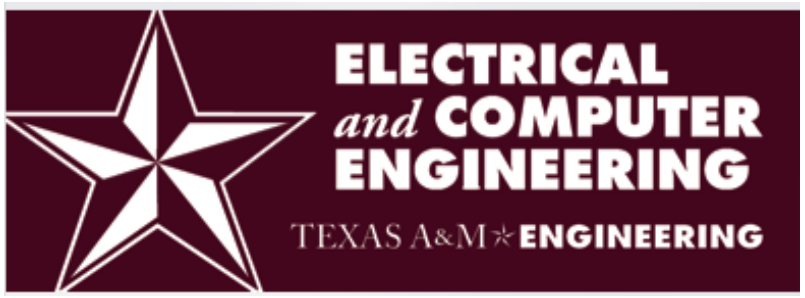
Materials Informatics

Lecture 5: Classification Algorithms

Ulisses Braga Neto

Department of Electrical and Computer Engineering
Texas A&M University

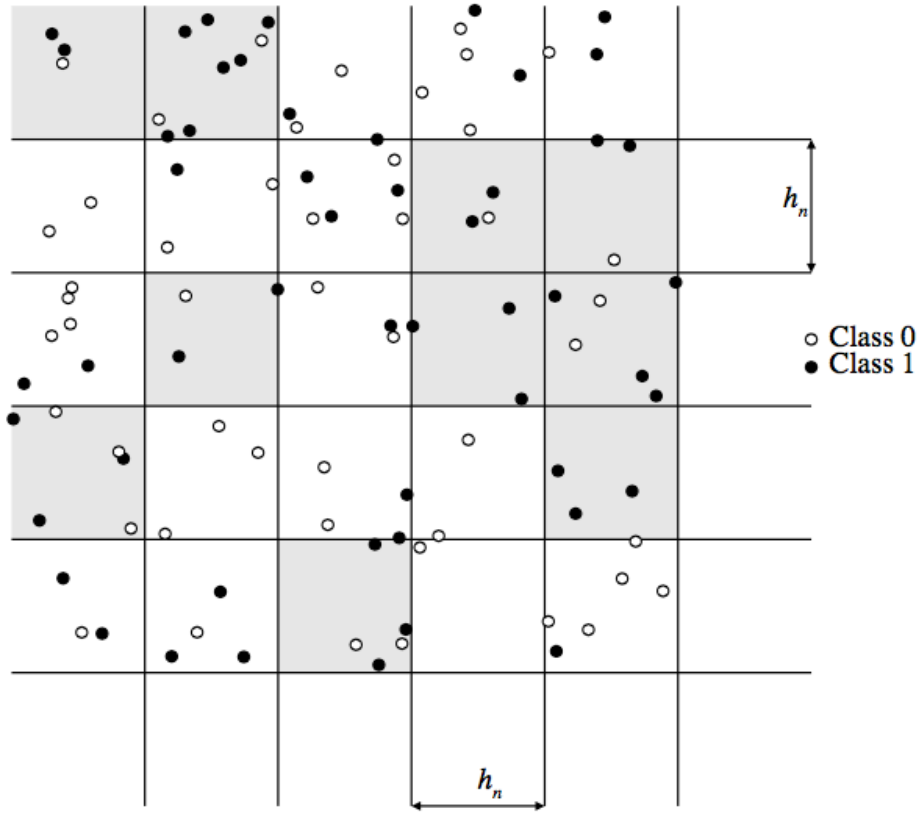


Classification Algorithms

- Classification algorithms or rules (input: data, output: classifier) can be divided into 3 categories.
- Parametric Rules:
  - LDA and its variants, e.g. Q(uadratic)DA, D(iagonal)LDA
  - Logistic Regression
- Nonparametric Rules:
  - Histogram Classifiers
  - Nearest Neighbor Classifiers
  - Kernel Classifiers
- Adjustable Discriminant ("Machine Learning") Rules:
  - Support Vector Machines
  - Neural Networks
  - Decision Trees

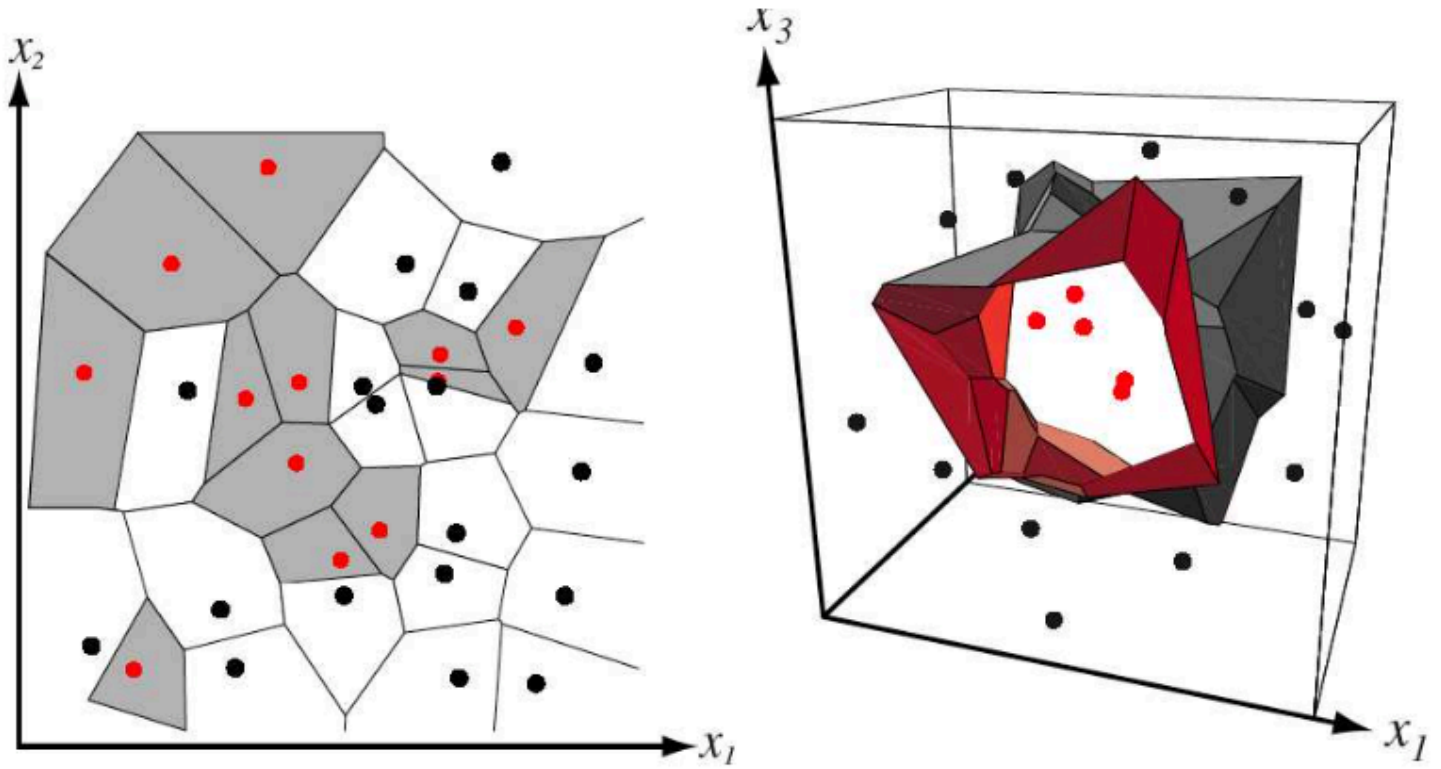
Histogram Rules

- Assign majority label in each cell.
- For continuous or discrete feature spaces.
- Universally consistent if $h_n \to 0$ but $n h_n^d \to \infty$ as $n \to \infty$.



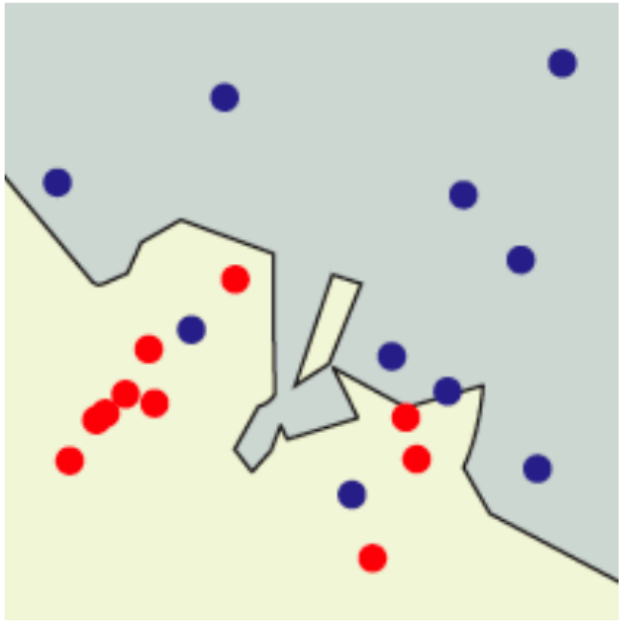
Nearest-Neighbor Rules

- Assign the majority label among a given number *K* of neighbors in the training data.
- Universally consistent if $K \to \infty$ while $K/n \to 0$ as $n \to \infty$.
- Example: K = 1 (nearest neighbor rule).



3-Nearest-Neighbor Rule (3NN)

- The nearest neighbor rule is known to suffer from extreme overfitting (the apparent error is always zero!).
- The case $K = 3$ shows in practice, especially with small samples, to be a good compromise between too little smoothing ($K = 1$) and too much smoothing ($K \geq 5$).



Cover-Hart Theorem

- One of the most famous theorems of Pattern Recognition was proved by Cover and Hart in 1967. It implies that the asymptotic performance of the nearest neighbor (1NN) rule is at worst twice the Bayes error.
- (DGL Theorem 5.1) For any distribution F_{XY}, one has

$$\epsilon_{NN} = E[2\eta(X)(1-\eta(X))]$$

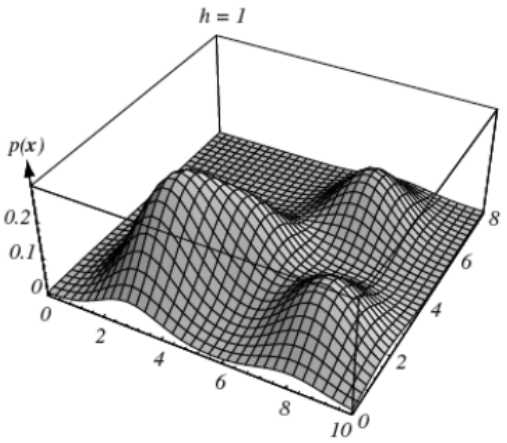
- As can be shown easily (DGL p. 22), this implies

$$\epsilon_{NN} \leq 2\epsilon^*(1-\epsilon^*) \leq 2\epsilon^*$$

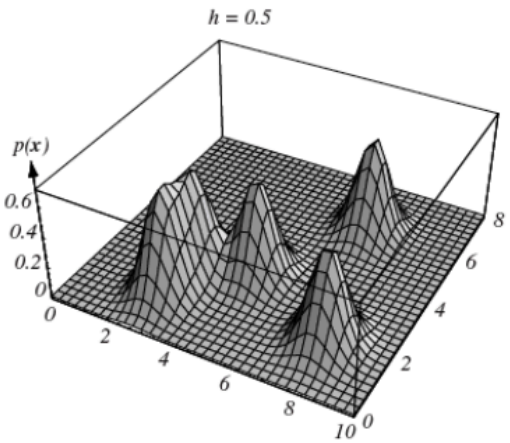
where ϵ_{NN} denotes the classification error as $n \to \infty$ with **fixed** *K*.

Kernel Rules

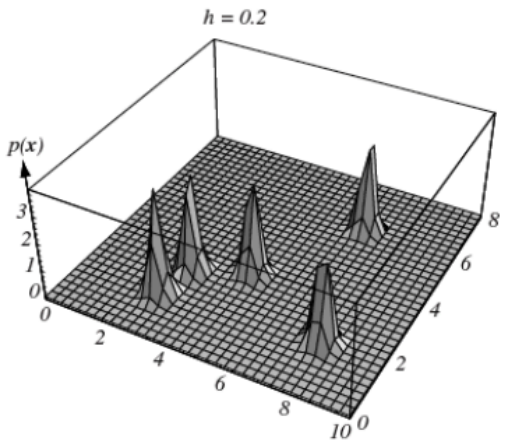
- Each class discriminant is a smoothed version of the empirical distribution (akin to bolstering). The label is assigned to the dicriminant that is largest at the new observation.
- Universally consistent under certain conditions on the kernel.



gaussian kernel

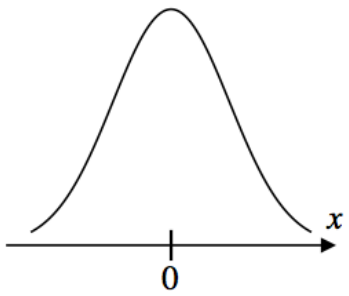


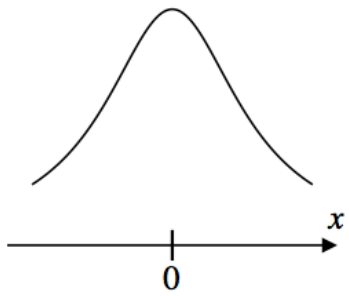
Cauchy kernel

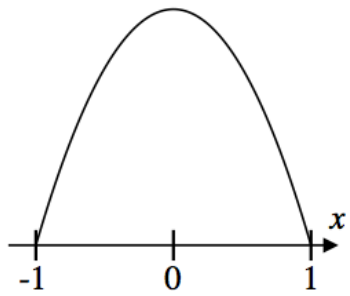


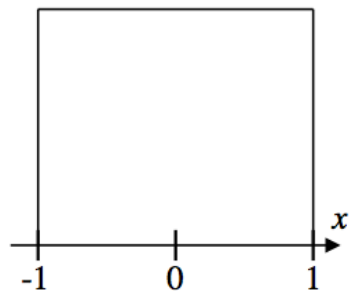
Epanechnikov kernel

uniform kernel







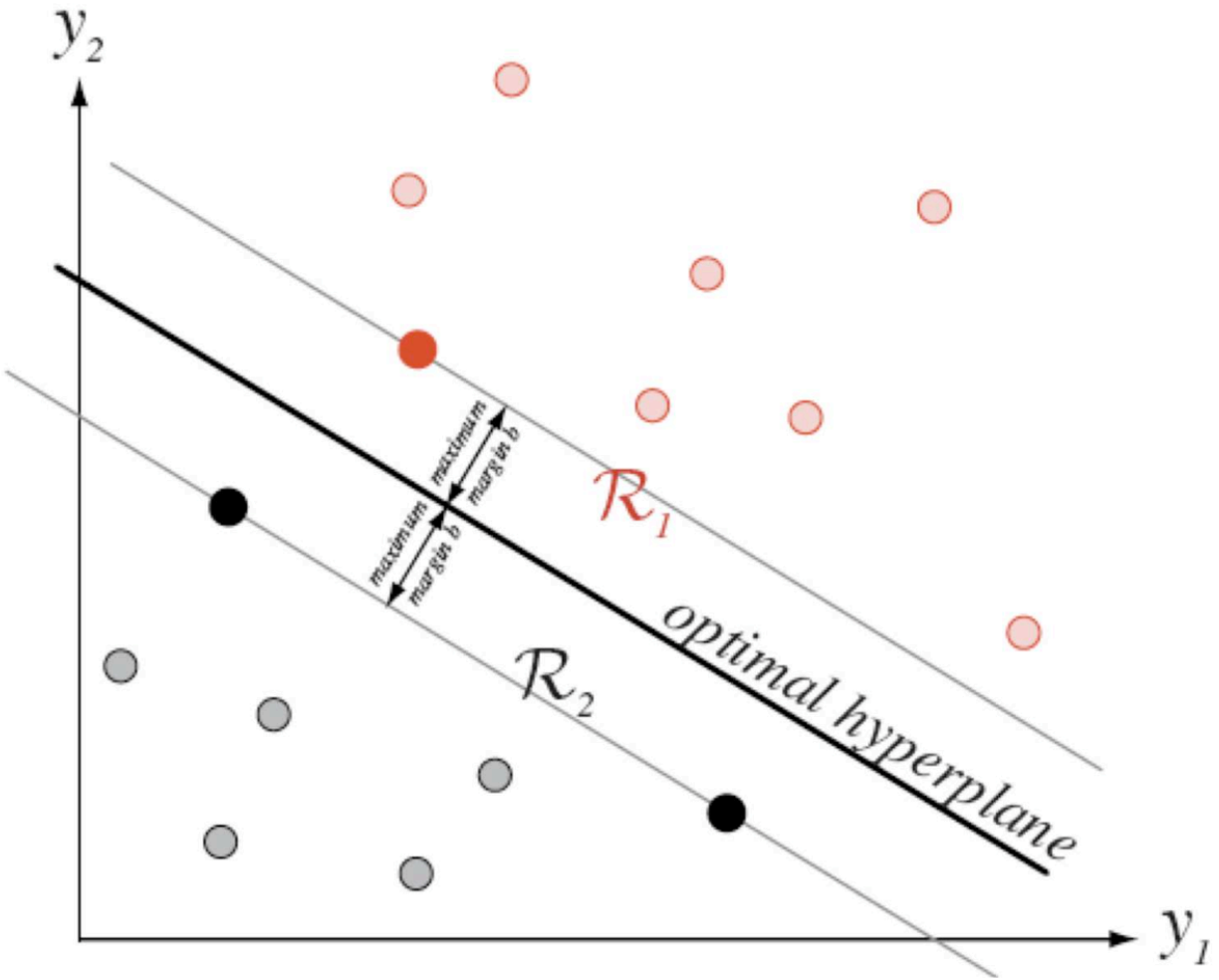


Support Vector Machines

- Adjust linear discriminant with margins such that the margin is *maximal* — this is called the *maximum margin algorithm*.
- The points closest to the hyperplane are called the *support vectors* and determine a lot of the properties of the classifier.
- A solution for the case where the data is not linearly separable can be readily found by using slack variables in the optimization problem.
- Alternatively, project the data to a high-dimensional space, where it is linearly separable. The solution is such that the classifier can be written in terms of *kernels* in the original space.

Linear SVM

- Maximum-Margin Hyperplane (MMH) algorithm.



MMH Algorithm

The idea is to maximize the margin $1/||a||$.

For this, it suffices to minimize $\frac{1}{2}\,||a||^2 = \frac{1}{2}\,a^T a$ subject to the constraints:

$$y_i(a^T x_i + a_0) \geq 1, \quad i = 1, \ldots, n$$

The solution vector a^* determines the Maximal Margin Hyperplane (MMH).

Note: the corresponding optimal value a_0^* will be determined later from a^* and the constraints.

Nonseparable Case

If the data is not linearly separable, it is still possible to formulate the problem and find a solution by introducing *slack variables* ξ_i, $i = 1, \ldots, n$, for each of the constraints, resulting in a new set of $2n$ constraints:

$$y_i(a^T x_i + a_0) \geq 1 - \xi_i \text{ and } \xi_i \geq 0, \quad i = 1, \ldots, n$$

Therefore, if $\xi_i > 0$, the corresponding training point is an "outlier," i.e., it can lie closer to the hyperplane than the margin, or even be misclassified. We introduce a penalty term $C \sum_{i=1}^{n} \xi_i$ in the functional, which then becomes:

$$\frac{1}{2} a^T a + C \sum_{i=1}^{n} \xi_i$$

Penalty Constant

The constant C modulates how large the penalty for the presence of outliers are. If C is small, the penalty is small and a solution is more likely to incorporate outliers. If C is large, the penalty is large and therefore a solution is unlikely to incorporate many outliers.

This means that small C favors a "softer" margin and therefore less overfitting, whereas large C means that the solution will attempt to adjust to the training data so as to minimize the number of outliers, leading to more overfitting. In summary, the amount of overfitting is directly proportional to the magnitude of C. Too small a C on the other hand may lead to *underfitting*, that is, too much slackness is allowed and the classifier does not fit the data at all.

Penalty Constant

The constant C modulates how large the penalty for the presence of outliers are. If C is small, the penalty is small and a solution is more likely to incorporate outliers. If C is large, the penalty is large and therefore a solution is unlikely to incorporate many outliers.

This means that small C favors a "softer" margin and therefore less overfitting, whereas large C means that the solution will attempt to adjust to the training data so as to minimize the number of outliers, leading to more overfitting. In summary, the amount of overfitting is directly proportional to the magnitude of C. Too small a C on the other hand may lead to *underfitting*, that is, too much slackness is allowed and the classifier does not fit the data at all.

Nonlinear SVM

Idea: Find transformation $\phi(x)$ that takes the original feature space into a higher-dimensional space, and apply the MMH algorithm there. The corresponding classifier back in the original space will generally be non-linear.

"Kernel Trick": One can avoid directly dealing with $\phi(x)$ by introducing a kernel $K(x, y) = \phi^T(x)\phi(y)$.

Some examples of kernels used in applications:

- Polynomial: $k(x, y) = (1 + x^T y)^p$
- Gaussian: $k(x, y) = \exp(-|x - y|^2/\sigma^2)$
- Sigmoid: $k(x, y) = \tanh(kx^T y - \delta)$

Example: XOR Problem

This is the simplest non-linearly separable problem (with minimal number of points).

The data set consists of:

- Class 0: $\{(-1, 1), (1, -1)\}$
- Class 1: $\{(-1, -1), (1, 1)\}$

We consider the polynomial kernel of order 2:

$$\begin{aligned} k(x, y) &= (1 + x^T y)^2 = (1 + x_1 y_1 + x_2 y_2)^2 \\ &= 1 + 2x_1 y_1 + 2x_2 y_2 + 2x_1 x_2 y_1 y_2 + x_1^2 y_1^2 + x_2^2 y_2^2 \\ &= \phi^T(x)\phi(y) \end{aligned}$$

where $\phi(x) = (1, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1 x_2, x_1^2, x_2^2)$.

Example: XOR Problem

The optimal solution is given by:

$$a^* = \sum_{i=1}^{4} \lambda_i^* y_i \phi(x_i) = (0, 0, 0, \frac{1}{\sqrt{2}}, 0, 0)^T$$

and

$$a_0^* = -\frac{1}{4} \sum_{i=1}^{4} \sum_{j=1}^{4} \lambda_i^* y_i k(x_i, x_j) + \frac{1}{4} \sum_{i=1}^{4} y_i = 0$$

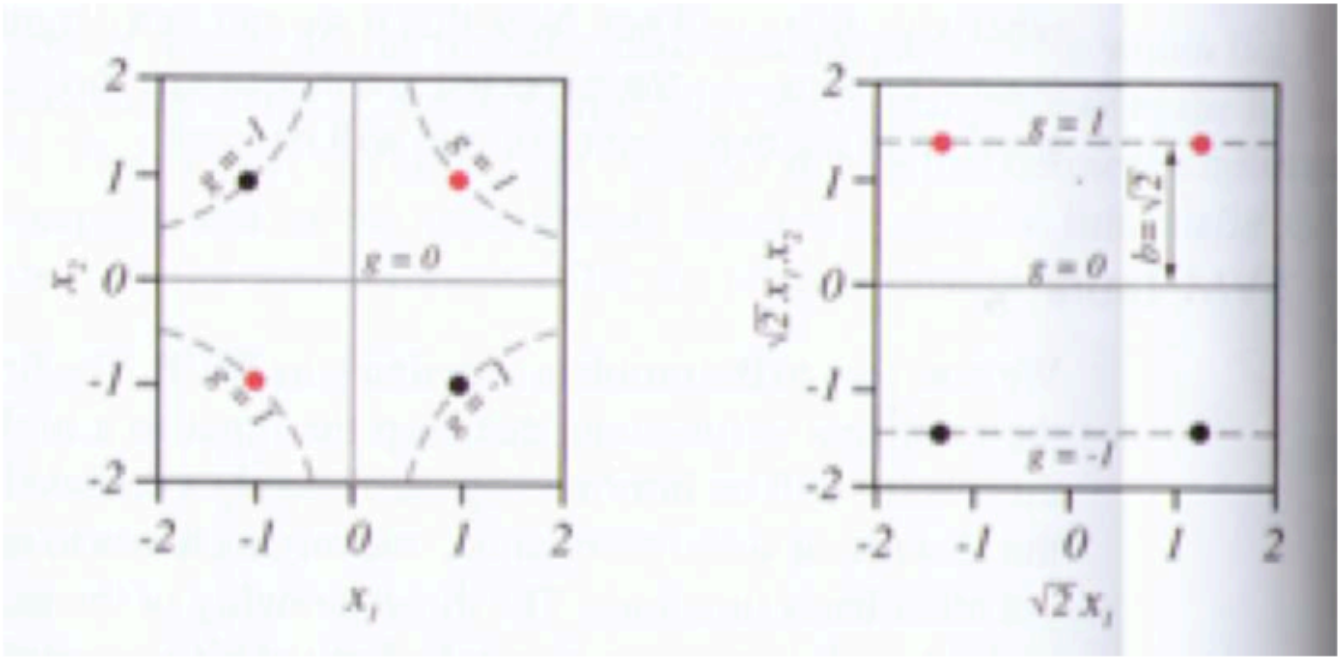
so that the MMH in transformed space is determined by:

$$(a^*)^T \phi(x) + a_0^* = x_1 x_2 = 0$$

with optimal margin $\frac{1}{||a||} = \sqrt{2}$.

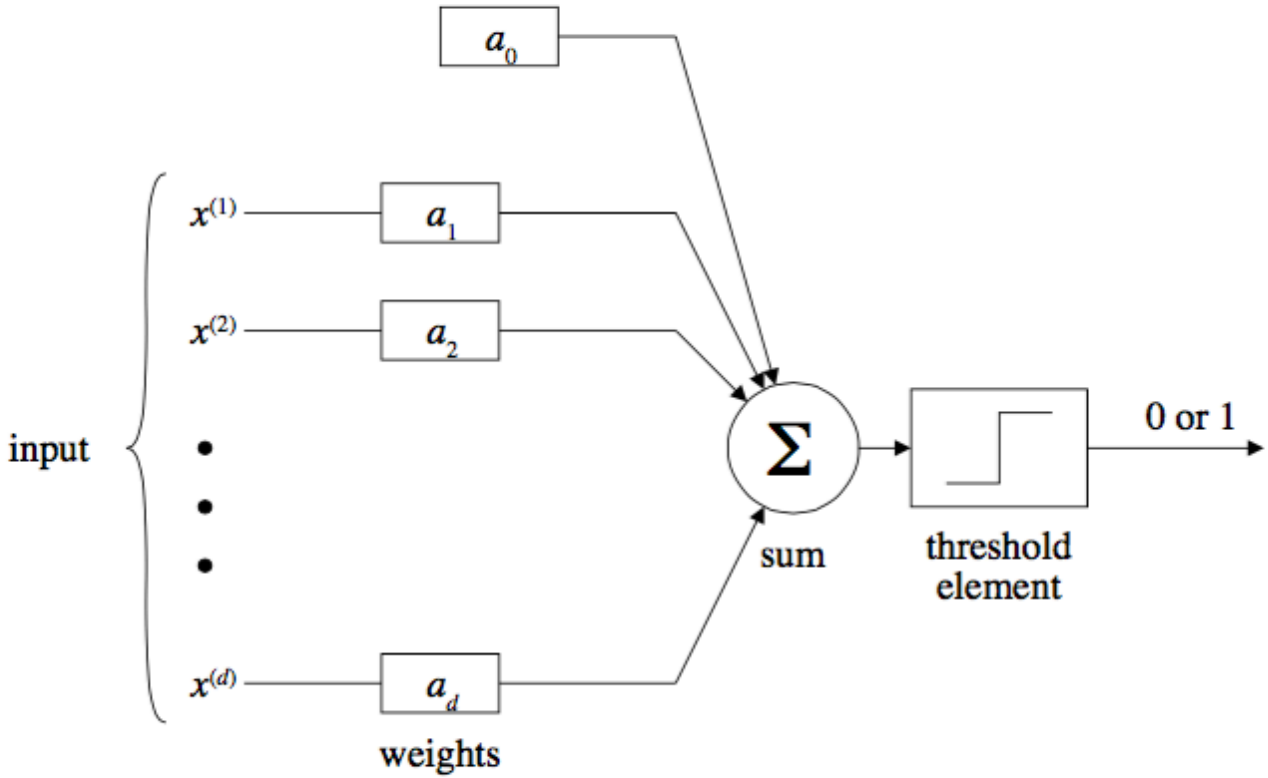
Example: XOR Problem

The classifier in the original and transformed feature spaces is depicted below.



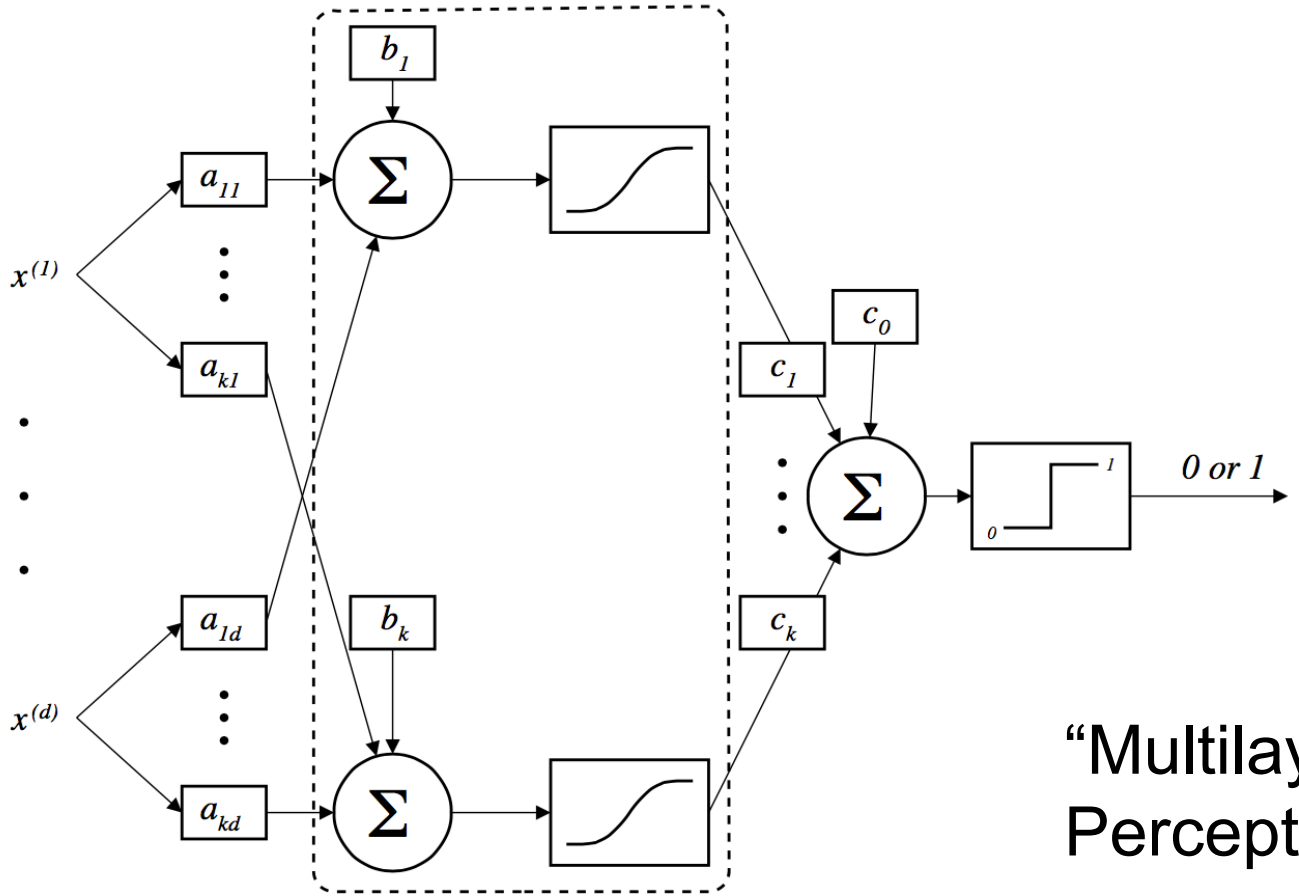
Perceptron

Rosenblatt's Perceptron (1959)



Neural Networks

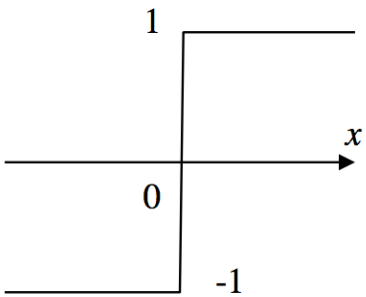
Diagram for a neural network with one hidden layer.



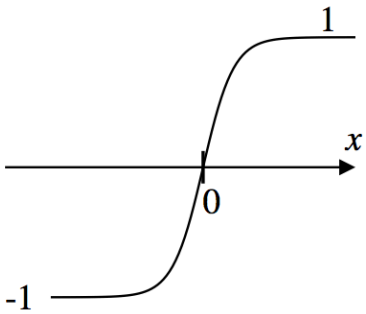
"Multilayer Perceptron"

Sigmoids

Threshold

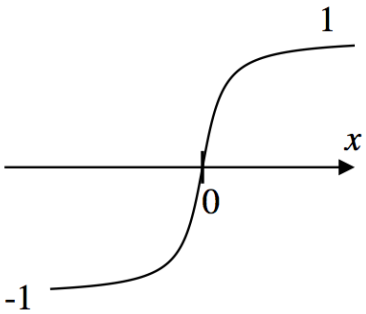


Standard (logistic)



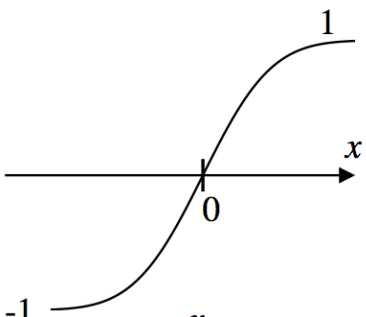
$$\sigma(x) = \frac{1-e^{-x}}{1+e^{-x}}$$

Arctan



$$\sigma(x) = \frac{2}{\pi}\arctan(x)$$

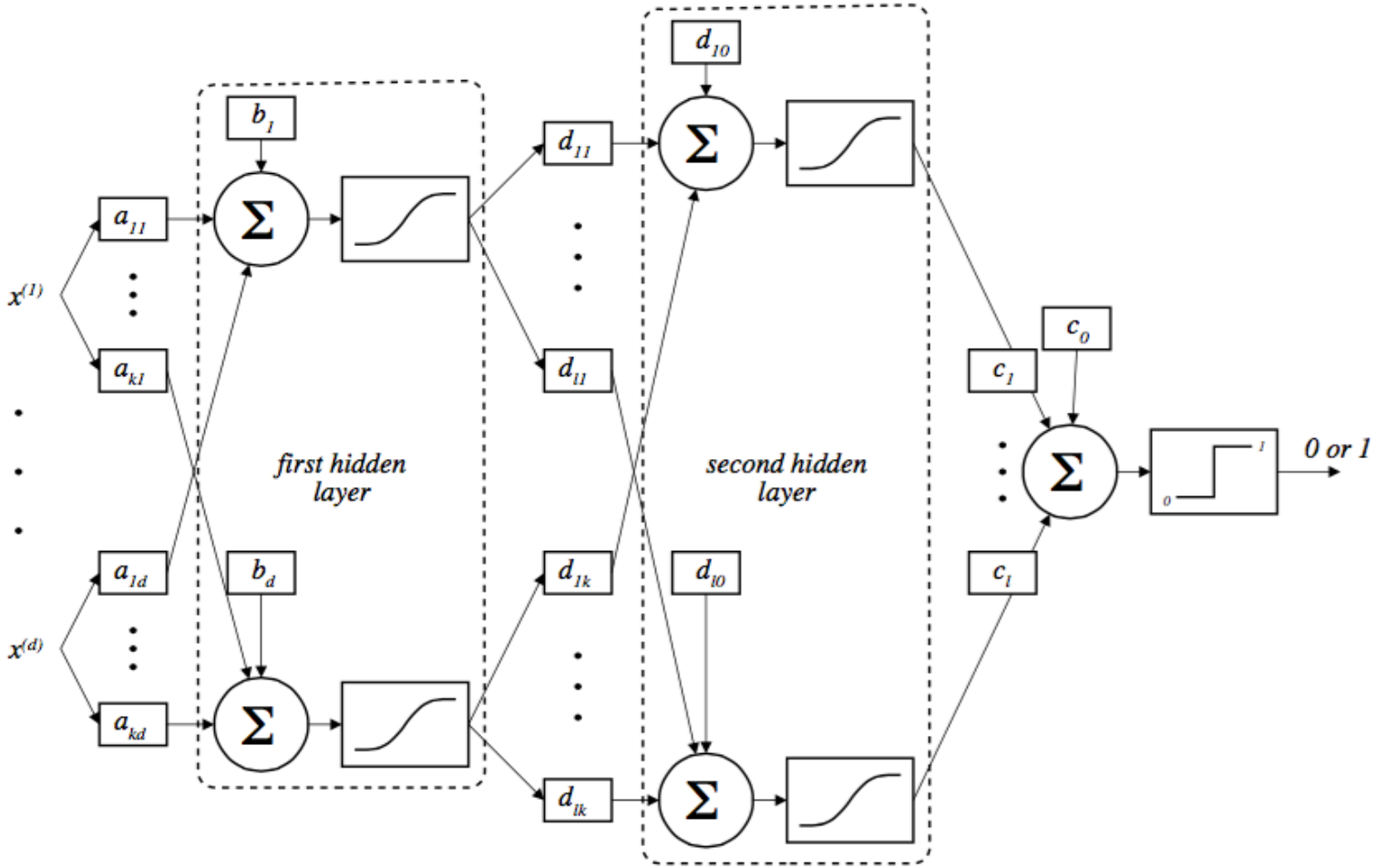
Gaussian



$$\sigma(x) = \sqrt{\frac{2}{\pi}} \int_{-\infty}^{x} e^{-v^2/2}\, dv - 1$$

Multilayer Neural Networks

This idea can be extended to any number of hidden layers:



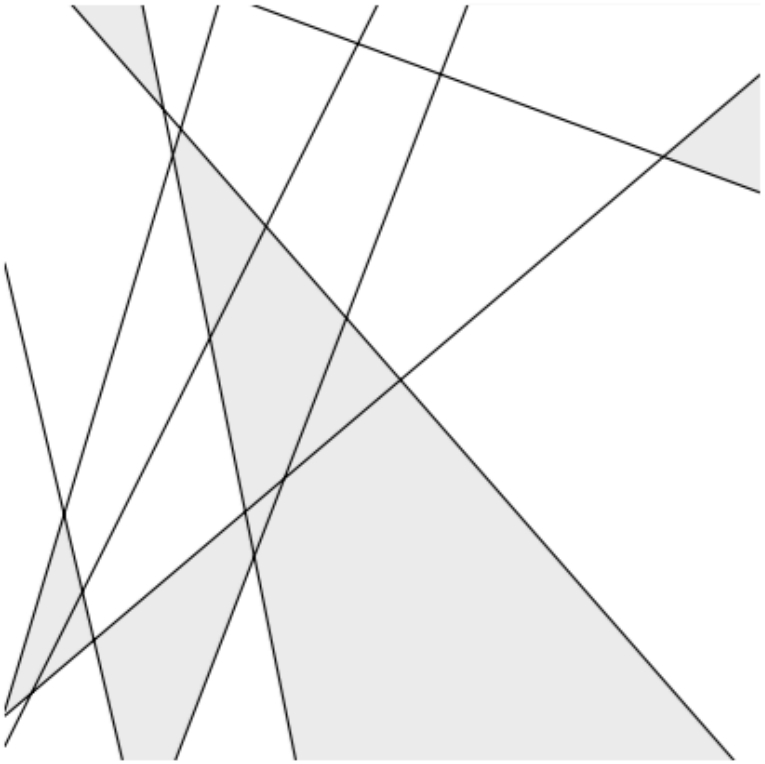
Some Observations

- The neural network approach is similar to the non-linear SVM approach. The hidden layers nonlinearly map the original feature space into a different space, and the output layer acts on the transformed features by means of a linear decision (hyperplane). The decision in the original feature space is nonlinear.

- If the *first* hidden layer is composed of threshold sigmoids, then no matter what follows, the decision regions will be piecewise linear, given by the intersection of k hyperplanes (this is called an *arrangement classifier*.

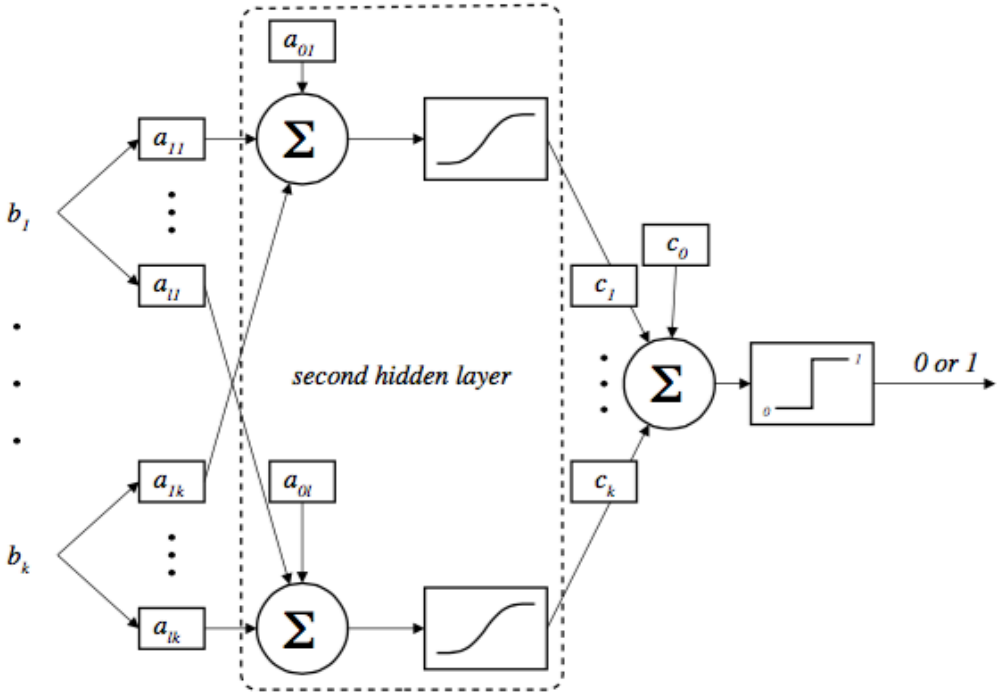
Arrangement Classifiers

The decision regions for an arrangement classifier are *convex polytopes*, which are determined by intersections of half-spaces. The classifier is therefore *piecewise linear*.



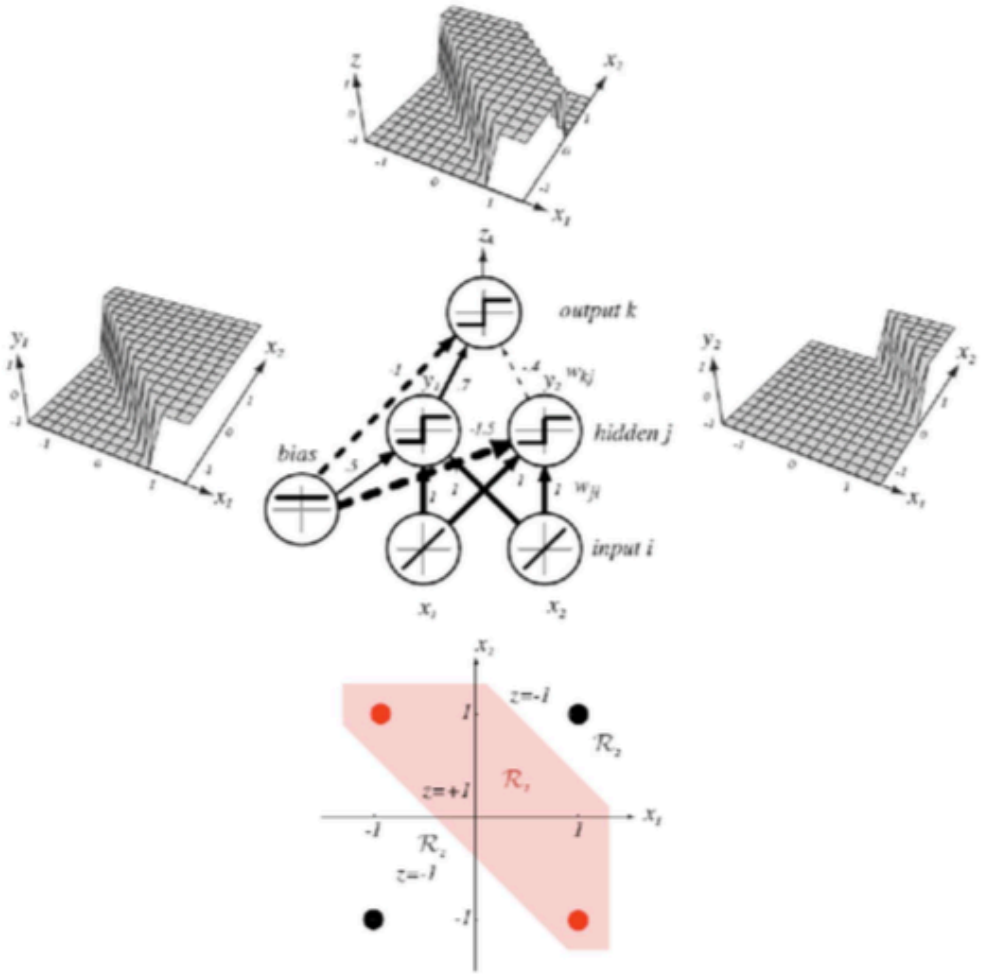
Arrangement Classifiers - II

The decision regions in the arrangement classifier are determined by the first hidden layer of k threshold sigmoids. Each of the regions corresponds to a binary vector $b = [b_1, \ldots, b_k]$. A second (or all subsequent) hidden layers can only alter the class attributed to each of these regions.



XOR Problem

The neural network solution to the XOR problem is an arrangement classifier.



How to Pick Parameters?

- Minimize the empirical error:

$$J(w) = \frac{1}{n} \sum_{i=1}^{n} |Y_i - \psi_n(X_i)|$$

- Minimize the *mean absolute error* (where $t_i = \pm 1$):

$$J(w) = \sum_{i=1}^{n} |t_i - \zeta(X_i)|$$

- Minimize the *mean square error*:

$$J(w) = \frac{1}{2} \sum_{i=1}^{n} [t_i - \zeta(X_i)]^2$$

(this leads to the *Backpropagation Algorithm*.)

Network Training

Our problem is to find the vector of weights w that minimizes the error criterion.

For this we need to choose a non-linear optimization technique. A few examples that are common in least-square problems (such as the current problem):

- Gauss-Newton Algorithm
- Levenberg-Marquardt Algorithm
- Gradient Descent ("Backpropagation Algorithm")

It should be noted that all these procedures are iterative.

Backpropagation Algorithm

The basic iteration step in gradient descent is given by:

$$\Delta w = -\ell \, \nabla J$$

that is, the vector of weights w is updated in the direction that decreases the error criterion, according to the step length ℓ.

By writing the above in component form, we get the updates for each weight separately:

$$\Delta w_i = -\ell \, \frac{\partial J}{\partial w_i}$$

The backpropagation algorithm consists of applying the chain rule to compute these partial derivatives.

Backpropagation Algorithm

- Initalize the weight vector with random numbers (not all zeros!).
- Present one training pattern at a time and update each weight according to the equation given previously.
- Stop when there is no significant change in the error criterion function.

Notes:

1. A single presentation of the entire training set is called an *epoch*. The amount of training is measured by the number of epochs.

2. Each update is guaranteed to lower the individual error $J_i(w)$ for the pattern, but it could increase the total error $J_T(w) = \sum_{i=1}^{n} J_i(w)$. But, in the long run, $J_T(w)$ decreases.

Overfitting in NN Training

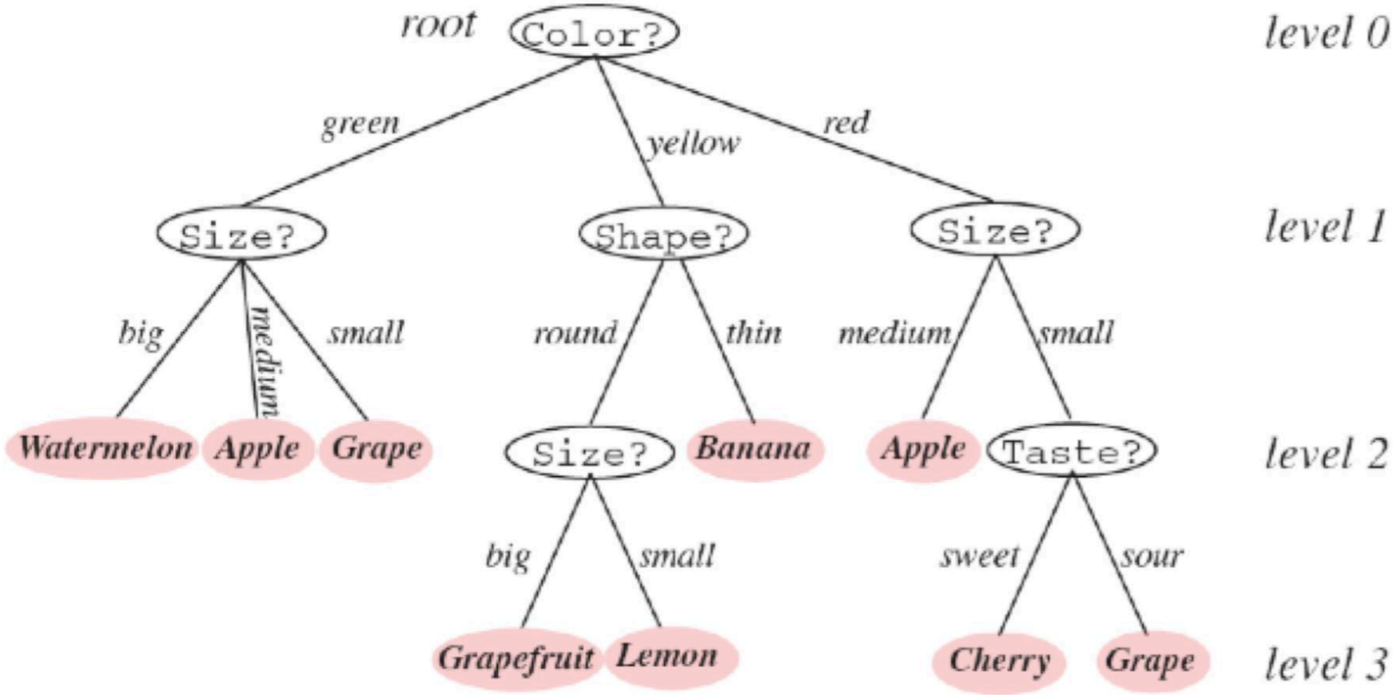
Even though $J_T(w)$ decreases in the long run, this is still the error on the training data, so there is danger of overfitting.

Decision Trees

- Adjustable partition-based discriminants that cut up the space recursively (“20-questions” approach).
- Non-metric data: tree classifiers can handle data where some or all the features are *nominal*, i.e., not numeric.
- High degree of *interpretability*: Inferred classifier can be specified in terms of logical rules that can lead to scientific hypotheses about the mechanism that produced the data.
- Very popular in *Data Mining*.

Tree Classifier

A tree classifier consists of nodes where data splitting occurs. There is a *root* node, and terminal *leaf* nodes where label assignment occurs.



Logical Rules

- Each leaf node in a tree classifier correponds to a rule, obtained by *traversing* the tree from the root node to the leaf. For example:

  Banana := `(color = yellow) AND (shape = thin)`

  `= (yellow AND thin)`

- We can combine rules for identical leafs by using the operator OR:

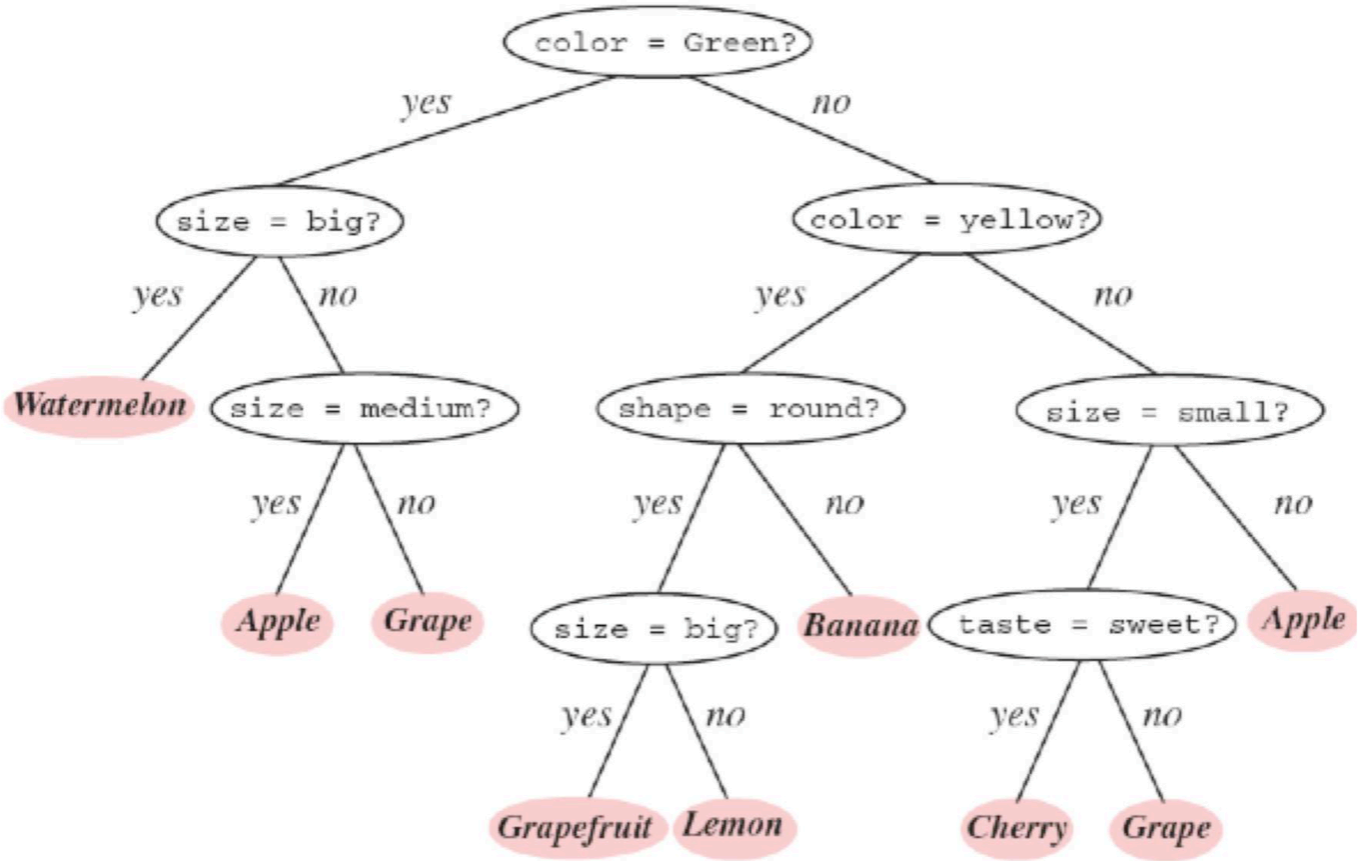
  Apple := `(green AND medium) OR (red AND medium)`

- Rules can also be simplified logically to aid interpretability

  Apple := `(medium AND NOT yellow)`

Binary Tree Classifier

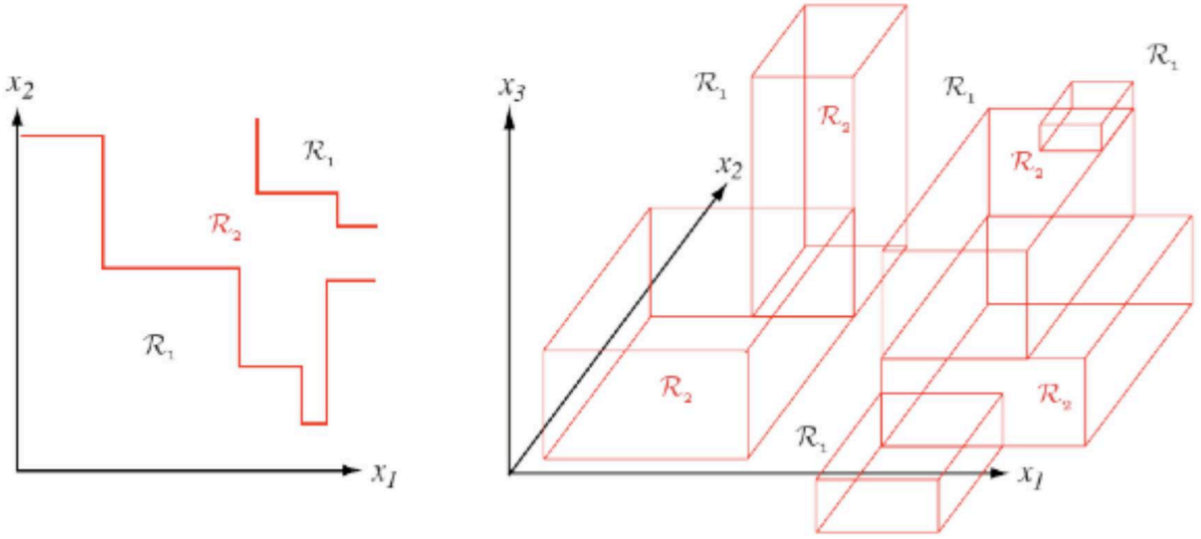
Any decision tree can be written as a *binary* tree, i.e, trees for which the splitting criteria are all binary.



CART

CART (Classification and Regression Trees) is a very popular tree rule for numeric data.

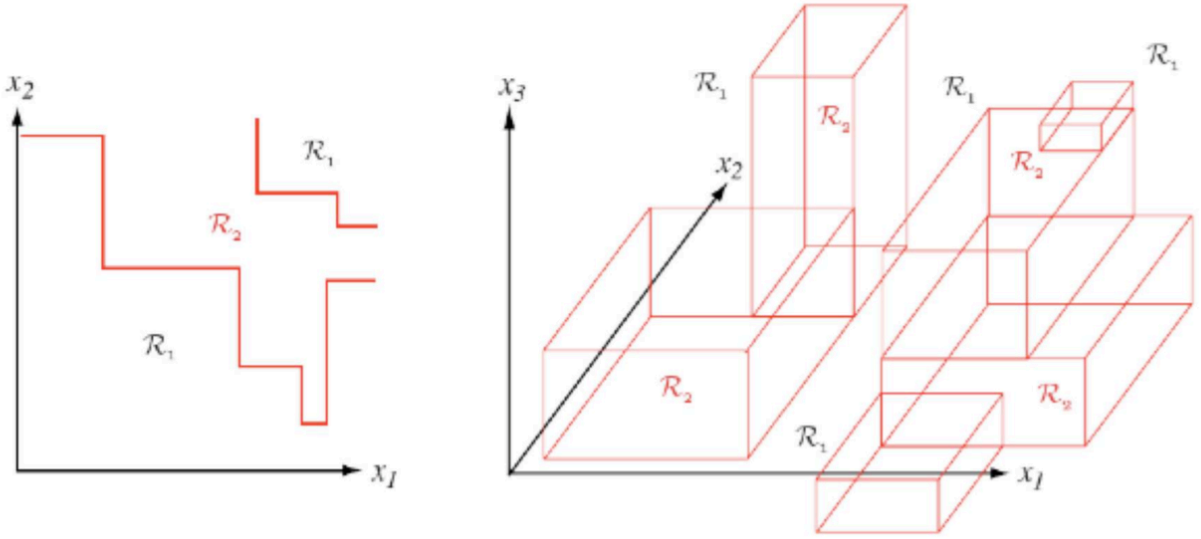
CART produce tree classifiers where the decision at each node is of the form $x^j \leq \alpha?$ where x^j is one of the coordinates of the pattern x. Such classifiers produce linear boundaries that are parallel to the axes.



CART

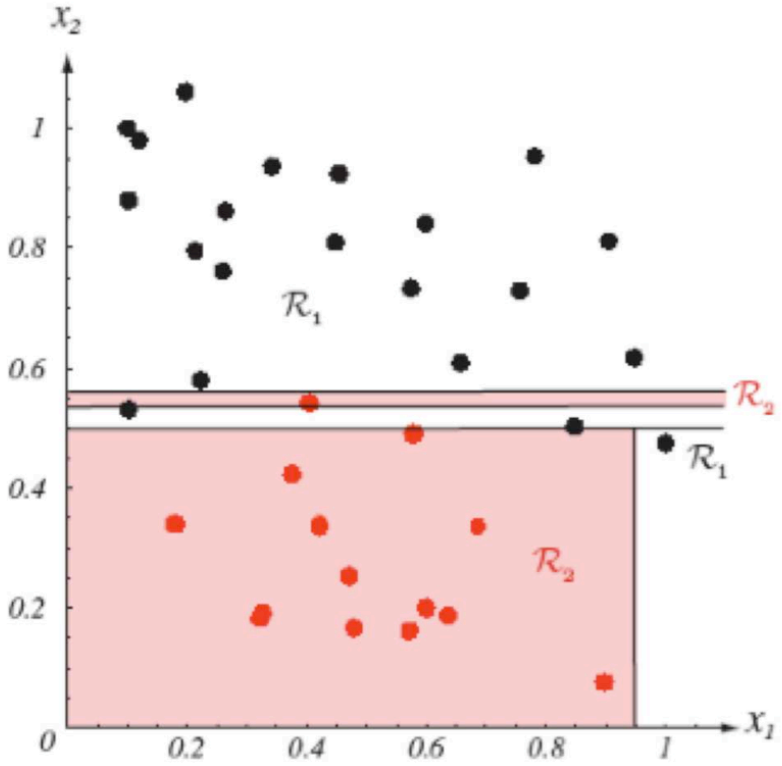
CART (Classification and Regression Trees) is a very popular tree rule for numeric data.

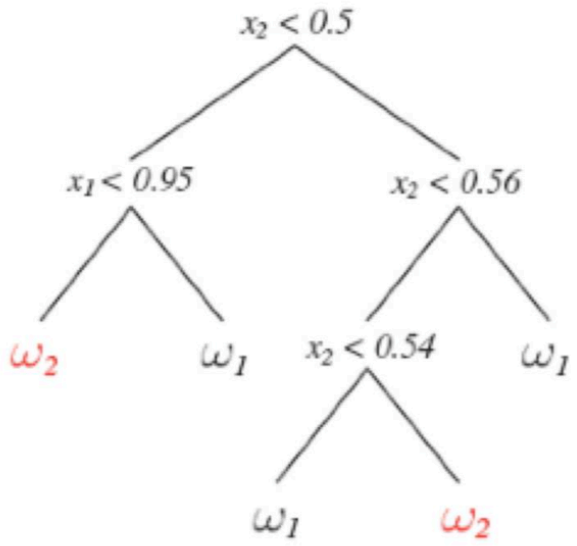
CART produce tree classifiers where the decision at each node is of the form $x^j \leq \alpha?$ where x^j is one of the coordinates of the pattern x. Such classifiers produce linear boundaries that are parallel to the axes.



Fully-Grown CART

In this case, the tree is grown until there is a single point in each leaf. Example:





Regularization

A fully-grown CART tree classifier almost surely overfits the training data, and is not a good classifier in terms of true classification error.

Two basic regularization techniques are available:

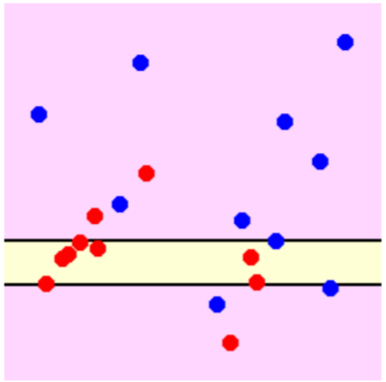
- Stopped Splitting: Call a node a leaf and assign the majority label if
  (1) there are fewer than k samples in it.
  (2) the best impurity drop is below a threshold θ.
- Pruning: Fully grow tree then consider pairs of neighboring leafs that can be merged without increasing impurity too much.

In addition, one can design several fully-grown CARTs via perturbation of the training data and combine the decisions by majority voting — this is the *random forest* approach.

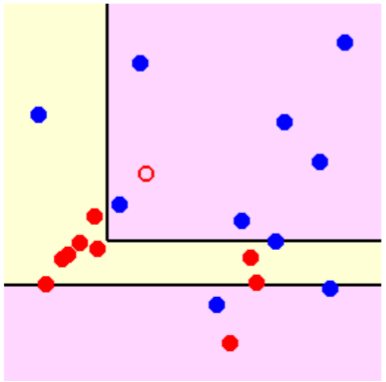
Small-Sample Instability of CART

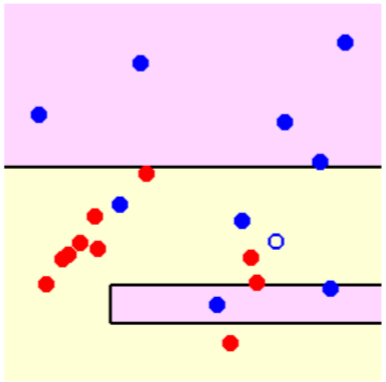
Example of CART with perturbed data (with stopping rule!)

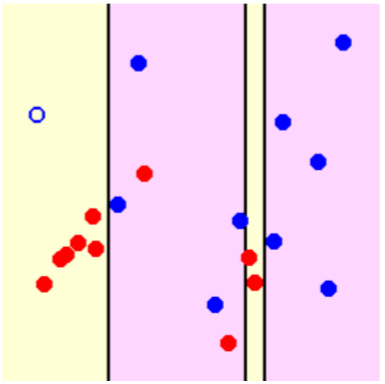
Original CART classifier



CART classifiers with one point deleted from training data





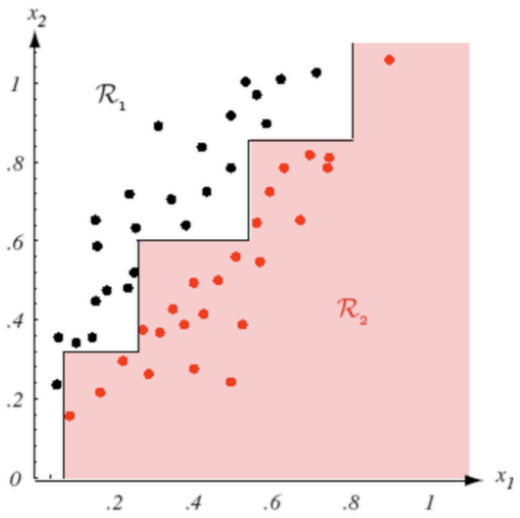


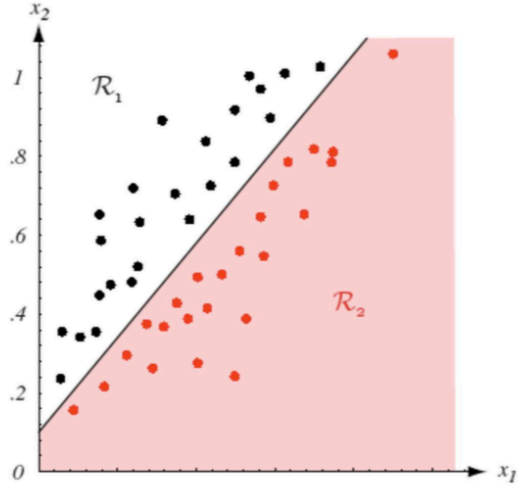
Lack of Invariance

CART rules are invariant with respect to scaling of the axes and translation, meaning that if T denotes such a transformation, we have:

$$\Psi_n(T(x); T(S_n) = \{(T(x_i), y_i)\}) \;=\; \Psi_n(x;\; S_n = \{(x_i, y_i)\})$$

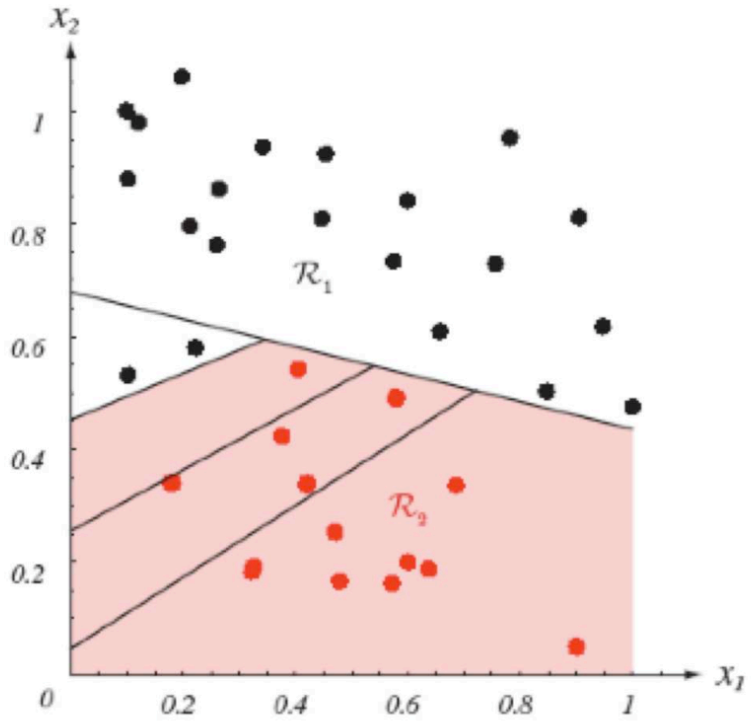
CART rules are *NOT* invariant to more complicated transformations, such as rotation.

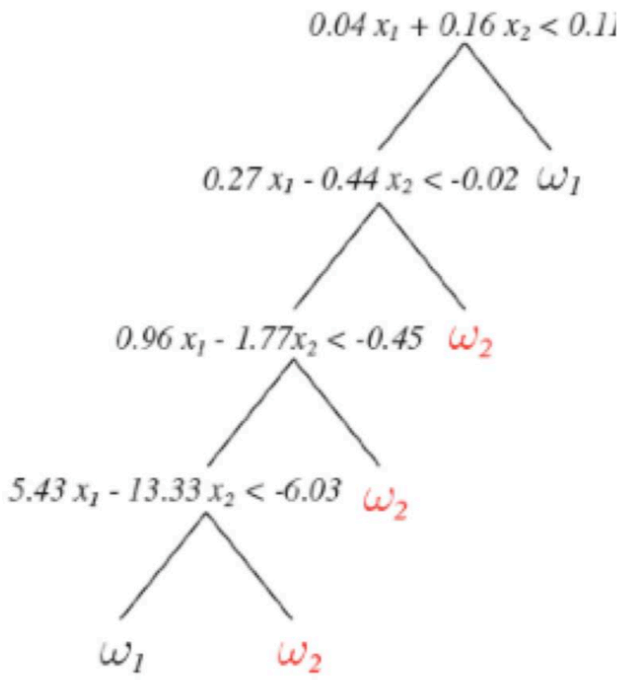




BSP Tree

Binary Space Partition (BSP) trees are similar to CART, but they employ hyperplane splits along arbitrary directions.





Model Selection

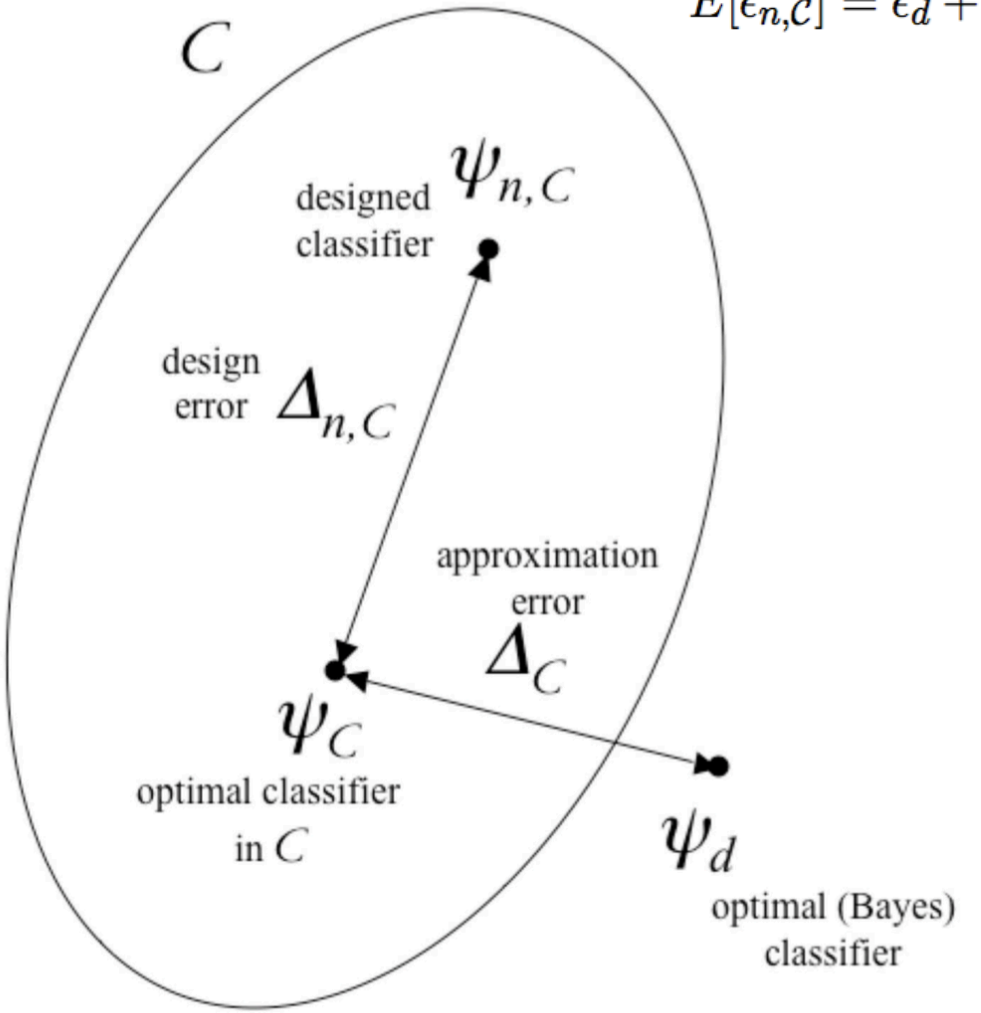
- Obvious questions about classification are: Which classification rule should one choose? How many features should one use?
- A related question is: given a classification rule, how does one pick its free parameters? For example,
  - In k-NN classification, which k to use?
  - In kernel classification, how to choose the kernel bandwidth?
  - With neural networks, which architecture to use? How many epochs to use in training?
- The answer to such questions is not simple. It depends on sample size, the complexity of the classification rule, and on the distribution. In this lecture, we will examine these issues.

Graphical Representation

set of classifiers
produced by algorithm

Expected Classification Error:

$$E[\epsilon_{n,\mathcal{C}}] = \epsilon_d + \Delta_{\mathcal{C}} + E[\Delta_{n,\mathcal{C}}]$$



Complexity Dilemma

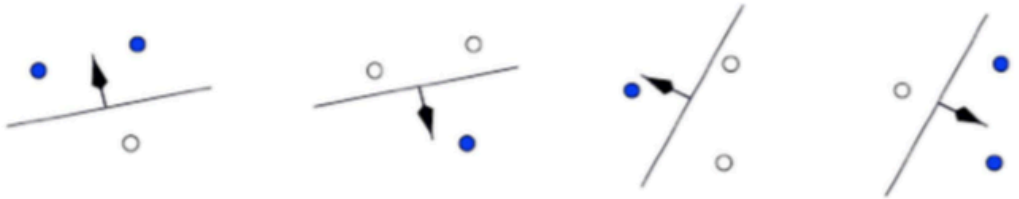
- In order to pick a $\mathcal{C}$, we want the the expected classification error $E[\epsilon_{n,\mathcal{C}}]$ to be as small as possible.
- Therefore, we would like both the approximation error $\Delta_{\mathcal{C}}$ and the expected design error $E[\Delta_{n,\mathcal{C}}]$ to be small.
- Unfortunately, that is not in general possible, due to the *complexity dilemma*.
- The complexity of a classification rule can be defined as its *expressive power*, or how finely it can cut up the feature space. That is, the size of $\mathcal{C}$.

Vapnik-Chervonenkis Dimension

- The Vapnik–Chervonenkis (VC) dimension is a measure of the size, i.e., the complexity, of a class of classifiers $\mathcal{C}$.
- It agrees very naturally with our intuition of complexity as the ability of a classifier to cut up the space finely.

Example: For a linear classification rule with d features, VC = d + 1.

d = 2,
VC = 3



Vapnik-Chervonenkis Theorem

- If $V_{\mathcal{C}}$ is finite,

$$P\left(\sup_{\psi \in \mathcal{C}} |\hat{\epsilon}[\psi] - \epsilon[\psi]| > \tau\right) \leq 8(n+1)^{V_{\mathcal{C}}} e^{-n\tau^2/32}, \text{ for all } \tau > 0$$

- Therefore, if $V_{\mathcal{C}}$ is finite, the term $e^{-n\tau^2/32}$ dominates, and the bound decreases *exponentially* fast as $n \to \infty$.

Structural Risk Minimization

- The *Structural Risk Minimization* (SRM) principle is a model selection method that balances a small apparent error against the complexity of the classification rule.
- Let us begin by rewriting the bound in the VC Theorem, doing away with the supremum and the absolute value:

$$P\left(\epsilon[\psi] - \hat{\epsilon}[\psi] > \tau\right) \leq 8(n+1)^{V_{\mathcal{C}}} e^{-n\tau^2/32}, \quad \text{for all } \tau > 0$$

  which holds for any $\psi \in \mathcal{C}$.
- Assuming $V_{\mathcal{C}} < \infty$, let the right-hand side be equal to ξ, where $0 \leq \xi \leq 1$. Solving for τ gives:

$$\tau(\xi) = \sqrt{\frac{32}{n}\left[V_{\mathcal{C}} \log(n+1) - \log\left(\frac{\xi}{8}\right)\right]}$$

Structural Risk Minimization

- For a given ξ, we thus have

$$P\left(\epsilon[\psi] - \hat{\epsilon}[\psi] > \tau(\xi)\right) \leq \xi \Rightarrow P\left(\epsilon[\psi] - \hat{\epsilon}[\psi] \leq \tau(\xi)\right) \geq 1 - \xi$$

- This allows us to say that the inequality

$$\epsilon[\psi] \leq \hat{\epsilon}[\psi] + \tau(\xi)$$

holds with probability at least $1 - \xi$.

- But this holds for any $\psi \in \mathcal{C}$ (that was the whole point of the VC Theorem), therefore we can in particular let $\psi = \psi_{n,\mathcal{C}}$, a designed classifier from S_n.

Structural Risk Minimization

- With $\psi = \psi_{n,\mathcal{C}}$, we have $\epsilon[\psi] = \epsilon_{n,\mathcal{C}}$, the designed classifier error, and $\hat{\epsilon}[\psi] = \hat{\epsilon}_{n,\mathcal{C}}$, the apparent error (i.e., the resubstitution error), and we can write that

$$\epsilon_{n,\mathcal{C}} \leq \hat{\epsilon}_{n,\mathcal{C}} + \sqrt{\frac{32}{n}\left[V_{\mathcal{C}} \log(n+1) - \log\left(\frac{\xi}{8}\right)\right]}$$

  holds with probability at least $1 - \xi$.

- The second term on the right-hand side is the so-called *VC confidence*. The smaller this term is, the better.
- The actual form of the VC confidence may vary according to the derivation, but in any case it
  - depends only on $V_{\mathcal{C}}$ and n, for a given ξ;
  - is small if $n \gg V_{\mathcal{C}}$, and large otherwise.

Structural Risk Minimization

- Now let $\mathcal{C}^k$ be a sequence of classes associated with classification rules Ψ_n^k, for $k = 1, 2, \ldots$
- We want to be able to pick a classification rule such that the classification error $\epsilon_{n,\mathcal{C}}$ is minimal.
- From the previous slide, this can be done by picking a ξ sufficiently close to 1, computing

$$\hat{\epsilon}_{n,\mathcal{C}^k} + \sqrt{\frac{32}{n}\left[V_{\mathcal{C}^k}\log(n+1) - \log\left(\frac{\xi}{8}\right)\right]}$$

  for each k, and then picking k such that this is minimal.
- So we will want to minimize $\hat{\epsilon}_{n,\mathcal{C}^k}$, but will penalize large $V_{\mathcal{C}}$ compared to n.

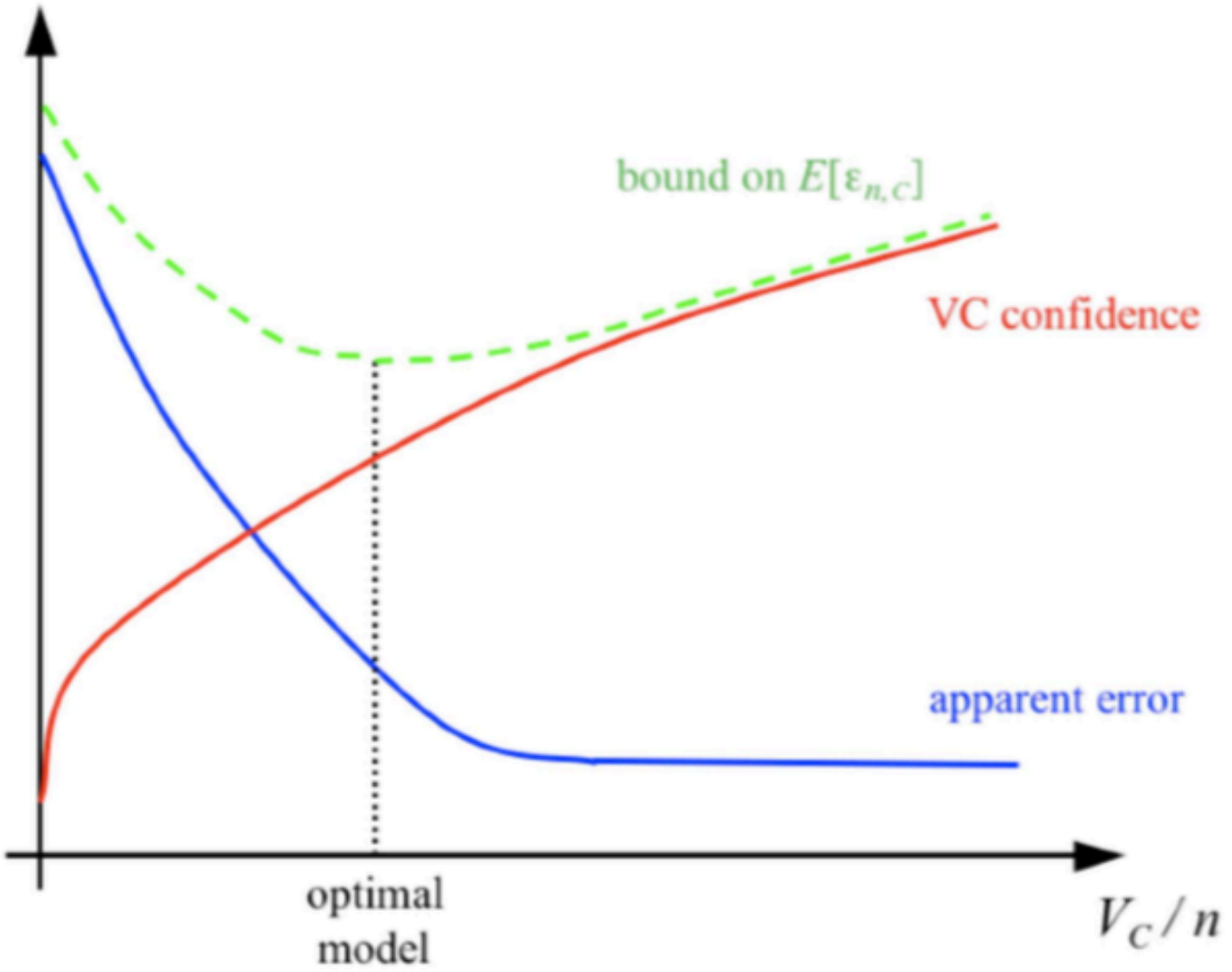
Structural Risk Minimization

- Now let $\mathcal{C}^k$ be a sequence of classes associated with classification rules Ψ_n^k, for $k = 1, 2, \ldots$
- We want to be able to pick a classification rule such that the classification error $\epsilon_{n,\mathcal{C}}$ is minimal.
- From the previous slide, this can be done by picking a ξ sufficiently close to 1, computing

$$\hat{\epsilon}_{n,\mathcal{C}^k} + \sqrt{\frac{32}{n}\left[V_{\mathcal{C}^k}\log(n+1) - \log\left(\frac{\xi}{8}\right)\right]}$$

  for each k, and then picking k such that this is minimal.
- So we will want to minimize $\hat{\epsilon}_{n,\mathcal{C}^k}$, but will penalize large $V_{\mathcal{C}}$ compared to n.

Graphical Representation



Some Words of Caution

- The VC theory provides a powerful method to obtain distribution-free performance guarantees.
- However, its approach is the minimax, or worst-case, approach, because it has to do with universal performance guarantees, that is, for any distribution of (X, Y).
- Therefore, the bounds derived in the theory, though tight with respect to worst-case scenarios, tend to be slack in practice, particularly with small sample sizes.

Other Model Selection Methods

- SRM works for classification rules with finite VC dimension. We also have the caveats associated with VC bounds.
- We may not want to be restricted by this. We mention alternative approaches below.

Data-Independent Heuristics

- For some parameters of some classification rules, heuristic choices are common and effective. For a few examples:
  - KNN: use small odd numbers for K (K=3 works well for small samples).
  - SVM: values for the penalty constant C between 0.1 and 1.
  - SVM: linear or gaussian kernel choice.
  - Neural Networks: use one-hidden layer with threshold sigmoids.
  - Decision Trees: stop splitting at $n < 5$ points in a node.

Minimum-Description Length

- *Minimum-Description-Length* (MDL) methods replace error minimization by minimization of a sum of entropies, one relative to encoding the error and the other relative to encoding the classifier description, in an effort to balance increased error against increased model complexity.

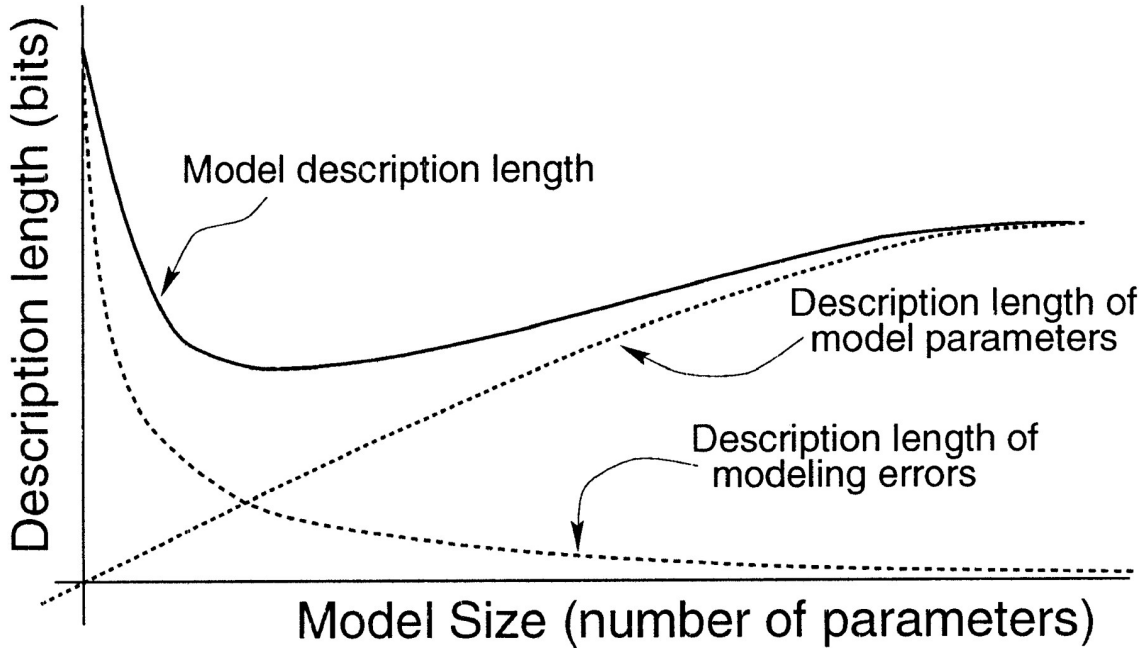


Figure from: M. Small, K. Judd, M. Lowe, S. Stick, " Is breathing in infants chaotic? Dimension estimates for respiratory patterns during quiet sleep" Journal of Applied Physiology, 1999 Vol. 86 no. 1, 359-376

Validation Set Method

- In *Validation Set* methods, one designs the classifiers on the training set, and picks the one that produces the smallest error count on a set of independent samples, called the *validation set*. Note that a third independent set of samples, the *test set*, would be needed to get an unbiased assessment of the true error for the selected classifier. For example, this approach is commonly used to decide when to stop training a neural network.

(DHS Figure 6.6)

Cross-Validation

- If there is not enough data to set aside a validation set, one may simply use a given error estimator on the training data itself to pick the classifier. This approach is only recommended if the error estimator has very good properties for the classification rules and sample size under consideration (e.g., simply using the apparent error will usually not work).

- Cross-validation is a popular choice here, because of its unbiasedness, however one has to keep in mind its large variance, especially in accentuated small-sample situations.