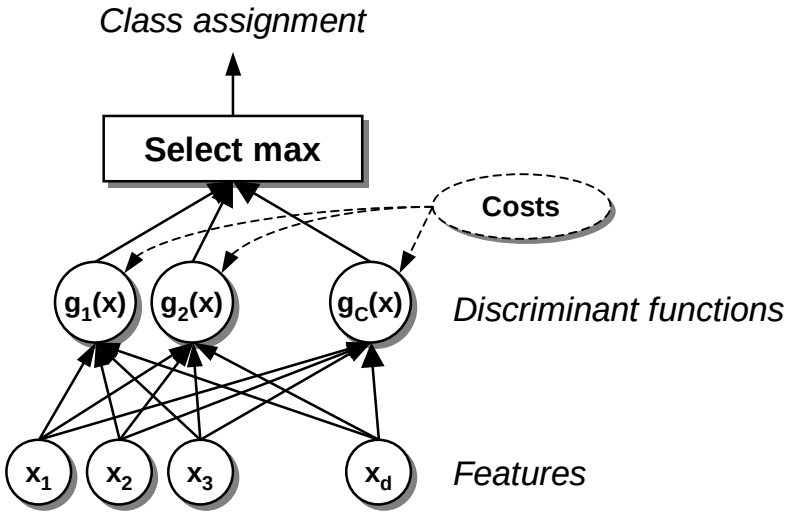
Justin Doak
"An evaluation of feature selection methods and their application to Computer Security"
University of California at Davis, Tech Report CSE-92-18

# SECTION III: Classification

- Discriminant functions
- The optimal Bayes classifier
- Quadratic classifiers
- Euclidean and Mahalanobis metrics
- K Nearest Neighbor Classifiers


# Discriminant functions

- **A convenient way to represent a pattern classifier is in terms of a family of discriminant functions $g_i(x)$ with a simple MAX gate as the classification rule**




**Assign x to class $\omega_i$ if $g_i(x) > g_j(x)\ \forall j \neq i$**

- **How do we choose the discriminant functions $g_i(x)$**
  - Depends on the objective function to minimize
    - Probability of error
    - Bayes Risk



# Minimizing probability of error

- **Probability of error P[error|x] is "the probability of assigning x to the wrong class"**
  - For a two-class problem, P[error|x] is simply

$$P(\text{error}\,|\,x) = \begin{cases} P(\omega_1\,|\,x) & \text{if we decide } \omega_2 \\ P(\omega_2\,|\,x) & \text{if we decide } \omega_1 \end{cases}$$

    - It makes sense that the classification rule be designed to minimize the average probability of error P[error] across all possible values of x

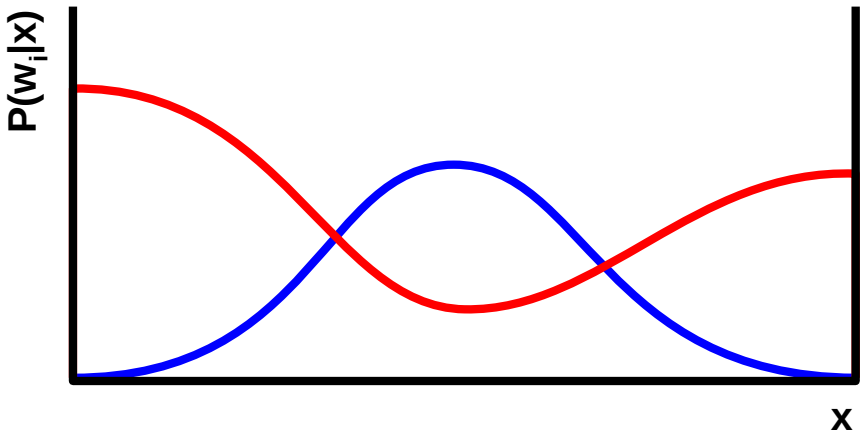
$$P(\text{error}) = \int_{-\infty}^{+\infty} P(\text{error}, x)\,dx = \int_{-\infty}^{+\infty} P(\text{error}\,|\,x)P(x)\,dx$$

    - To ensure P(error) is minimum we minimize P(error|x) by choosing the class with maximum posterior $P(\omega_i|x)$ at each x
  - This is called the **MAXIMUM A POSTERIORI (MAP) RULE**
    - And the associated discriminant functions become

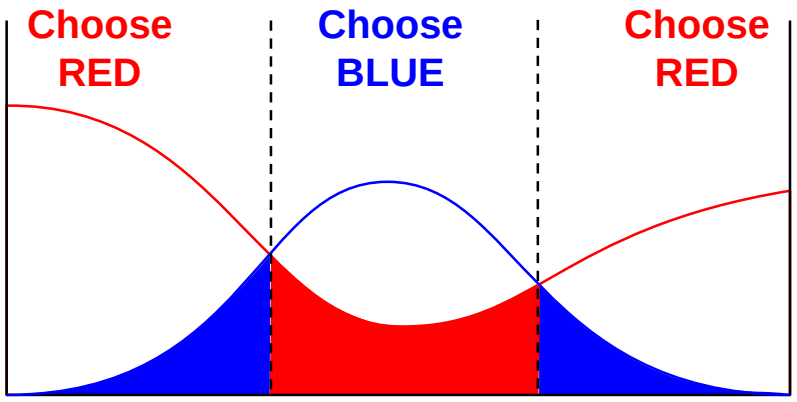
$$g_i^{MAP}(x) = P(\omega_i\,|\,x)$$

# Minimizing probability of error

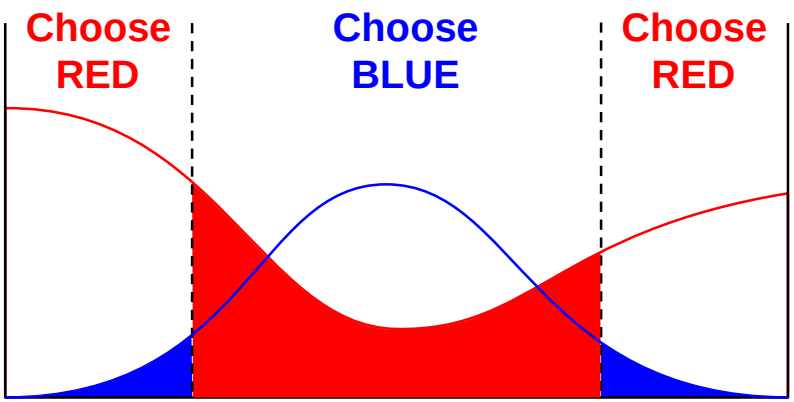
- **We "prove" the optimality of the MAP rule graphically**
  - The right plot shows the posterior for each of the two classes
  - The bottom plots shows the P(error) for the MAP rule and another rule
  - Which one has lower P(error) (color-filled area) ?




**THE MAP RULE**




**THE "OTHER" RULE**



# Quadratic classifiers

- **Let us assume that the likelihood densities are Gaussian**

$$P(x \mid \omega_i) = \frac{1}{(2\pi)^{n/2}\left|\Sigma_i\right|^{1/2}} \exp\left[-\frac{1}{2}(x-\mu_i)^T \Sigma_i^{-1}(x-\mu_i)\right]$$

- **Using Bayes rule, the MAP discriminant functions become**

$$g_i(x) = P(\omega_i \mid x) = \frac{P(x \mid \omega_i)P(\omega_i)}{P(x)} = \frac{1}{(2\pi)^{n/2}\left|\Sigma_i\right|^{1/2}} \exp\left[-\frac{1}{2}(x-\mu_i)^T \Sigma_i^{-1}(x-\mu_i)\right] P(\omega_i)\frac{1}{P(x)}$$

  - Eliminating constant terms

$$g_i(x) = \left|\Sigma_i\right|^{-1/2} \exp\left[-\frac{1}{2}(x-\mu_i)^T \Sigma_i^{-1}(x-\mu_i)\right] P(\omega_i)$$

  - We take natural logs (the logarithm is monotonically increasing)

$$g_i(x) = -\frac{1}{2}(x-\mu_i)^T \Sigma_i^{-1}(x-\mu_i) - \frac{1}{2}\log\left(\left|\Sigma_i\right|\right) + \log\left(P(\omega_i)\right)$$

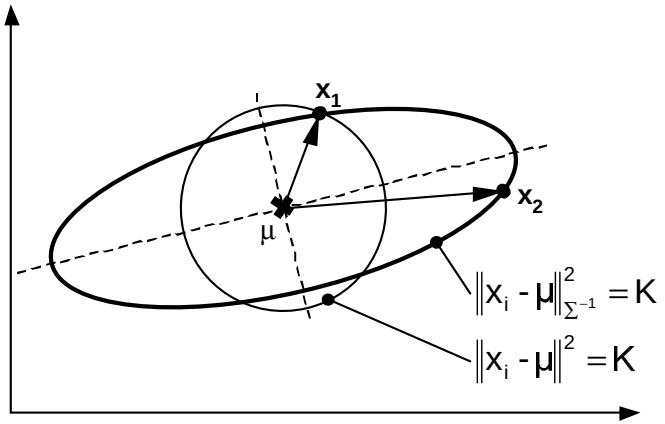
    - This is known as a **Quadratic Discriminant Function**
    - The quadratic term is know as the **Mahalanobis distance**

# Mahalanobis distance

- **The Mahalanobis distance can be thought of vector distance that uses a $\sum_i^{-1}$ norm**

**Mahalanobis Distance**

$$\|x - y\|^2_{\sum_i^{-1}} = (x-y)^T \sum_i^{-1} (x-y)$$



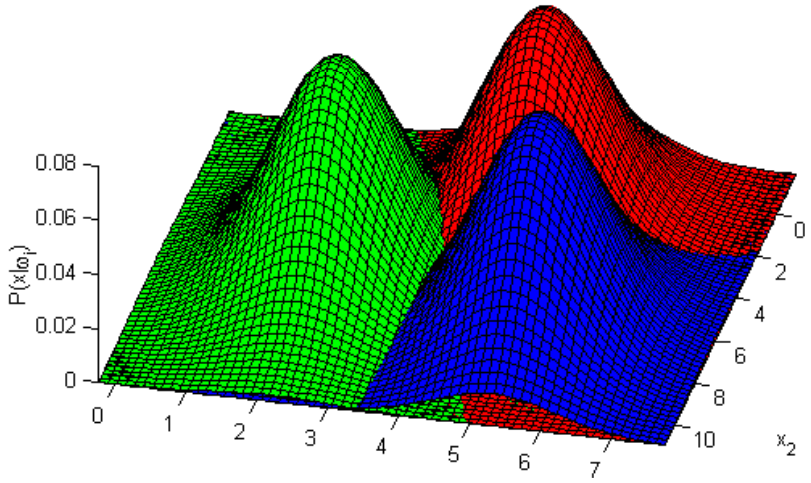

  - $\sum^{-1}$ can be thought of as a stretching factor on the space
  - Note that for an identity covariance matrix ($\sum_i = I$), the Mahalanobis distance becomes the familiar **Euclidean distance**
- **In the following slides we look at special cases of the Quadratic classifier**
  - For convenience we will assume equiprobable priors so we can drop the term $\log(P(\omega_i))$

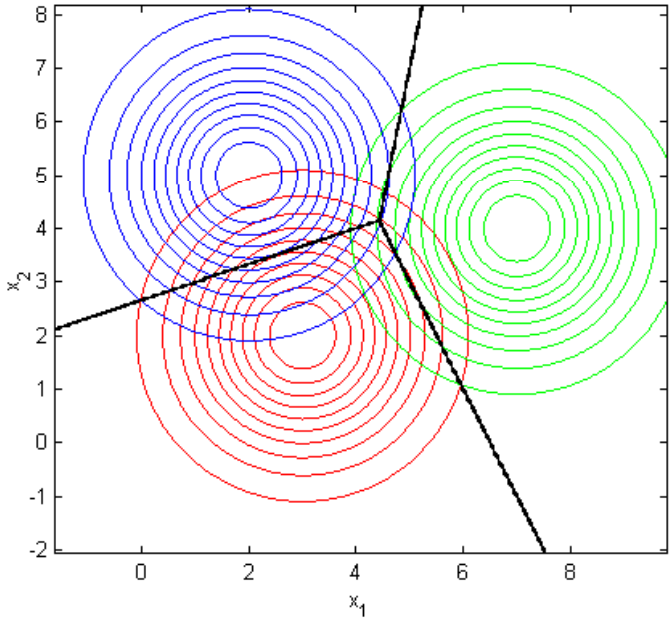
# Special case I: $\Sigma_i=\sigma^2 I$

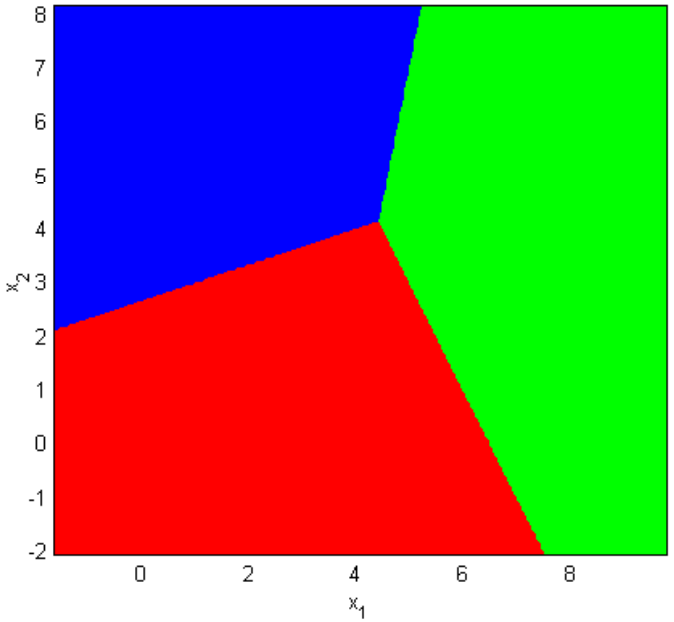
- **In this case, the discriminant becomes**

$$g_i(x) = -(x-\mu_i)^T(x-\mu_i)$$

  - This is known as a **MINIMUM DISTANCE CLASSIFIER**
  - Notice the linear decision boundaries



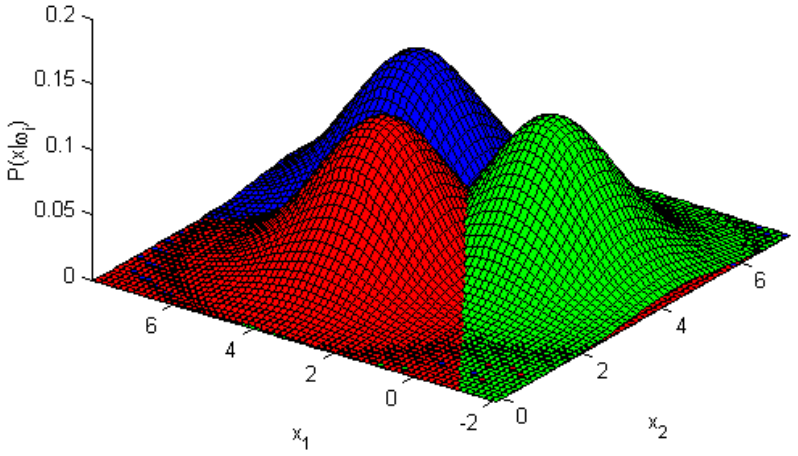


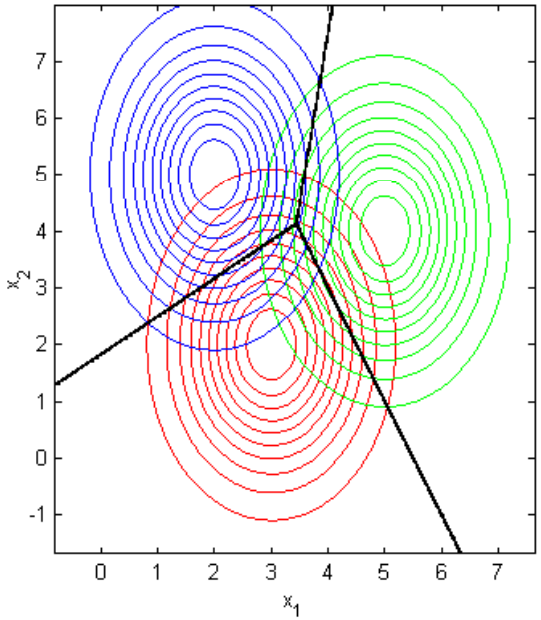

# Special case 2: $\Sigma_i = \Sigma$ ($\Sigma$ diagonal)

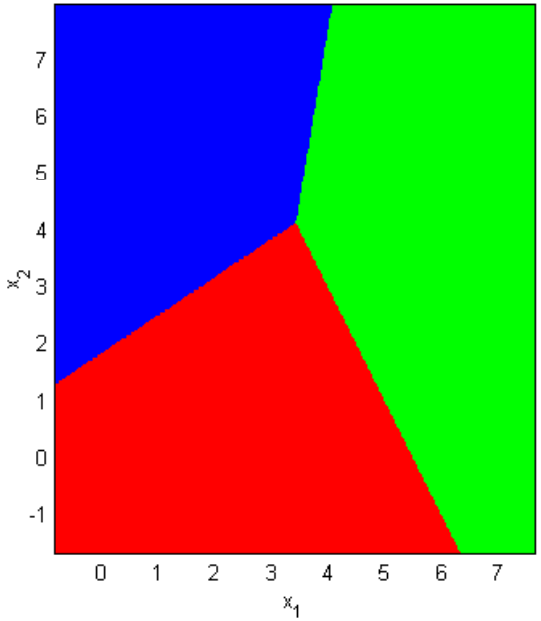
- **In this case, the discriminant becomes**

$$g_i(x) = -\frac{1}{2}(x-\mu_i)^T \Sigma^{-1}(x-\mu_i)$$

  - This is known as a **MAHALANOBIS DISTANCE CLASSIFIER**
  - Still linear decision boundaries



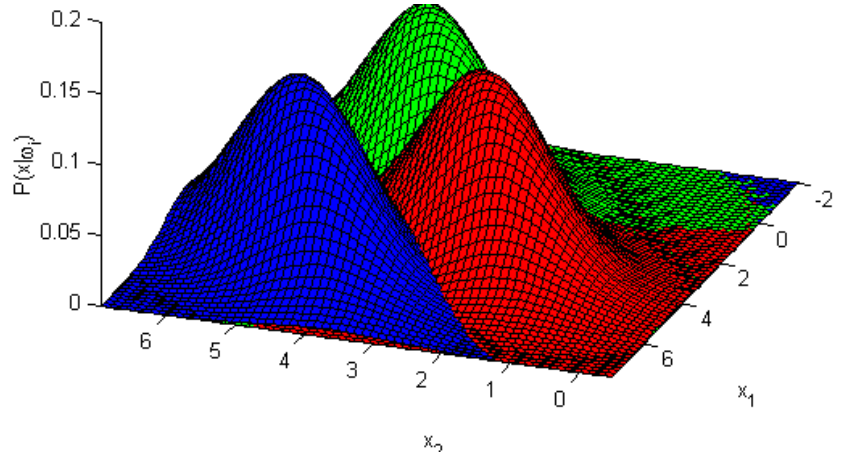


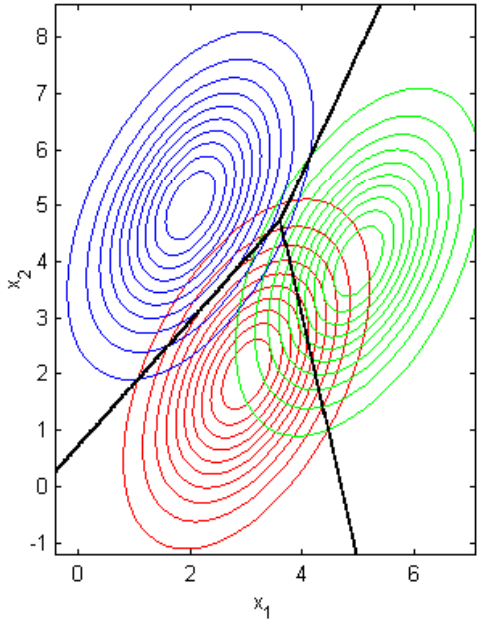

# *Special case 3:* $\Sigma_i=\Sigma$ ($\Sigma$ non-diagonal)

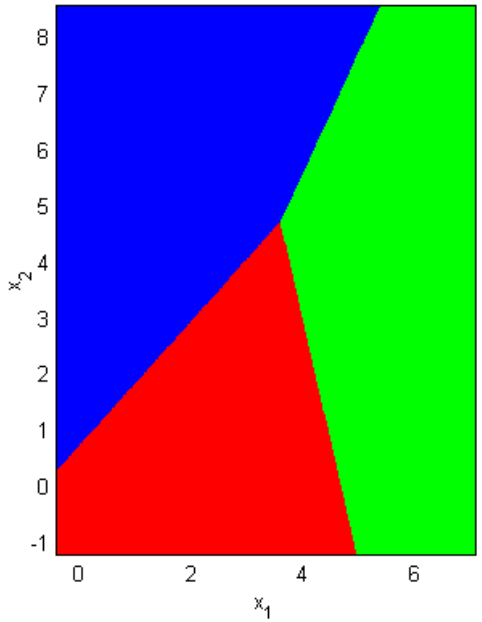
- **In this case, the discriminant becomes**

$$g_i(x) = -\frac{1}{2}(x-\mu_i)^T \Sigma_i^{-1}(x-\mu_i)$$

  - This is also known as a **MAHALANOBIS DISTANCE CLASSIFIER**
  - Still linear decision boundaries

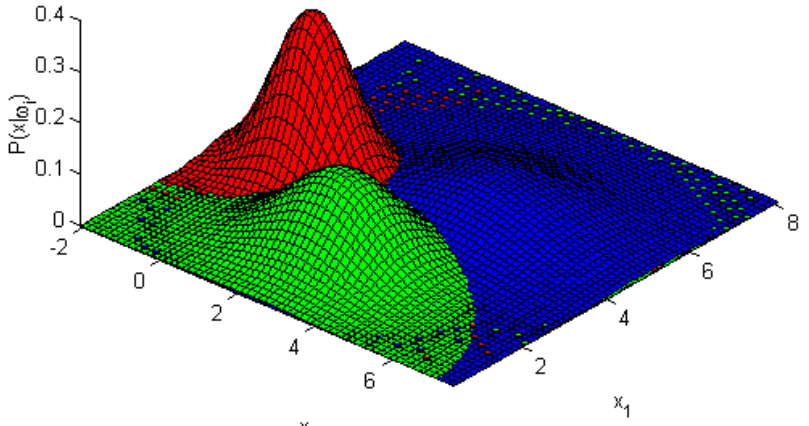


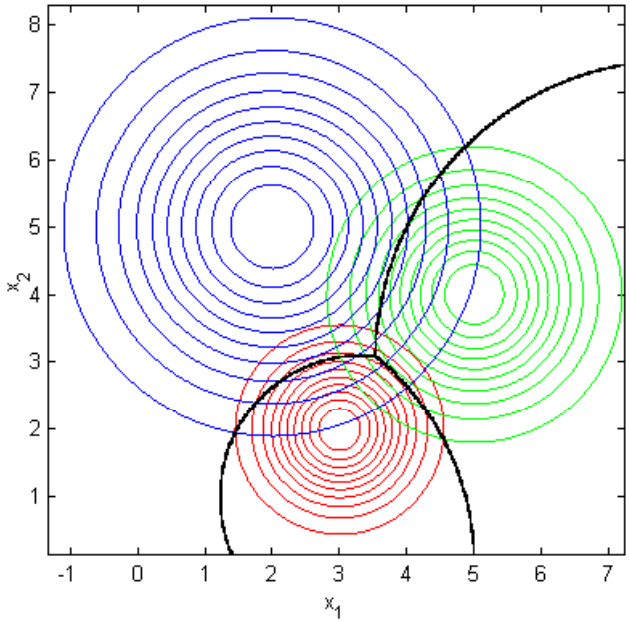



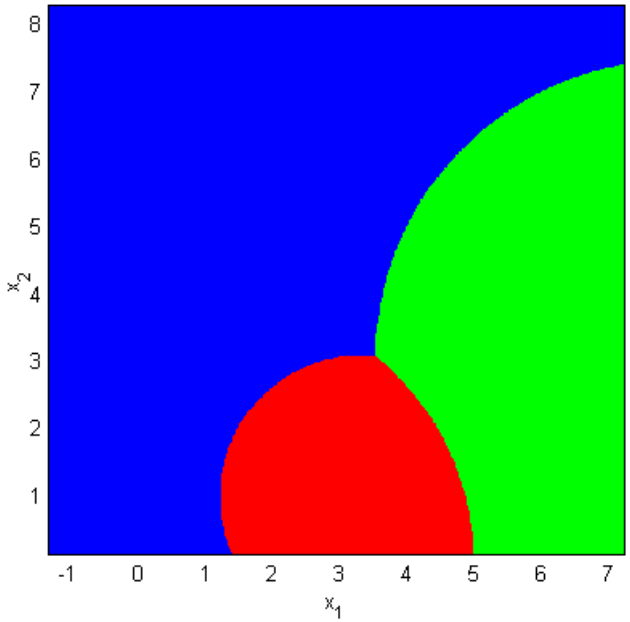
# *Case 4:* $\Sigma_i=\sigma_i^2 I$, example

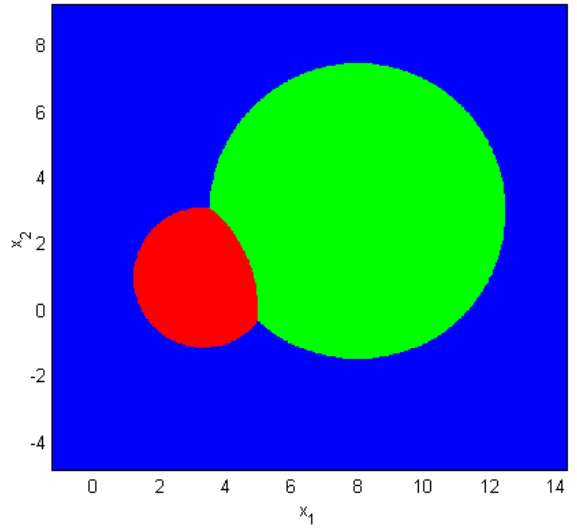
- In this case the quadratic expression cannot be simplified any further
- Notice that the decision boundaries are no longer linear but quadratic




Zoom out



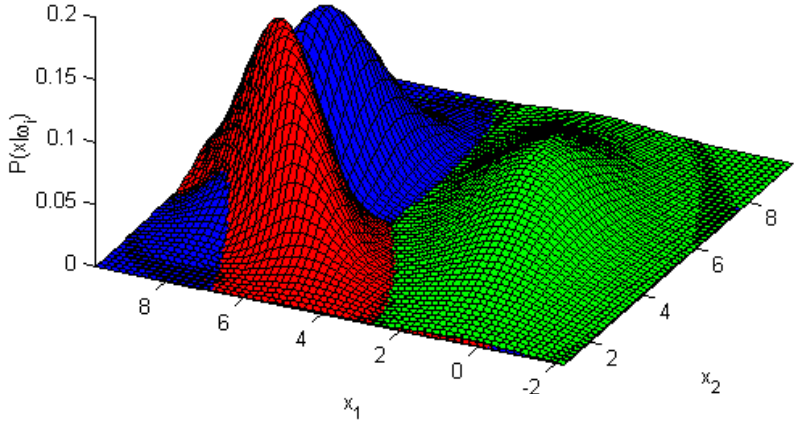



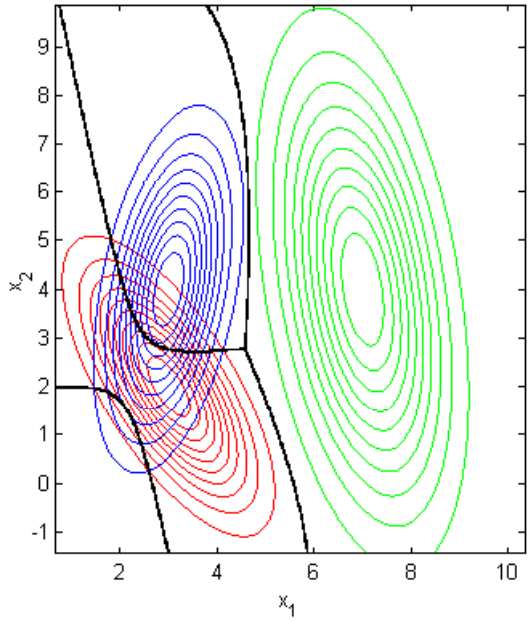



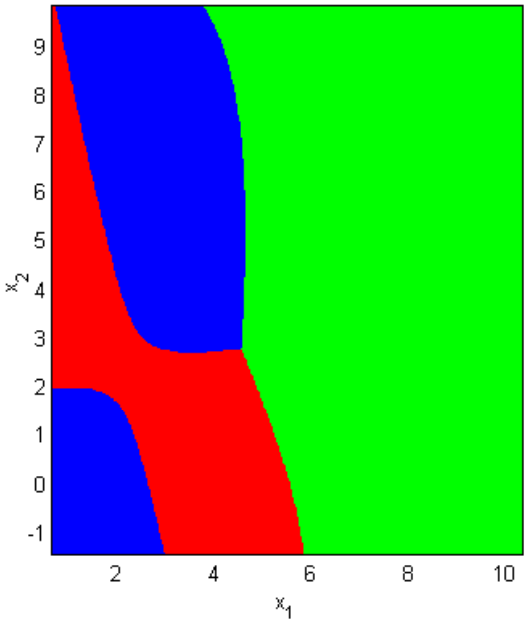



# *Case 5:* $\Sigma_i \neq \Sigma_j$ general case, example

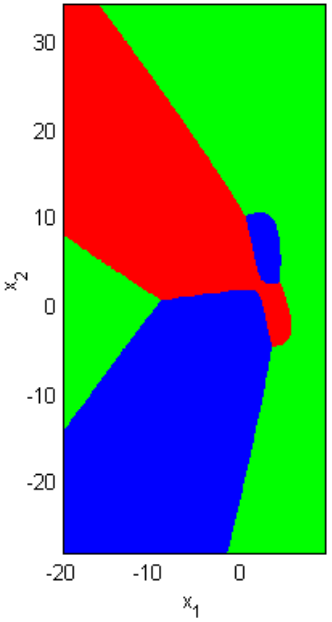
- In this case there are no constraints so the quadratic expression cannot be simplified any further
- Notice that the decision boundaries are also quadratic



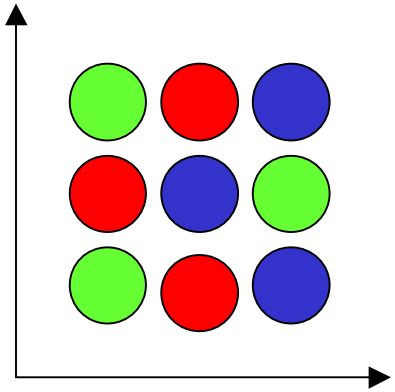

Zoom out






# Limitations of quadratic classifiers

- **The fundamental limitation is the unimodal Gaussian assumption**
  - For non-Gaussian or multimodal Gaussian, the results may be significantly sub-optimal



- **A practical limitation is associated with the minimum required size for the dataset**
  - If the number of examples per class is less than the number of dimensions, the covariance matrix becomes singular and, therefore, its inverse cannot be computed
    - In this case it is common to assume the same covariance structure for all classes and compute the covariance matrix using all the examples, regardless of class

# Conclusions

- **We can extract the following conclusions**
  - The Bayes classifier for normally distributed classes is quadratic
  - The Bayes classifier for normally distributed classes with equal covariance matrices is a linear classifier
  - The minimum Mahalanobis distance classifier is optimum for
    - normally distributed classes and equal covariance matrices and equal priors
  - The minimum Euclidean distance classifier is optimum for
    - normally distributed classes and equal covariance matrices proportional to the identity matrix and equal priors
  - Both Euclidean and Mahalanobis distance classifiers are linear
- **The goal of this discussion was to show that some of the most popular classifiers can be derived from decision-theoretic principles and some simplifying assumptions**
  - It is important to realize that using a specific (Euclidean or Mahalanobis) minimum distance classifier implicitly corresponds to certain statistical assumptions
  - The question whether these assumptions hold or don't can rarely be answered in practice; in most cases we are limited to posting and answering the question "*does this classifier solve our problem or not?*"

# K Nearest Neighbor classifier

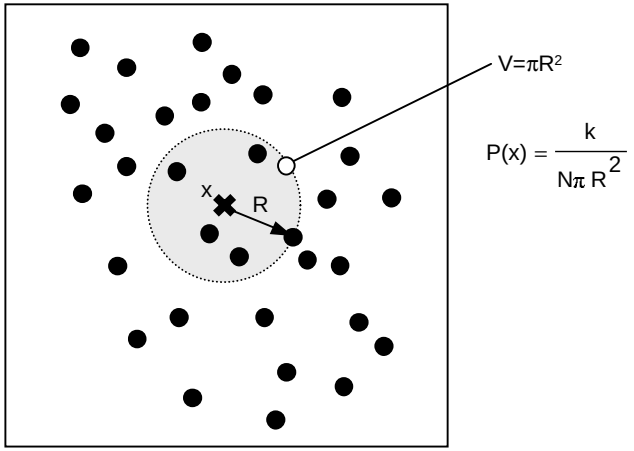
- **The kNN classifier is based on non-parametric density estimation techniques**
  - Let us assume we seek to estimate the density function P(x) from a dataset of examples
  - P(x) can be approximated by the expression

$$P(x) \cong \frac{k}{NV} \quad \text{where} \quad \begin{cases} V \text{ is the volume surrounding } x \\ N \text{ is the total number of examples} \\ k \text{ is the number of examples inside } V \end{cases}$$

  - The volume V is determined by the D-dim distance $R_k^D(x)$ between x and its k nearest neighbor

$$P(x) \cong \frac{k}{NV} = \frac{k}{N \cdot c_D \cdot R_k^D(x)}$$

    - Where $c_D$ is the volume of the unit sphere in D dimensions



# K Nearest Neighbor classifier

- **We use the previous result to estimate the posterior probability**
  - The unconditional density is, again, estimated with

$$P(x|\omega_i) = \frac{k_i}{N_i V}$$

  - And the priors can be estimated by

$$P(\omega_i) = \frac{N_i}{N}$$

  - The posterior probability then becomes

$$P(\omega_i|x) = \frac{P(x|\omega_i)P(\omega_i)}{P(x)} = \frac{\frac{k_i}{N_i V}\cdot\frac{N_i}{N}}{\frac{k}{NV}} = \frac{k_i}{k}$$

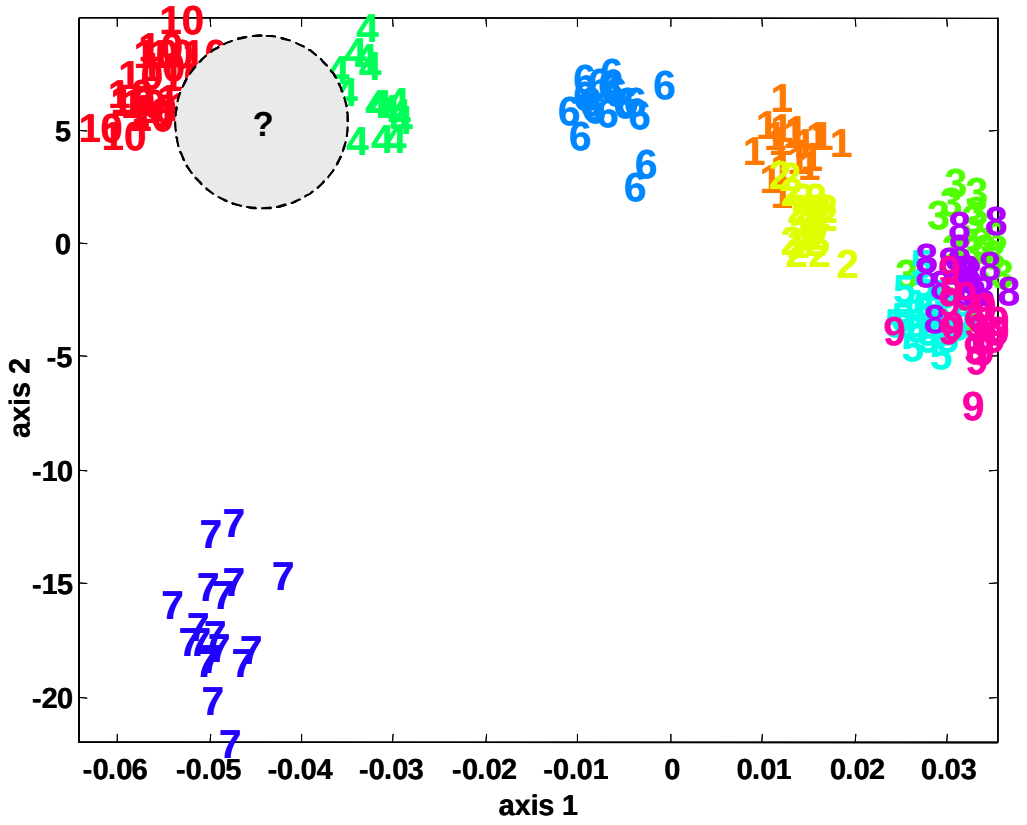
  - Yielding discriminant functions

$$\boxed{g_i(x) = \frac{k_i}{k}}$$

    - This is known as the k Nearest Neighbor classifier

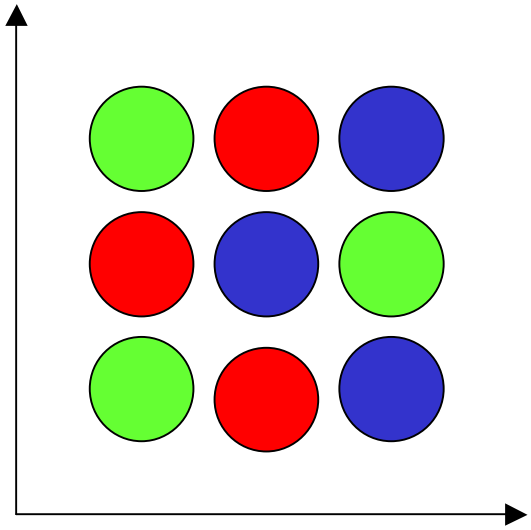
# K Nearest Neighbor classifier

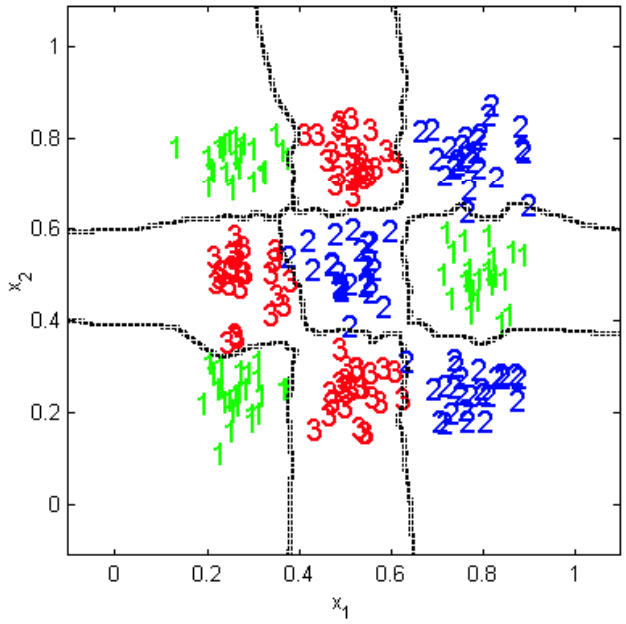
- **The kNN classifier is a very intuitive method**
  - Examples are classified based on their similarity with training data
    - For a given unlabeled example $x_u \in \Re^D$, find the k "closest" labeled examples in the training data set and assign $x_u$ to the class that appears most frequently within the k-subset
- **The kNN only requires**
  - An integer k
  - A set of labeled examples
  - A measure of "closeness"

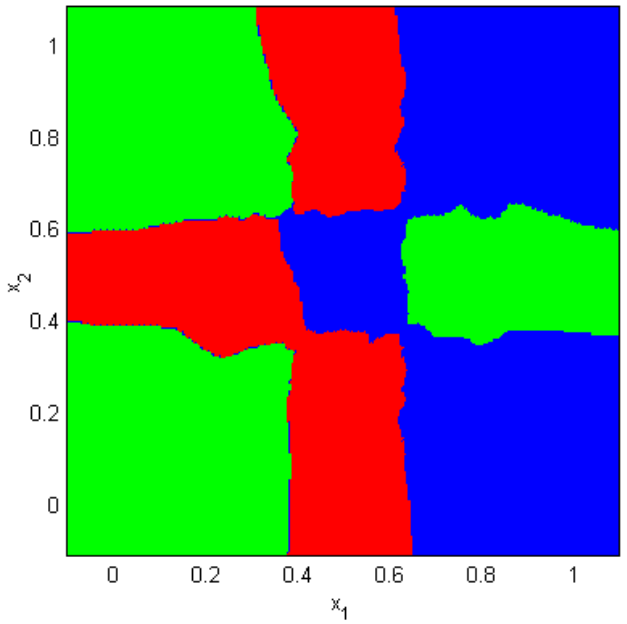


# kNN in action: example 1

- **We generate data for a 2-dimensional 3-class problem, where the class-conditional densities are multi-modal, and non-linearly separable**
- **We used kNN with**
  - k = five
  - Metric = Euclidean distance

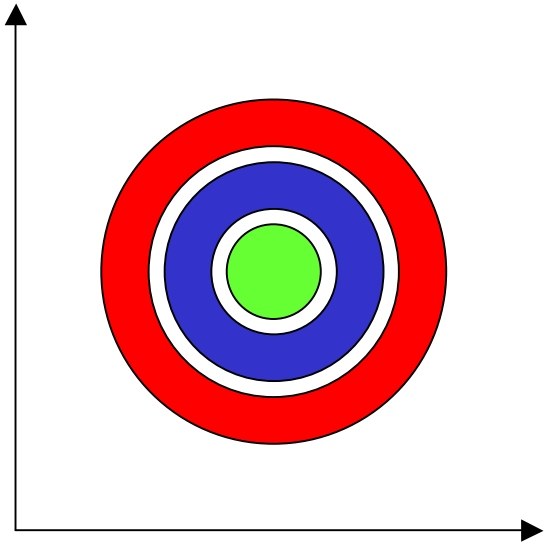


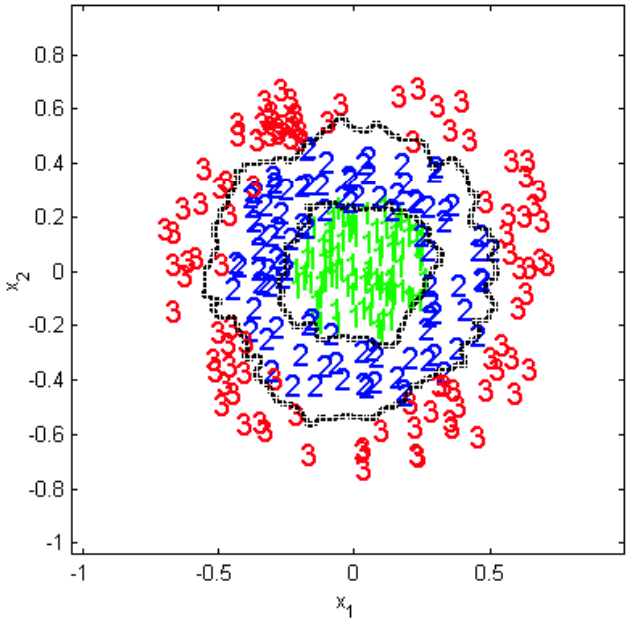


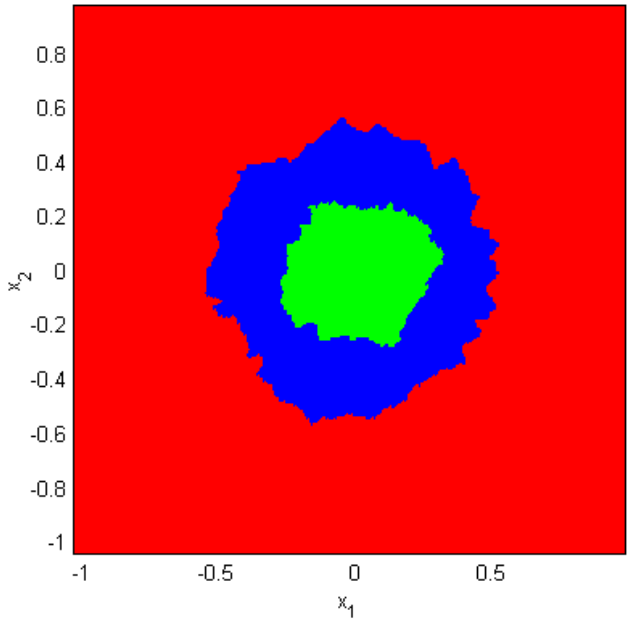



# kNN in action: example 2

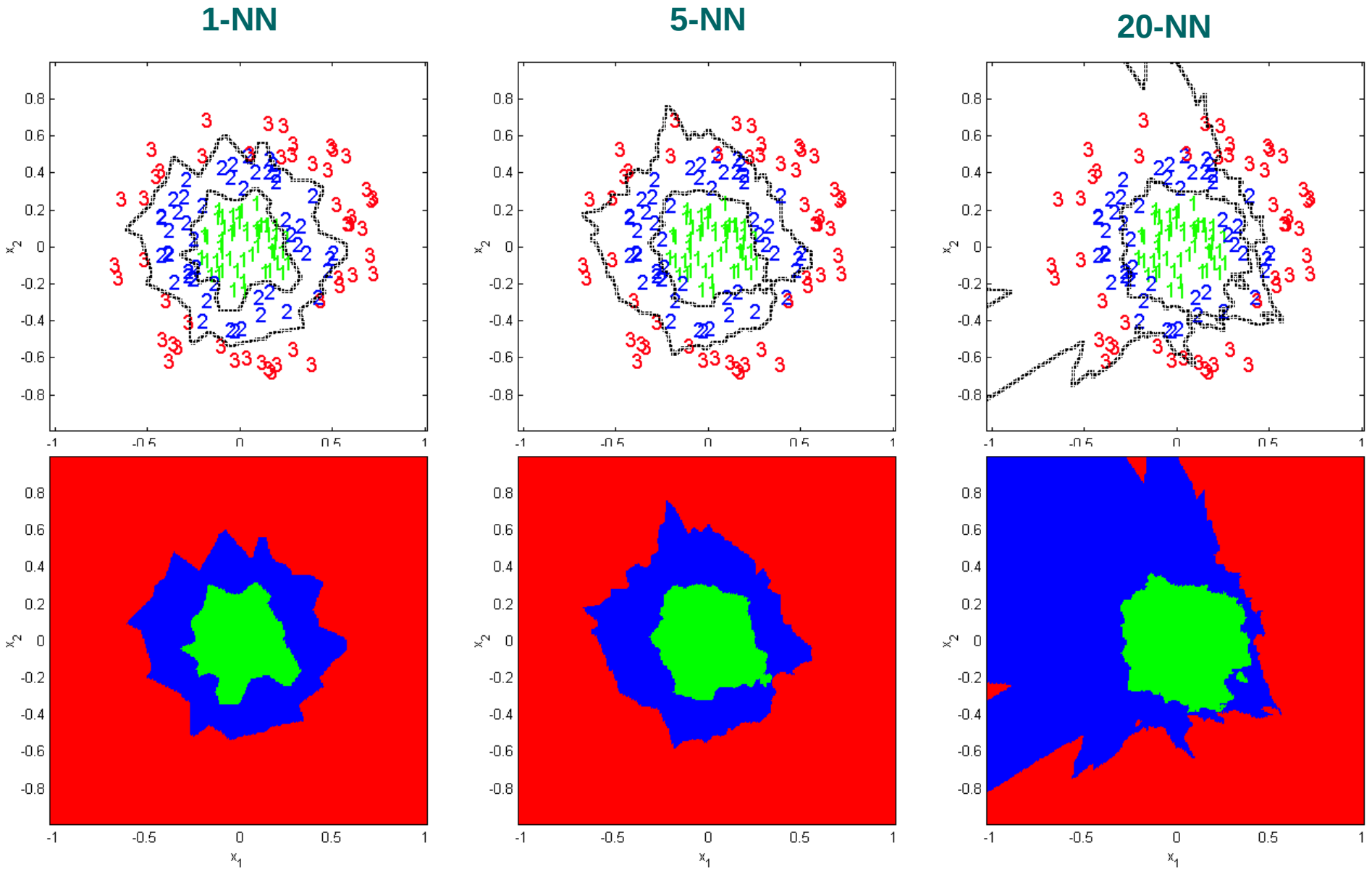
- **We generate data for a 2-dim 3-class problem, where the likelihoods are unimodal, and are distributed in rings around a common mean**
  - These classes are also non-linearly separable
- **We used kNN with**
  - k = five
  - Metric = Euclidean distance





# kNN versus 1NN



# Characteristics of the kNN classifier

- **Advantages**
  - Analytically tractable, simple implementation
  - Nearly optimal in the large sample limit ($N \rightarrow \infty$)
    - $P[error]_{Bayes} < P[error]_{1\text{-}NNR} < 2P[error]_{Bayes}$
  - Uses local information, which can yield highly adaptive behavior
  - Lends itself very easily to parallel implementations
- **Disadvantages**
  - Large storage requirements
  - Computationally intensive recall
  - Highly susceptible to the curse of dimensionality
- **1NN versus kNN**
  - The use of large values of k has two main advantages
    - Yields smoother decision regions
    - Provides probabilistic information: The ratio of examples for each class gives information about the ambiguity of the decision
  - However, too large values of k are detrimental
    - It destroys the locality of the estimation



# Section IV: Validation

- Motivation
- The Holdout
- Re-sampling techniques
- Three-way data splits

# Motivation

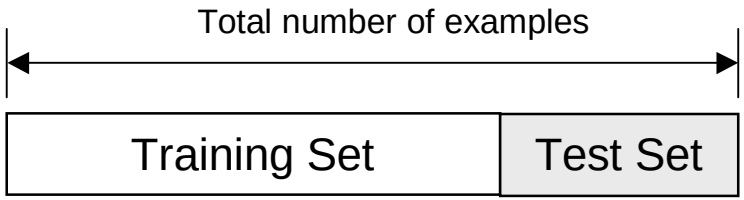
- **Validation techniques are motivated by two fundamental problems in pattern recognition: model selection and performance estimation**
- **Model selection**
  - Almost invariably, all pattern recognition techniques have one or more free parameters
    - The number of neighbors in a kNN classification rule
    - The network size, learning parameters and weights in MLPs
  - How do we select the “optimal” parameter(s) or model for a given classification problem?
- **Performance estimation**
  - Once we have chosen a model, how do we estimate its performance?
    - Performance is typically measured by the TRUE ERROR RATE, the classifier’s error rate on the ENTIRE POPULATION

# Motivation

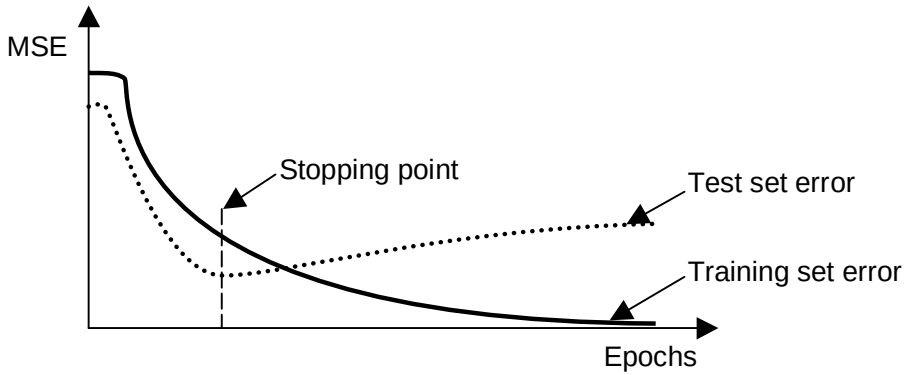
- **If we had access to an unlimited number of examples these questions have a straightforward answer**
  - Choose the model that provides the lowest error rate on the entire population and, of course, that error rate is the true error rate
- **In real applications we only have access to a finite set of examples, usually smaller than we wanted**
  - One approach is to use the entire training data to select our classifier and estimate the error rate
    - This naïve approach has two fundamental problems
      - The final model will normally overfit the training data
        - This problem is more pronounced with models that have a large number of parameters
      - The error rate estimate will be overly optimistic (lower than the true error rate)
        - In fact, it is not uncommon to have 100% correct classification on training data
  - A much better approach is to split the training data into disjoint subsets: the holdout method

# The holdout method

- **Split dataset into two groups**
  - Training set: used to train the classifier
  - Test set: used to estimate the error rate of the trained classifier




- **A typical application the holdout method is determining a stopping point for the back propagation error**



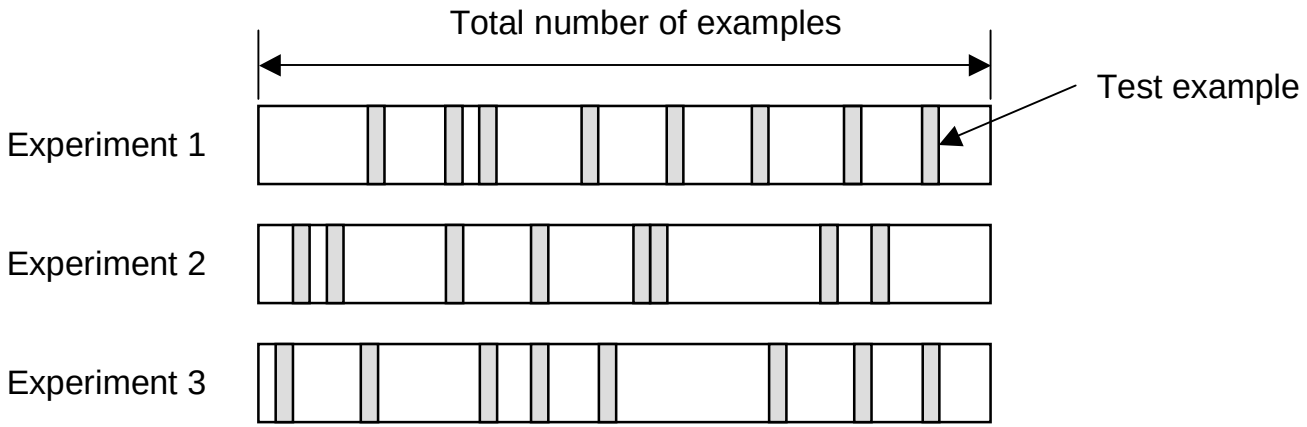



# The holdout method

- **The holdout method has two basic drawbacks**
  - In problems where we have a sparse dataset we may not be able to afford the “luxury” of setting aside a portion of the dataset for testing
  - Since it is a single train-and-test experiment, the holdout estimate of error rate will be misleading if we happen to get an “unfortunate” split
- **The limitations of the holdout can be overcome with a family of resampling methods at the expense of more computations**
  - Cross Validation
    - Random Subsampling
    - K-Fold Cross-Validation
    - Leave-one-out Cross-Validation
  - Bootstrap

# Random Subsampling

- **Random Subsampling performs K data splits of the dataset**
  - Each split randomly selects a (fixed) no. examples without replacement
  - For each data split we retrain the classifier from scratch with the training examples and estimate $E_i$ with the test examples



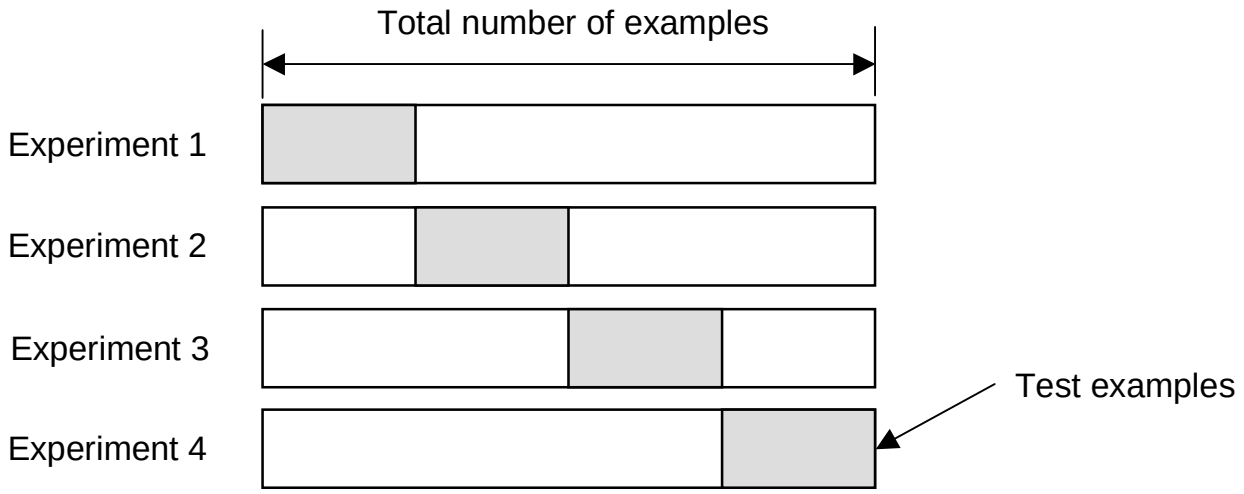

- **The true error estimate is obtained as the average of the separate estimates $E_i$**
  - This estimate is significantly better than the holdout estimate

$$E = \frac{1}{K}\sum_{i=1}^{K} E_i$$



# K-Fold Cross-validation

- **Create a K-fold partition of the the dataset**
  - For each of K experiments, use K-1 folds for training and the remaining one for testing

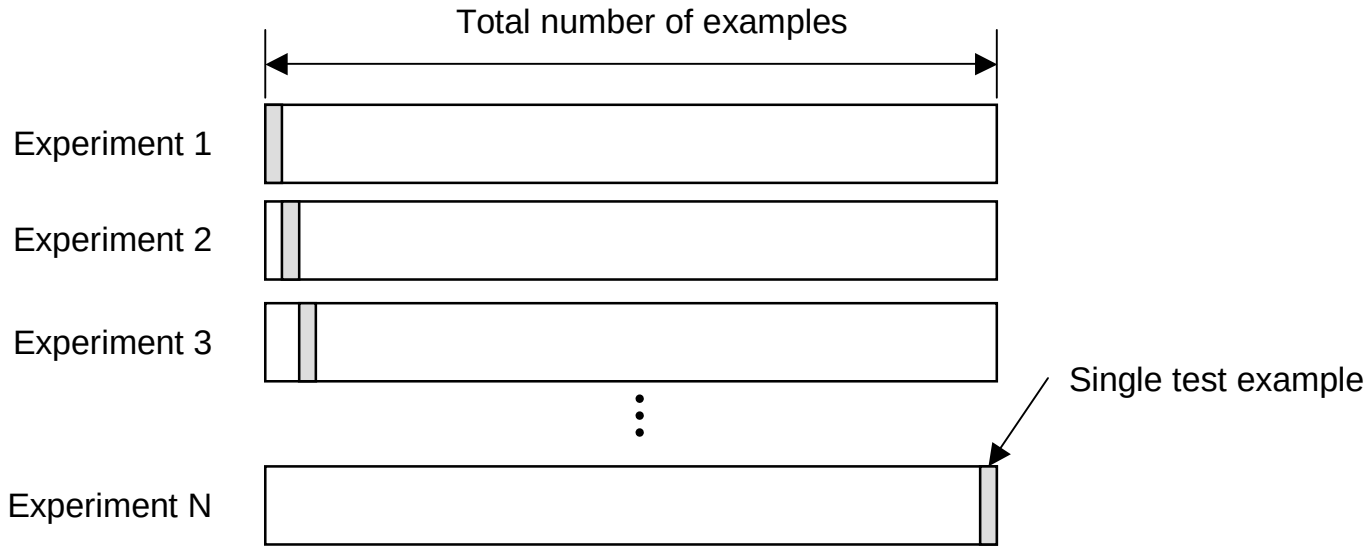



- **K-Fold Cross validation is similar to Random Subsampling**
  - The advantage of K-Fold Cross validation is that all the examples in the dataset are eventually used for both training and testing
- **As before, the true error is estimated as the average error rate**

$$E = \frac{1}{K}\sum_{i=1}^{K} E_i$$

# Leave-one-out Cross Validation

- **Leave-one-out is the degenerate case of K-Fold Cross Validation, where K is chosen as the total number of examples**
  - For a dataset with N examples, perform N experiments
  - For each experiment use N-1 examples for training and the remaining example for testing




- **As usual, the true error is estimated as the average error rate on test examples**

$$E = \frac{1}{N}\sum_{i=1}^{N} E_i$$

# How many folds are needed?

- **With a large number of folds**
  - \+ The bias of the true error rate estimator will be small (the estimator will be very accurate)
  - \- The variance of the true error rate estimator will be large
  - \- The computational time will be very large as well (many experiments)
- **With a small number of folds**
  - \+ The number of experiments and, therefore, computation time are reduced
  - \+ The variance of the estimator will be small
  - \- The bias of the estimator will be large (conservative or larger than the true error rate)
- **In practice, the choice of the number of folds depends on the size of the dataset**
  - For large datasets, even 3-Fold Cross Validation will be quite accurate
  - For very sparse datasets, we may have to use leave-one-out in order to train on as many examples as possible
- **A common choice for K-Fold Cross Validation is K=10**

# Three-way data splits

- **If model selection and true error estimates are to be computed simultaneously, the data needs to be divided into three disjoint sets**
  - **Training set**: a set of examples used for learning the parameters of the model
    - In the MLP case, we would use the training set to find the “optimal” weights with the back propagation rule
  - **Validation set**: a set of examples used to tune the meta-parameters of a model
    - In the MLP case, we would use the validation set to find the “optimal” number of hidden units or determine a stopping point for the back propagation algorithm
  - **Test set**: a set of examples used only to assess the performance of a fully-trained classifier
    - In the MLP case, we would use the test to estimate the error rate after we have chosen the final model (MLP size and actual weights)
    - After assessing the final model with the test set, YOU MUST NOT further tune the model

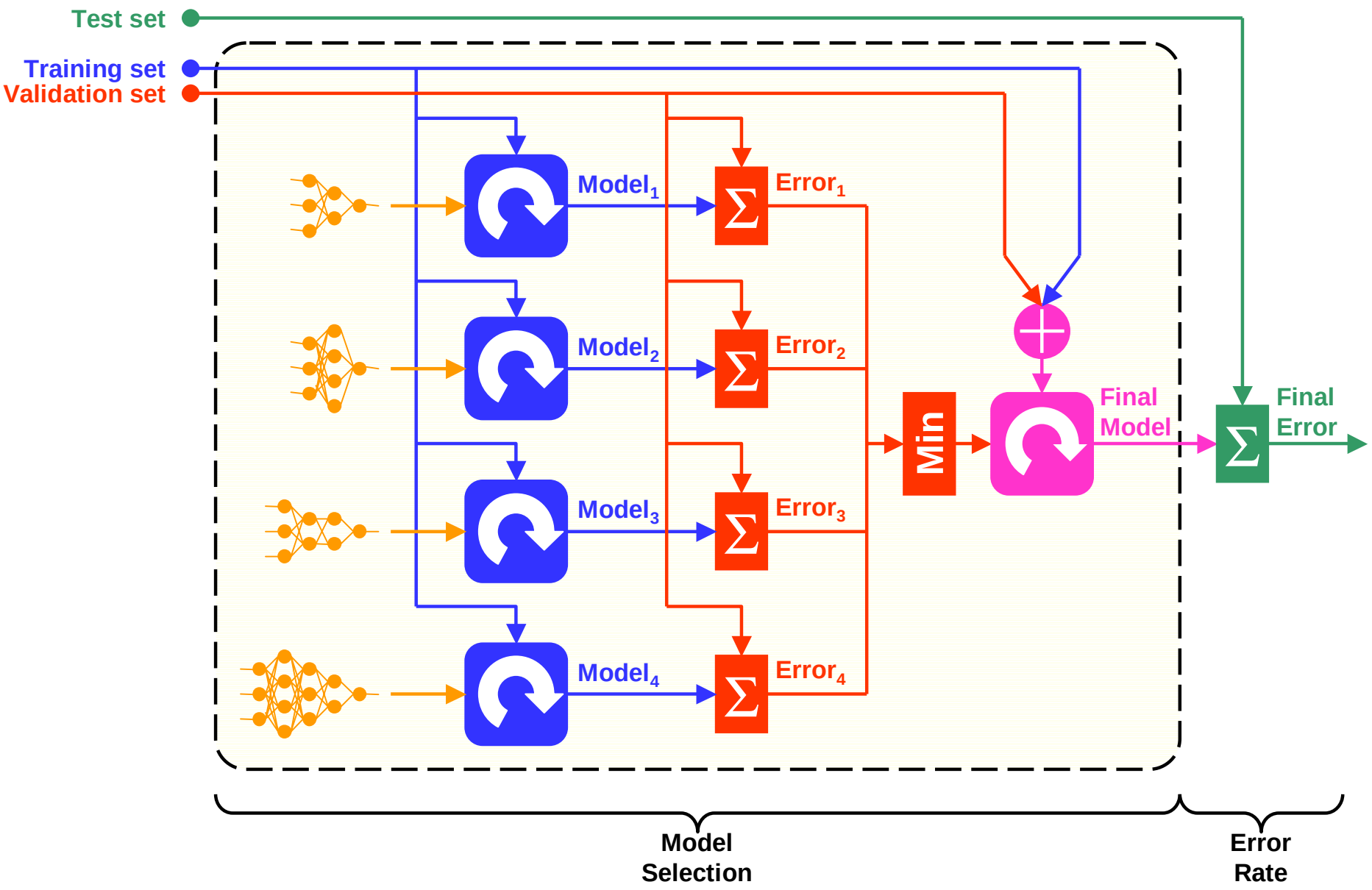
# Three-way data splits

- **Why separate test and validation sets?**
  - The error rate estimate of the final model on validation data will be biased (smaller than the true error rate) since the validation set is used to select the final model
  - After assessing the final model with the test set, YOU MUST NOT tune the model any further
- **Procedure outline**

1. Divide the available data into training, validation and test set
2. Select architecture and training parameters
3. Train the model using the training set
4. Evaluate the model using the validation set
5. Repeat steps 2 through 4 using different architectures and training parameters
6. Select the best model and train it using data from the training and validation set
7. Assess this final model using the test set

  - This outline assumes a holdout method
    - If CV or Bootstrap are used, steps 3 and 4 have to be repeated for each of the K folds

# Three-way data splits



# Section V: Clustering

- Unsupervised learning concepts
- K-means
- Agglomerative clustering
- Divisive clustering



# Unsupervised learning

- **In this section we investigate a family of techniques which use unlabeled data: the feature vector X without the class label $\omega$**
  - These methods are called unsupervised since they are not provided the correct answer
- **Although unsupervised learning methods may appear to have limited capabilities, there are several reasons that make them extremely useful**
  - Labeling large data sets can be costly (i.e., speech recognition)
  - Class labels may not be known beforehand (i.e., data mining)
  - Very large datasets can be compressed by finding a small set of prototypes (i.e., kNN)
- **Two major approaches for unsupervised learning**
  - Parametric (mixture modeling)
    - A functional form for the underlying density is assumed, and we seek to estimate the parameters of the model
  - Non-parametric (clustering)
    - No assumptions are made about the underlying densities, instead we seek a partition of the data into clusters
      - These methods are typically referred to as clustering

# Unsupervised learning

- **Non-parametric unsupervised learning involves three steps**
  - Defining a measure of similarity between examples
  - Defining a criterion function for clustering
  - Defining an algorithm to minimize (or maximize) the criterion function
- **Similarity measures**
  - The most straightforward similarity measure is distance: Euclidean, city block or Mahalanobis
    - Euclidean and city block measures are very sensitive to scaling of the axis, so caution must be exercised
- **Criterion function**
  - The most widely used criterion function for clustering is the sum-of-square-error criterion

$$J_{MSE} = \sum_{i=1}^{C} \sum_{x \approx \omega_i} |x - \mu_i|^2 \qquad \text{where } \mu_i = \frac{1}{N_i} \sum_{x \approx \omega_i} x$$

    - This criterion measures how well the data set $X=\{x^{(1}, x^{(2}, \ldots, x^{(N}\}$ is represented by the cluster centers $\mu=\{\mu^{(1}, \mu^{(2}, \ldots, \mu^{(N}\}$

# Unsupervised learning

- **Once a criterion function has been defined, we must find a partition of the data set that minimizes the criterion**
  - Exhaustive enumeration of all partitions, which guarantees the optimal solution, is unfeasible
  - For example, a problem with 5 clusters and 100 examples yields $10^{67}$ different ways to partition the data
- **The common approach is to proceed in an iterative fashion**
  - Find some reasonable initial partition and then
  - Move samples from one cluster to another in order to reduce the criterion function
- **These iterative methods produce sub-optimal solution but are computationally tractable**
- **We will consider two groups of iterative methods**
  - Flat clustering algorithms
    - These algorithms produce a set of disjoint clusters (a.k.a. a partition)
    - Two algorithms are widely used: k-means and ISODATA
  - Hierarchical clustering algorithms:
    - The result is a hierarchy of nested clusters
    - These algorithms can be broadly divided into agglomerative and divisive approaches

# The k-means algorithm

- **The k-means algorithm is a simple clustering procedure that attempts to minimize the criterion function $J_{MSE}$ iteratively**

1. Define the number of clusters
2. Initialize clusters by
   - an arbitrary assignment of examples to clusters or
   - an arbitrary set of cluster centers (examples assigned to nearest centers)
3. Compute the sample mean of each cluster
4. Reassign each example to the cluster with the nearest mean
5. If the classification of all samples has not changed, stop, else go to step 3

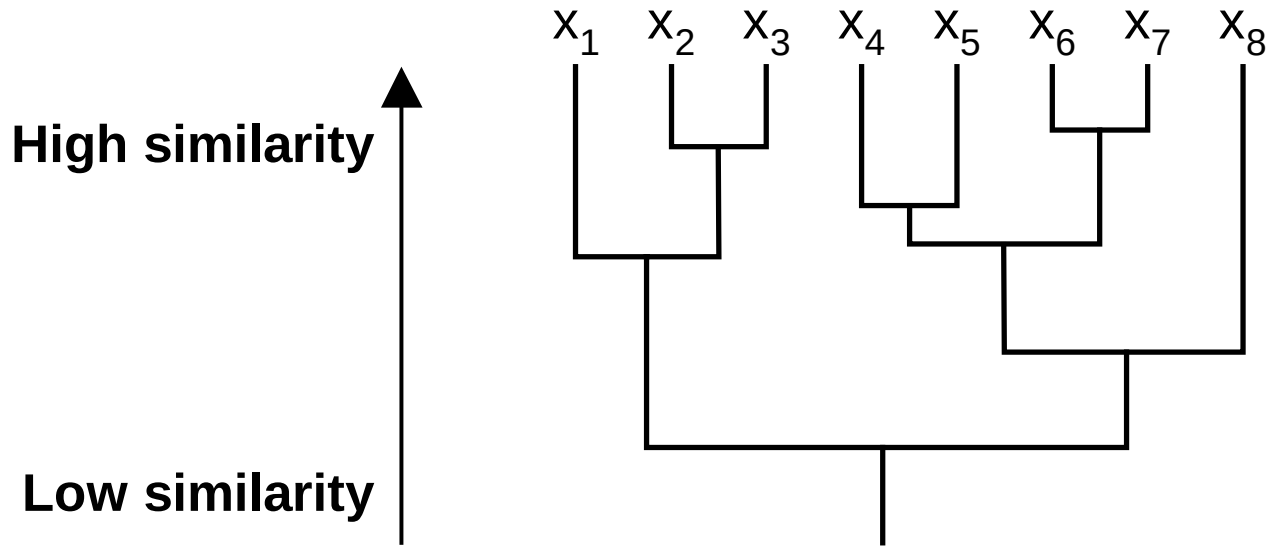
- The k-means algorithm is widely used in the fields of signal processing and communication for Vector Quantization
  - Scalar signal values are usually quantized into a number of levels (typically a power of 2 so the signal can be transmitted in binary)
  - The same idea can be extended for multiple channels
    - However, rather than quantizing each separate channel, we can obtain a more efficient signal coding if we quantize the overall multidimensional vector by finding a number of multidimensional prototypes (cluster centers)
  - The set of cluster centers is called a “codebook”, and the problem of finding this codebook is normally solved using the k-means algorithm

# Hierarchical clustering

- **k-means creates disjoint clusters, resulting in a "flat" data representation (a partition)**
- **Sometimes it is desirable to obtain a hierarchical representation of data, with clusters and sub-clusters arranged in a tree-structured fashion**
  - Hierarchical representations are commonly used in the sciences (i.e., biological taxonomy)
- **Hierarchical clustering methods can be grouped in two general classes**
  - Agglomerative (bottom-up, merging):
    - Starting with N singleton clusters, successively merge clusters until one cluster is left
  - Divisive (top-down, splitting)
    - Starting with a unique cluster, successively split the clusters until N singleton examples are left

# Hierarchical clustering

- **The preferred representation for hierarchical clusters is the dendrogram**
  - The dendrogram is a binary tree that shows the structure of the clusters
    - In addition to the binary tree, the dendrogram provides the similarity measure between clusters (the vertical axis)
  - An alternative representation is based on sets: $\{\{x_1, \{x_2, x_3\}\}, \{\{\{x_4, x_5\}, \{x_6, x_7\}\}, x_8\}\}$ but, unlike the dendrogram, sets cannot express quantitative information



# Divisive clustering

- **Outline**
  - Define
    - $N_C$: Number of clusters
    - $N_{EX}$: Number of examples

1. Start with one large cluster
2. Find “worst” cluster
3. Split it
4. If $N_C<N_{EX}$ go to 1

- **How to choose the “worst” cluster**
  - Largest number of examples
  - Largest variance
  - Largest sum-squared-error
  - ...
- **How to split clusters**
  - Mean-median in one feature direction
  - Perpendicular to the direction of largest variance
  - …
- **The computations required by divisive clustering are more intensive than for agglomerative clustering methods and, thus, agglomerative approaches are more common**

# Agglomerative clustering

- **Outline**
  - Define
    - $N_C$: Number of clusters
    - $N_{EX}$: Number of examples

1. Start with $N_{EX}$ singleton clusters
2. Find nearest clusters
3. Merge them
4. If $N_C$>1 go to 1

- **How to find the "nearest" pair of clusters**
  - Minimum distance $d_{min}(\omega_i,\omega_j)=\min_{\substack{x\in\omega_i\\y\in\omega_j}}\|x-y\|$
  - Maximum distance $d_{max}(\omega_i,\omega_j)=\max_{\substack{x\in\omega_i\\y\in\omega_j}}\|x-y\|$
  - Average distance $d_{avg}(\omega_i,\omega_j)=\frac{1}{N_iN_j}\sum_{x\in\omega_i}\sum_{y\in\omega_j}\|x-y\|$
  - Mean distance $d_{mean}(\omega_i,\omega_j)=\|\mu_i-\mu_j\|$

# Agglomerative clustering

- **Minimum distance**
  - When $d_{min}$ is used to measure distance between clusters, the algorithm is called the nearest-neighbor or single-linkage clustering algorithm
  - If the algorithm is allowed to run until only one cluster remains, the result is a minimum spanning tree (MST)
  - This algorithm favors elongated classes
- **Maximum distance**
  - When $d_{max}$ is used to measure distance between clusters, the algorithm is called the farthest-neighbor or complete-linkage clustering algorithm
  - From a graph-theoretic point of view, each cluster constitutes a complete sub-graph
  - This algorithm favors compact classes
- **Average and mean distance**
  - The minimum and maximum distance are extremely sensitive to outliers since their measurement of between-cluster distance involves minima or maxima
  - The average and mean distance approaches are more robust to outliers
  - Of the two, the mean distance is computationally more attractive
    - Notice that the average distance approach involves the computation of $N_i N_j$ distances for each pair of clusters

# Conclusions

- **Dimensionality reduction**
  - PCA may not find the most discriminatory axes
  - LDA may overfit the data
- **Classifiers**
  - Quadratic classifiers are efficient for unimodal data
  - kNN is very versatile but is computationally inefficient
- **Validation**
  - An fundamental subject, oftentimes overlooked
- **Clustering**
  - Helps understand the underlying structure of the data


## ACKNOWLEDGMENTS

**This research was supported by award from NSF/CAREER 9984426.**




# References


- Fukunaga, K., 1990, *Introduction to statistical pattern recognition*, 2nd edition, Academic Pr.
- Duda, R. O.; Hart, P. E. and Stork, D. G., 2001, *Pattern classification, 2nd Ed.*, Wiley.
- Bishop, C. M., 1995, *Neural networks for Pattern Recognition*, Oxford.
- Therrien, C. W., 1989, *Decision, estimation, and classification : an introduction to pattern recognition and related topics*, Wiley.
- Devijver, P. A. and Kittler, J., 1982, *Pattern recognition: a statistical approach*, Prentice-Hall.
- Ballard, D., 1997, *An introduction to Natural Computation*, The MIT Press.
- Silverman, B. W., 1986, *Density estimation for statistics and data analysis*, Chapman & Hall.
- Smith, M., 1994, *Neural networks for statistical modeling*, Van Nostrand Reinhold.
- Doak, J., 1992, *An evaluation of feature selection methods and their application to Computer Security*, University of California at Davis, Tech Report CSE-92-18
- Kohavi, R. and Sommerfield, D., 1995, "Feature Subset Selection Using the Wrapper Model: Overfitting and Dynamic Search Space Topology" in *Proc. of the 1st Intl' Conf. on Knowledge Discovery and Data Mining*, pp. 192-197.