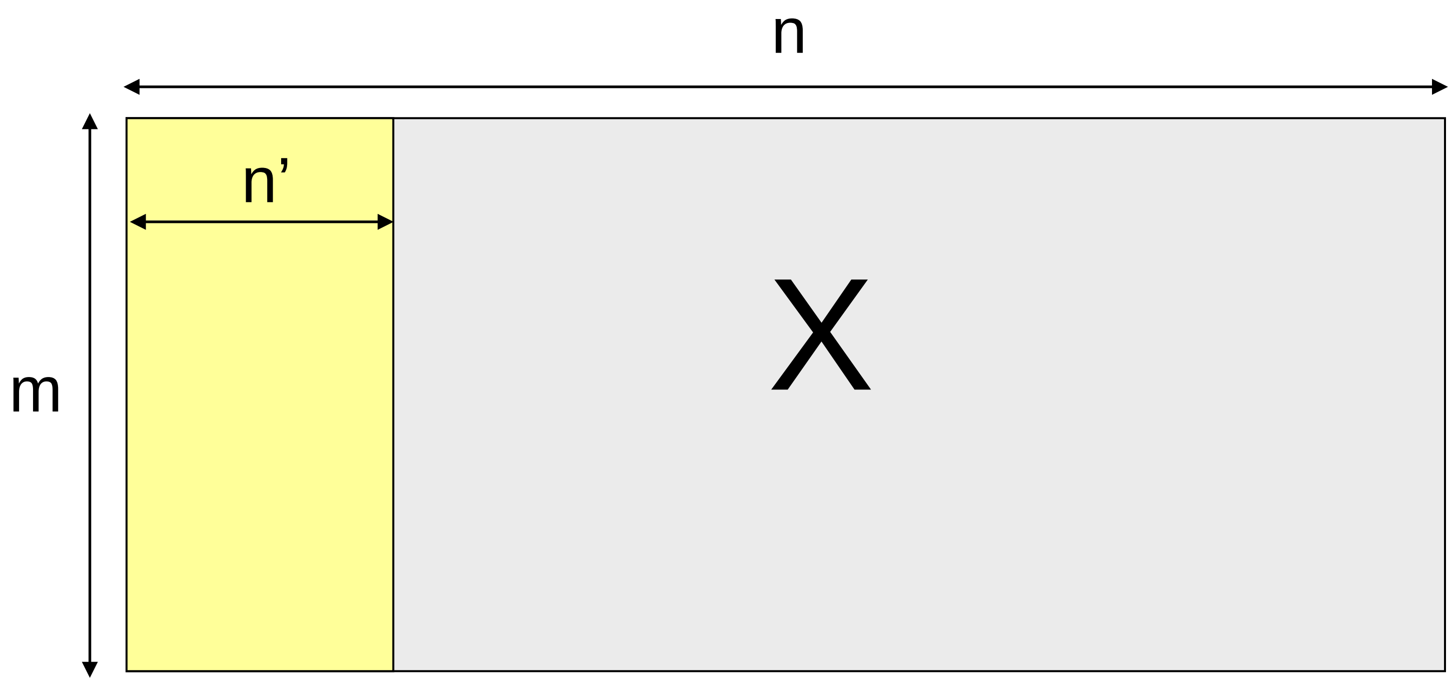
Overview of Feature and Model Selection Techniques

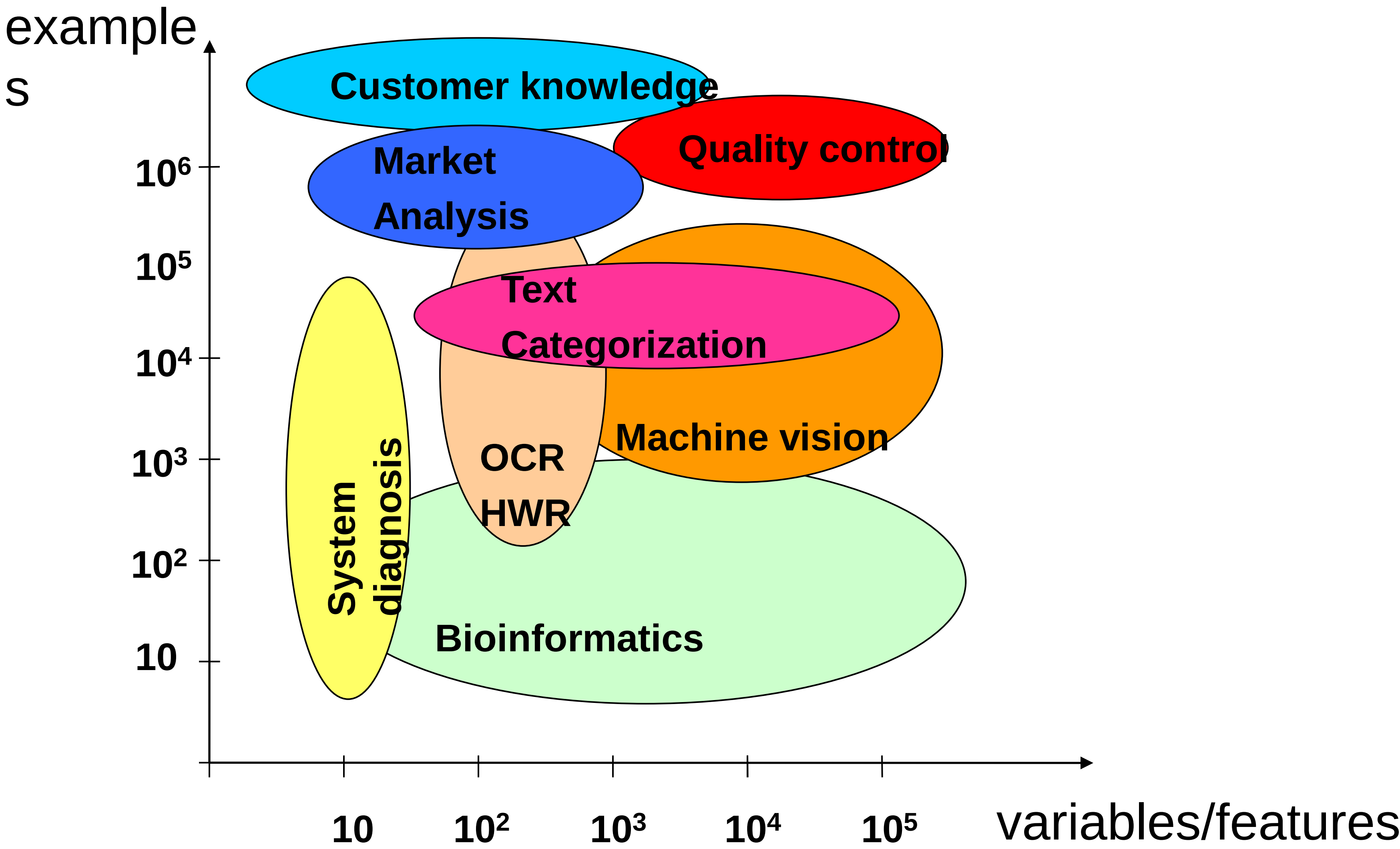
David Li

Feature Selection

- **Thousands to millions of low level features**: select the most relevant one to build **better, faster, and easier to understand** learning machines.



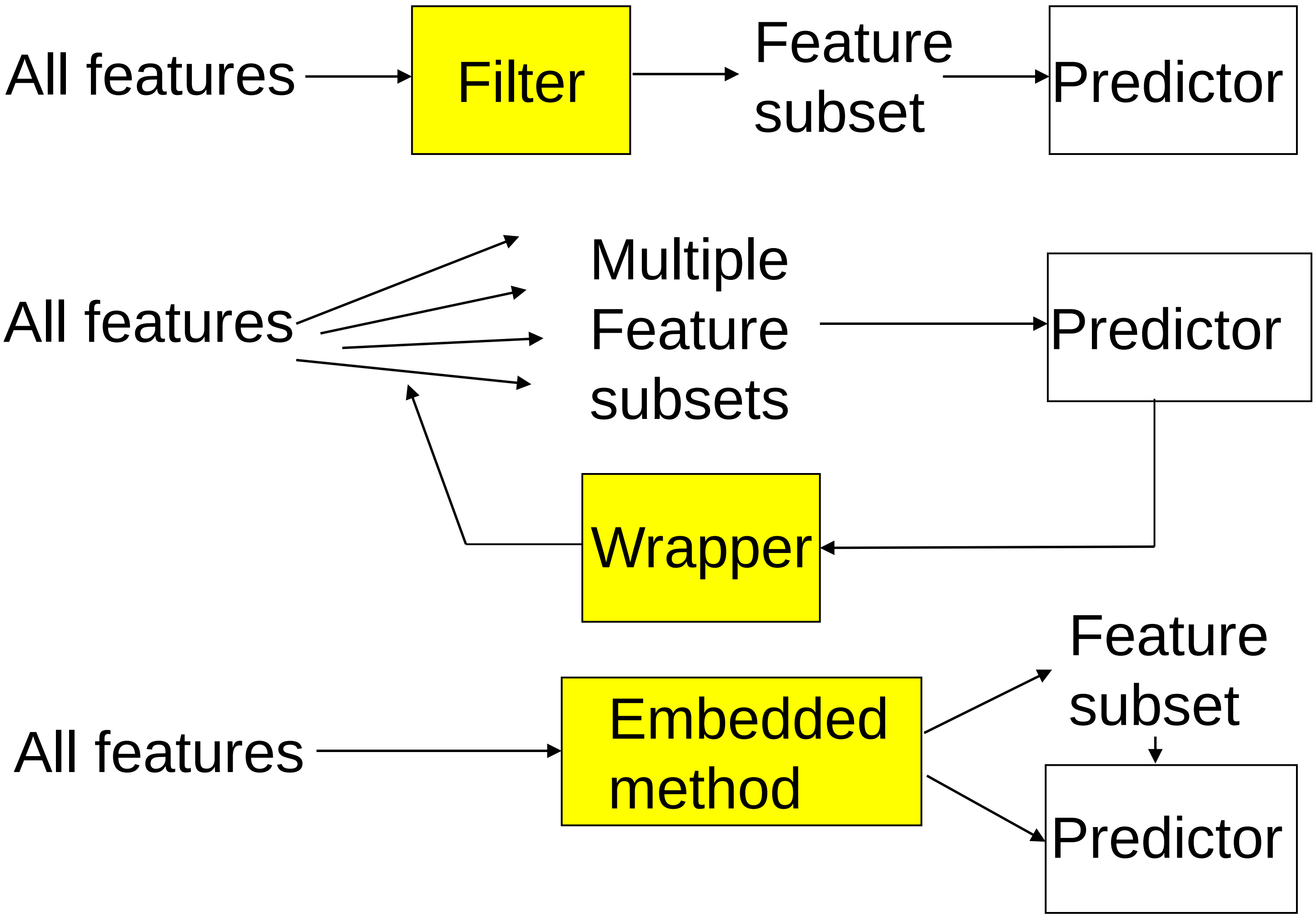
Applications



A taxonomy of feature selection techniques

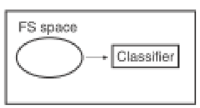
- **Filter method:** ranks features or feature subsets independently of the predictor (classifier).
  - **Univariate method**: considers one variable (feature) at a time.
  - **Multivariate method:** considers subsets of variables (features) together.
- **Wrapper method:** uses a classifier to assess features or feature subsets.
- **Embedded method:** the search for an optimal subset of features is built into the classifier construction, and can be seen as a search in the combined space of feature subsets and hypotheses

Filters,Wrappers, and Embedded methods

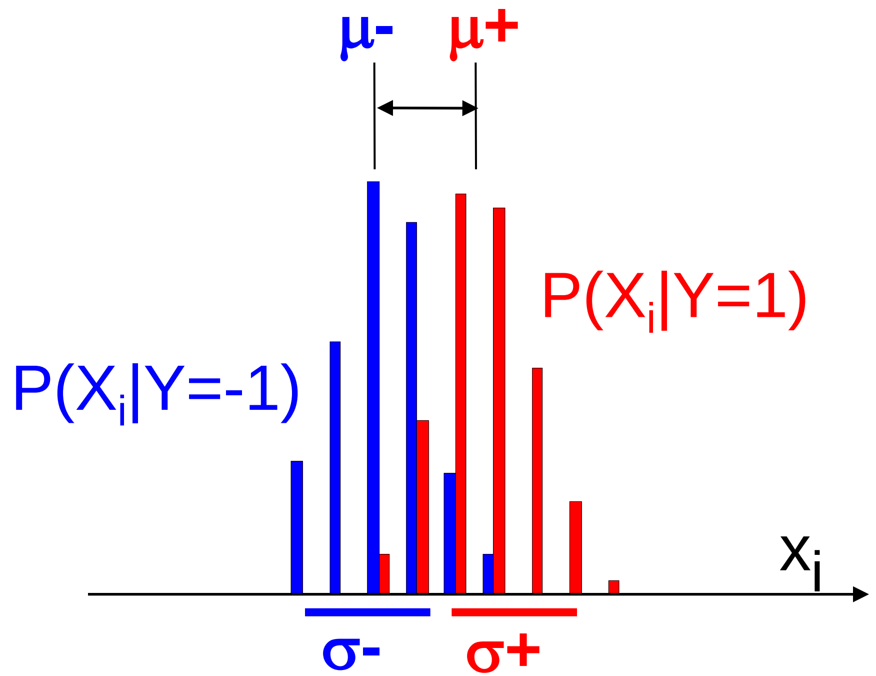


Filter

- **Filter method:** ranks features or feature subsets independently of the predictor (classifier).
  - **Univariate method**: considers one variable (feature) at a time.
  - **Multivariate method:** considers subsets of variables (features) together.

Model search	Advantages	Disadvantages	Examples
Filter	Univariate		
	Fast Scalable Independent of the classifier	Ignores feature dependencies Ignores interaction with the classifier	χ^2 Euclidean distance *i*-test Information gain, Gain ratio (Ben-Bassat, 1982)
	Multivariate		
	Models feature dependencies Independent of the classifier Better computational complexity than wrapper methods	Slower than univariate techniques Less scalable than univariate techniques Ignores interaction with the classifier	Correlation-based feature selection (CFS) (Hall, 1999) Markov blanket filter (MBF) (Koller and Sahami, 1996) Fast correlation-based feature selection (FCBF) (Yu and Liu, 2004)

Univariate feature ranking: two sample t-test

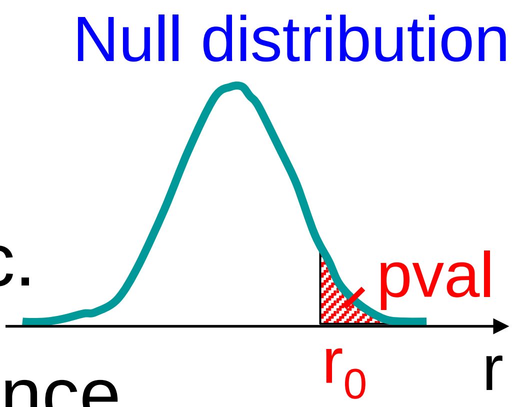


- Normally distributed classes, equal variance σ^2 unknown; estimated from data as σ^2_{within}.
- Null hypothesis H_0: $\mu+ = \mu-$
- **T statistic**: If H_0 is true,

$$t = (\mu+ - \mu-)/(\sigma_{within}\sqrt{1/m^+ + 1/m^-}) \sim \text{Student}(m^+ + m^- - 2 \text{ d.f.})$$

Statistical tests

- H_0: X and Y are independent.
- Relevance index ⇔ test statistic.
- P Value ⇔ Independence evidence
- Multiple testing problem
  - Family-wise error rate: use Bonferroni correction pval ← n pval
  - False discovery rate: FDR = $n_{fp} / n_{selected}$



Information Theory: Univariate Dependence

- Independence:

$$P(X, Y) = P(X)\,P(Y)$$

- Measure of dependence: Mutual Information

$$I(X;Y) = \sum_{x,y} p(x,y) \cdot \log\left(\frac{p(x,y)}{p(x)\,p(y)}\right)$$

- This definition is related to the Kullback-Leibler distance between two distributions
- Measures the dependence of the two distributions
- In feature selection choose the features that minimize I(X;Y) to ensure they are not related.
- I(X;Y) = H(X,Y) – H(X|Y) – H(Y|X) (H, entropy function)

Other criteria

The choice of feature selection ranking methods depends on the nature of:

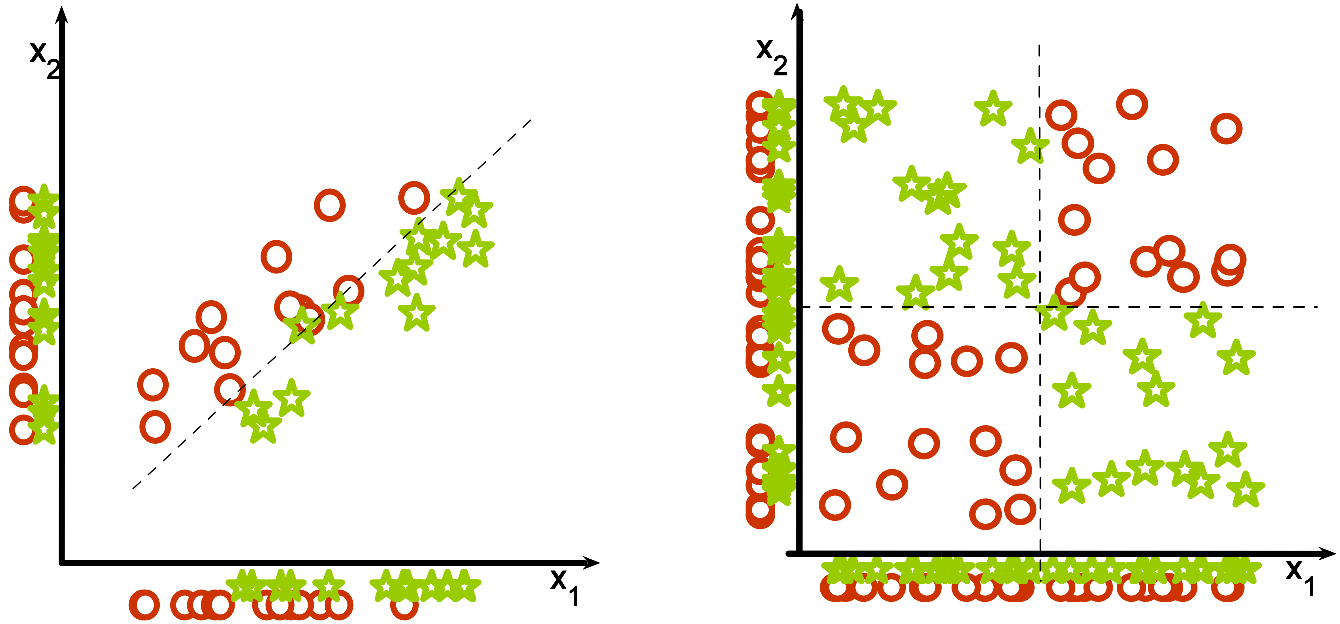
- the variables and the target (binary, categorical, continuous)
- the problem (dependencies between variables, linear/non-linear relationships between variables and target)
- the available data (number of examples and number of variables, noise in data)

Typical cases/methods for testing association of X ~ Y

- (1) X continuous, Y continuous
  - Regression Y ~ b*X+b0
  - Log Likelihood ratio test H0: b==0
- (2) X categorical, Y continuous
  - Method 1
    - Divide Y into K groups based on the K categories that X has
    - K=2, two-sample t-test; K>2, ANNOVA test
  - Method 2: regression Y ~ b*X+ b0 as above
- (3) X continuous, Y categorical
  - Similar as (2) but switch X with Y
- (4) X categorical, Y categorical
  - Fisher test, or chi square test.

Multivariate selection

Univariate selection may fail



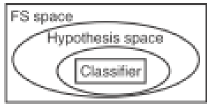
Guyon-Elisseeff, JMLR 2004; Springer 2006

Multivariate: Models feature dependencies

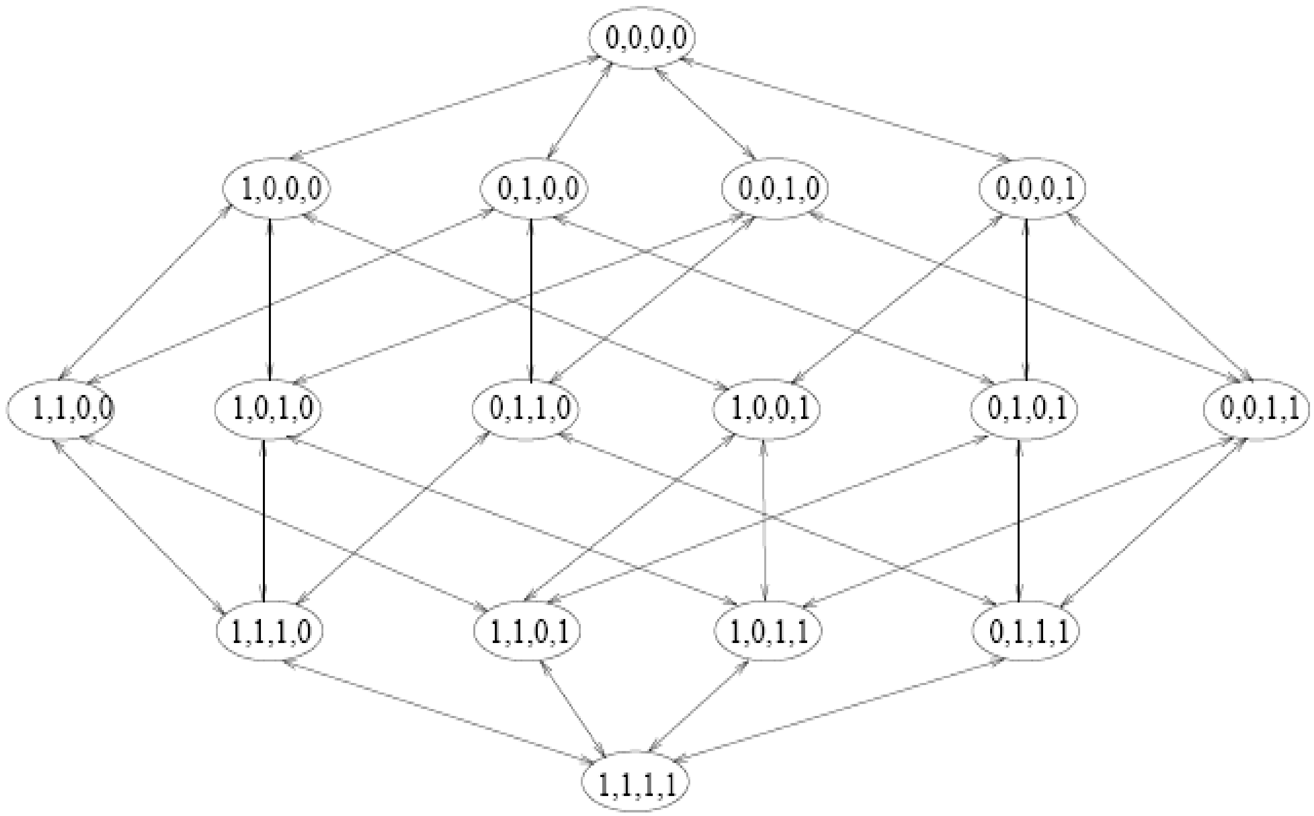
- Correlation-based feature selection (CFS) (Hall, 1999)
- Markov blanket filter (MBF) (Koller and Sahami, 1996)

Wrapper

- **Wrapper method:** uses a classifier to assess features or feature subsets.

Wrapper	Deterministic		
	Simple Interacts with the classifier Models feature dependencies Less computationally intensive than randomized methods	Risk of over fitting More prone than randomized algorithms to getting stuck in a local optimum (greedy search) Classifier dependent selection	Sequential forward selection (SFS) (Kittler, 1978) Sequential backward elimination (SBE) (Kittler, 1978) Plus q take-away r (Ferri *et al.*, 1994) Beam search (Siedelecky and Sklansky, 1988)
	Randomized		
	Less prone to local optima Interacts with the classifier Models feature dependencies	Computationally intensive Classifier dependent selection Higher risk of overfitting than deterministic algorithms	Simulated annealing Randomized hill climbing (Skalak, 1994) Genetic algorithms (Holland, 1975) Estimation of distribution algorithms (Inza *et al.*, 2000)

Wrappers for feature selection

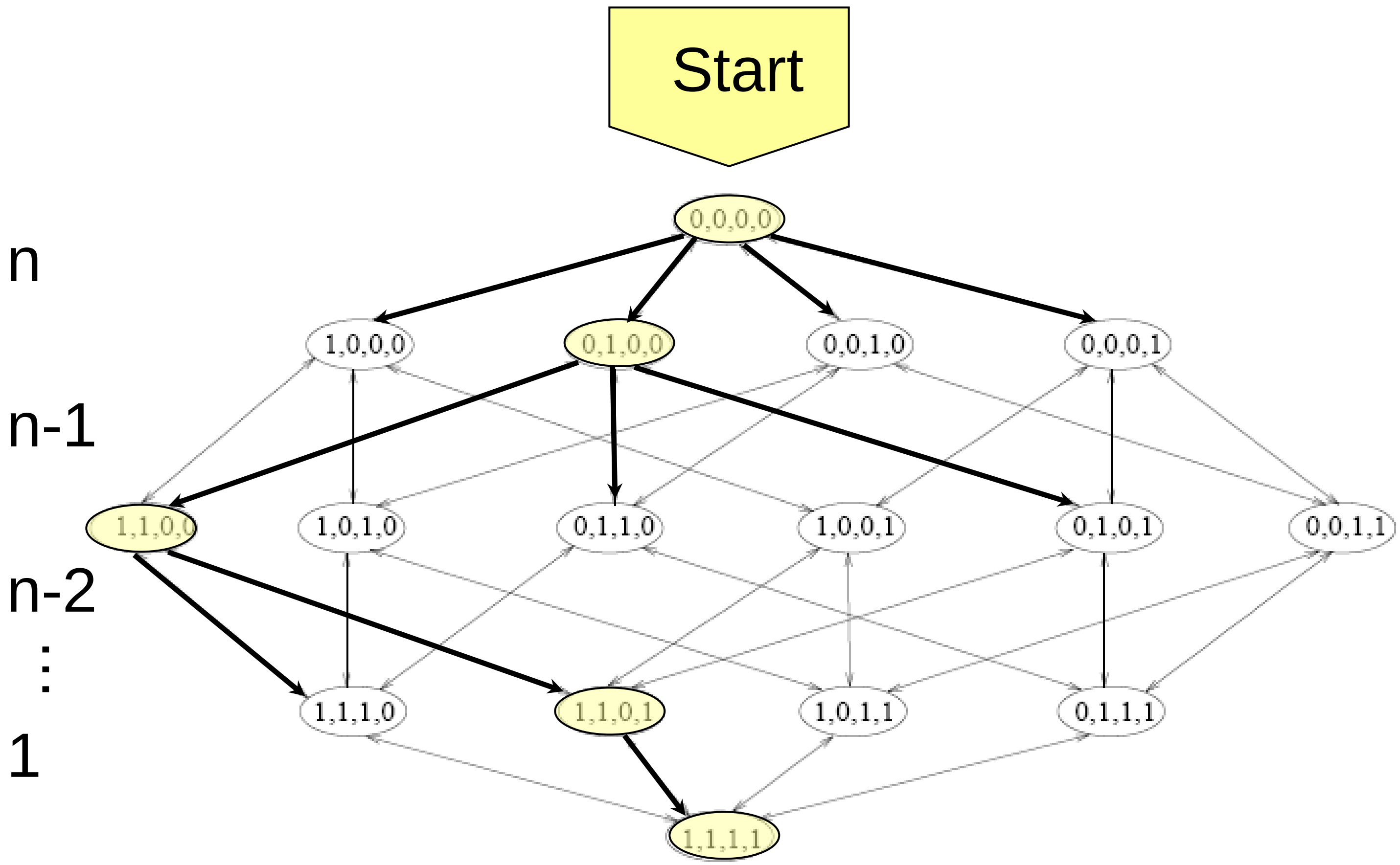


N features, 2^N possible feature subsets!

Search Strategies

- **Exhaustive search.**
- **Simulated annealing, genetic algorithms.**
- **Beam search:** keep k best path at each step.
- **Greedy search:** forward selection or backward elimination.

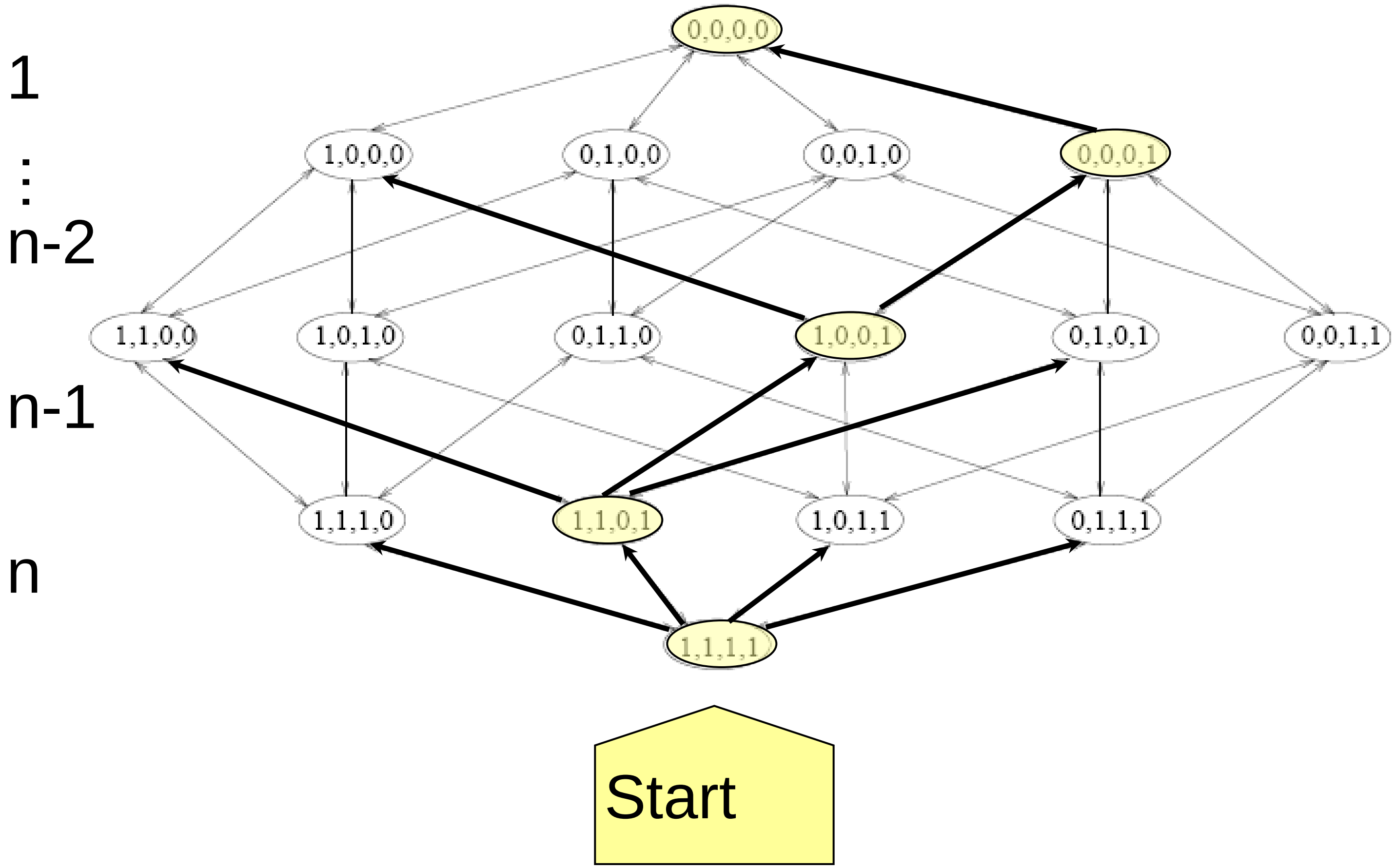
Forward Selection (wrapper)



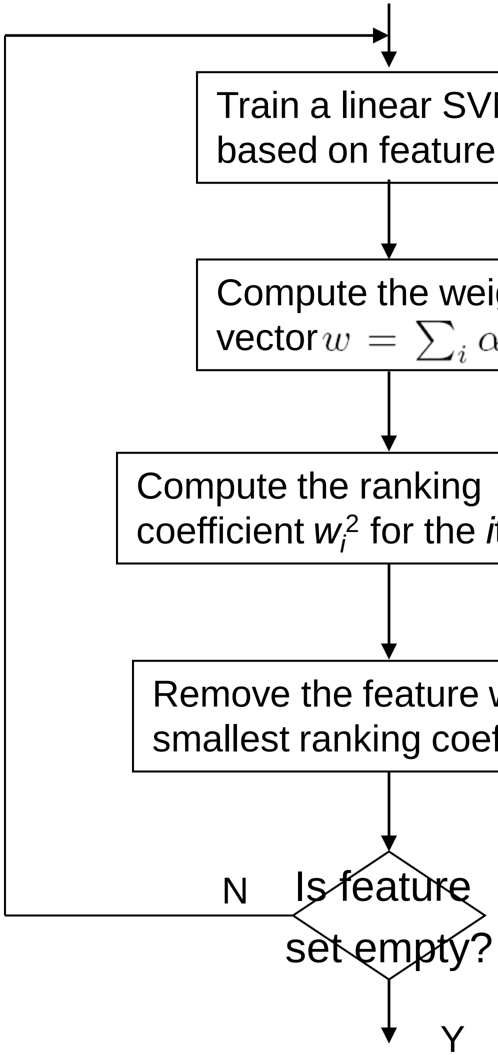
Also referred to as SFS: Sequential Forward Selection

Backward Elimination (wrapper)

Also referred to as SBS: Sequential Backward Selection



Support Vector Machine Recursive Feature Elimination (SVM-RFE)



Embedded method

- **Embedded method:** the search for an optimal subset of features is built into the classifier construction, and can be seen as a search in the combined space of feature subsets and hypotheses
  - **Features are selected while the classifier is built.**

- Embedded methods are therefore not too far from wrapper techniques and can be extended to multiclass, regression, etc…

- Feature selection by **regularization** (penalization)

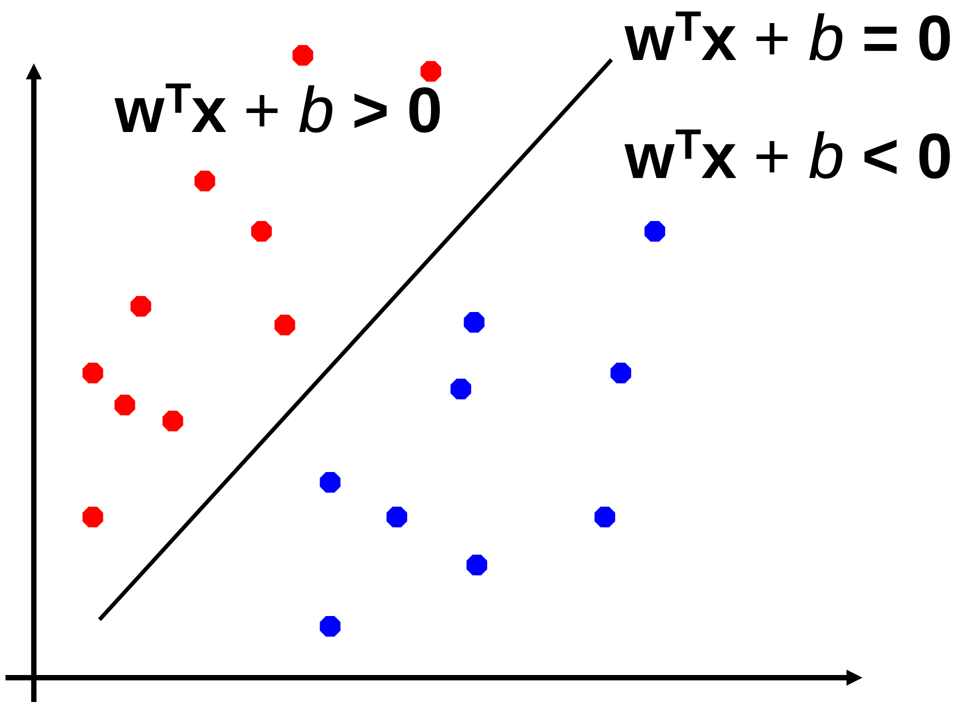
Feature selection by penalization

- Dimension reduction
  - Partial least squares
  - Principal component analysis
  - ...
- Feature selection
  - Filter
  - Wrapper
  - Embedded
    - Dicision trees
    - Naive Bayes
    - ...
    - **Penalized methods**
      - Classification objective function
        - Likelihood functions
        - LDA by optimal scoring
        - AUCs
        - SVM
        - ...
      - Penalt y functions
        - Lasso
        - Adaptive Lasso
        - Bridge
        - Elastic Net
        - SCAD
        - ...
      - ...
- Hybrid method
  - Sparse principal component analysis
  - ...

Figure I: A taxonomy of feature selection and dimension reduction.

Linear Separators revisited

- Binary classification can be viewed as the task of separating classes in feature space:



$$f(\mathbf{x}) = \text{sign}(\mathbf{w}^T\mathbf{x} + b)$$

Logistic Regression Model

- The log-ratio of positive class to negative class

$$\log \frac{p(y=1|\vec{x})}{p(y=-1|\vec{x})} = \vec{x}\cdot\vec{w}+c \quad \Rightarrow \quad \frac{p(y=1|\vec{x})}{p(y=-1|\vec{x})} = \exp(\vec{x}\cdot\vec{w}+c)$$

$$p(y=1|\vec{x}) + p(y=-1|\vec{x}) = 1$$

Logistic Regression Model

- The log-ratio of positive class to negative class

$$\log \frac{p(y=1|\vec{x})}{p(y=-1|\vec{x})} = \vec{x}\cdot\vec{w}+c \quad \Longrightarrow \quad \begin{aligned} &\frac{p(y=1|\vec{x})}{p(y=-1|\vec{x})} = \exp(\vec{x}\cdot\vec{w}+c) \\ &p(y=1|\vec{x}) + p(y=-1|\vec{x}) = 1 \end{aligned}$$

- Results

$$\left.\begin{aligned} p(y=-1|\vec{x}) &= \frac{1}{1+\exp(\vec{x}\cdot\vec{w}+c)} \\ p(y=1|\vec{x}) &= \frac{1}{1+\exp(-\vec{x}\cdot\vec{w}-c)} \end{aligned}\right\} \Rightarrow p(y|\vec{x}) = \frac{1}{1+\exp\left[-y(\vec{x}\cdot\vec{w}+c)\right]}$$

Logistic Regression Model

- Assume the inputs and outputs are related in the log linear function

$$p(y \mid \vec{x};\theta) = \frac{1}{1+\exp\left[-y(\vec{x}\cdot\vec{w}+c)\right]}$$

$$\theta = \{w_1, w_2, ..., w_d, c\}$$

- Estimate weights: MLE approach $\{w_1, w_2, ..., w_d, c\}$

$$\left\{\vec{w}, c\right\}^* = \max_{\vec{w},c} l(D_{train}) = \max_{\vec{w},c} \sum_{i=1}^{n} \log p(y_i \mid \vec{x}_i;\theta)$$

$$= \max_{\vec{w},c} \sum_{i=1}^{n} \log \frac{1}{1+\exp(-y\left[\vec{x}\cdot\vec{w}+c\right])}$$

Problems with Logistic Regression

$$\{\vec{w},c\}^* = \max_{\vec{w},c} \sum_{i=1}^{n} \log \frac{1}{1+\exp(-y[\vec{x}\cdot\vec{w}+c])}$$

- Convex objective function
- Non-existence of Maximum Likelihood Estimates
  - Complete Separation
  - Quasi-complete Separation
  - All samples from one class
  - These configurations produce nonunique infinite estimates. If the iterative process of maximizing the likelihood function is allowed to continue, the log likelihood diminishes to 0, and the dispersion matrix becomes unbounded.
- If neither complete nor quasi-complete separation exists in the sample points, there is an overlap of sample points. In this configuration, the maximum likelihood estimates exist and are unique.

Solution: L2 Regularization

- L2 Regularized log-likelihood

$$l_{reg}(D_{train}) = l_{reg}(D_{train}) - \lambda \|\vec{w}\|^2$$

$$= \sum_{i=1}^{n} \log p(y_i \mid \vec{x}_i; \theta) - \lambda \sum_{j=1}^{p} w_j^2$$

$$= \boxed{\sum_{i=1}^{n} \log \frac{1}{1+\exp(-y_i[\vec{x}_i \cdot \vec{w} + c])}} - \boxed{\lambda \sum_{j=1}^{p} w_j^2}$$

- $\lambda||w||^2$ is called the L2 regularizer
  - Favors small weights
  - Prevents weights from becoming too large

Sparse Solution

- What does the solution of regularized logistic regression look like ?
- A sparse solution
  - Most weights are small and close to zero

Why do We Need Sparse Solution?

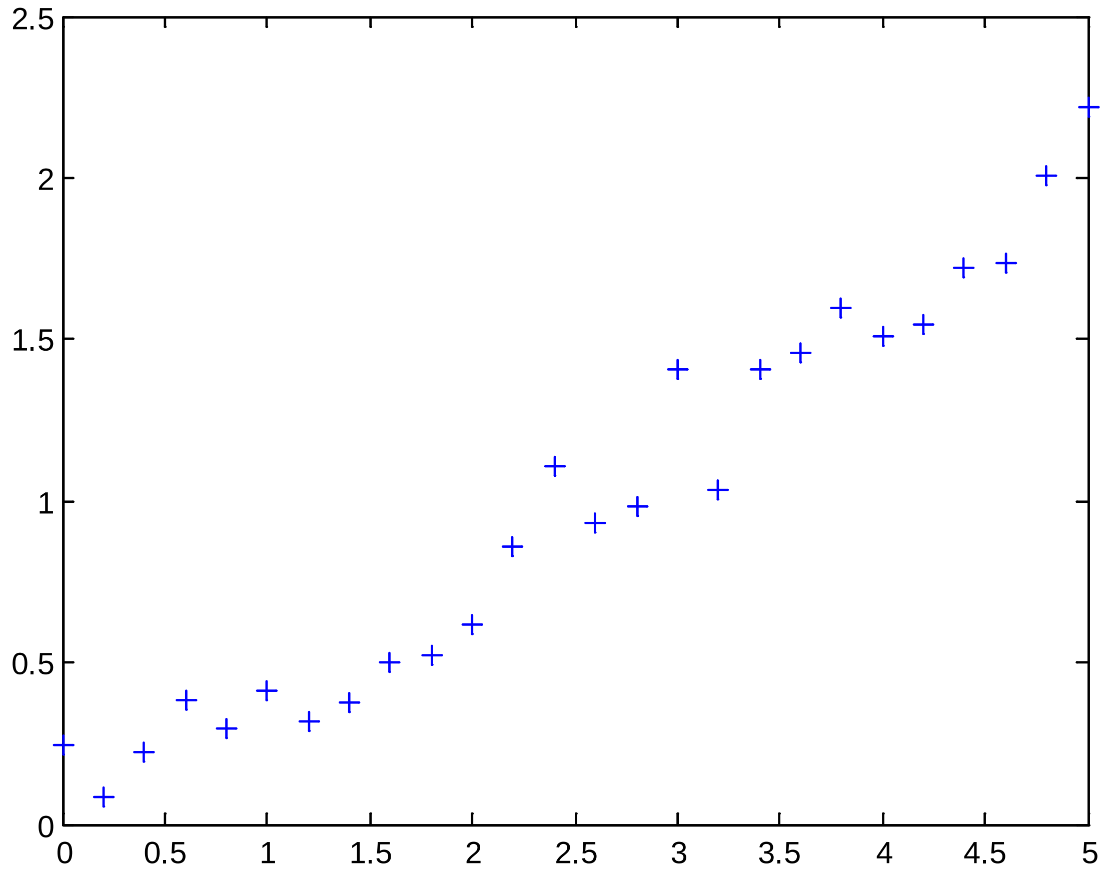
- Two types of solutions
  1. Many non-zero weights but many of them are small
  2. Only a small number of non-zero weights, and many of them are large
- Occam's Razor: the simpler the better
  - A simpler model that fits data unlikely to be coincidence
  - A complicated model that fit data might be coincidence
  - Smaller number of non-zero weights

    → less amount of evidence to consider

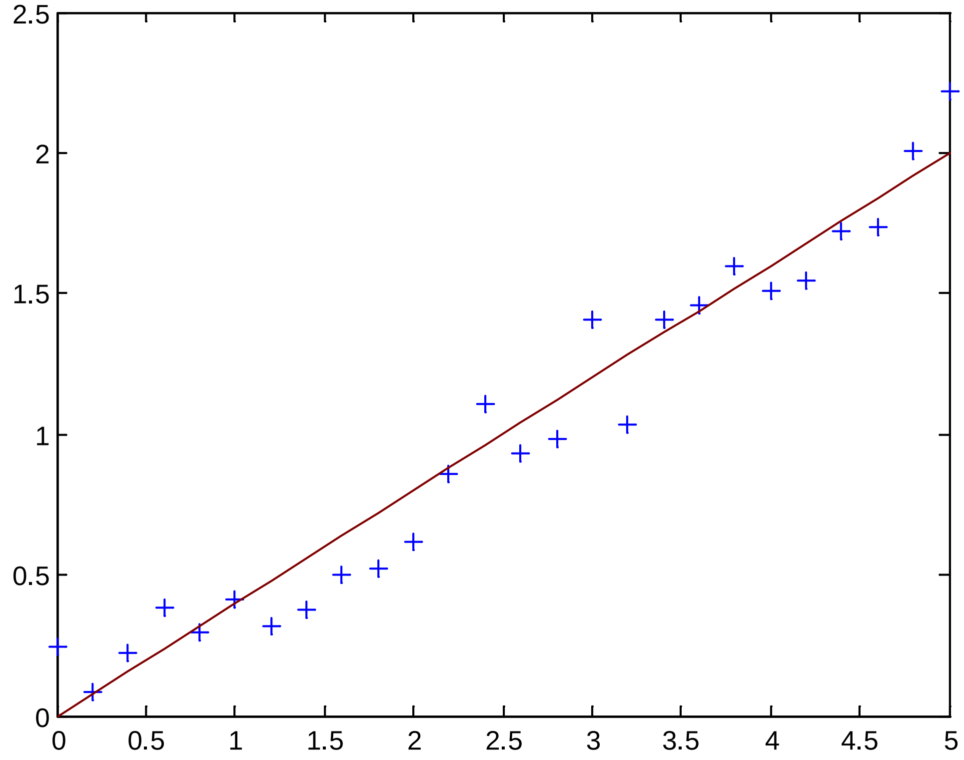
    → simpler model

    → case 2 is preferred

Occam’ s Razer

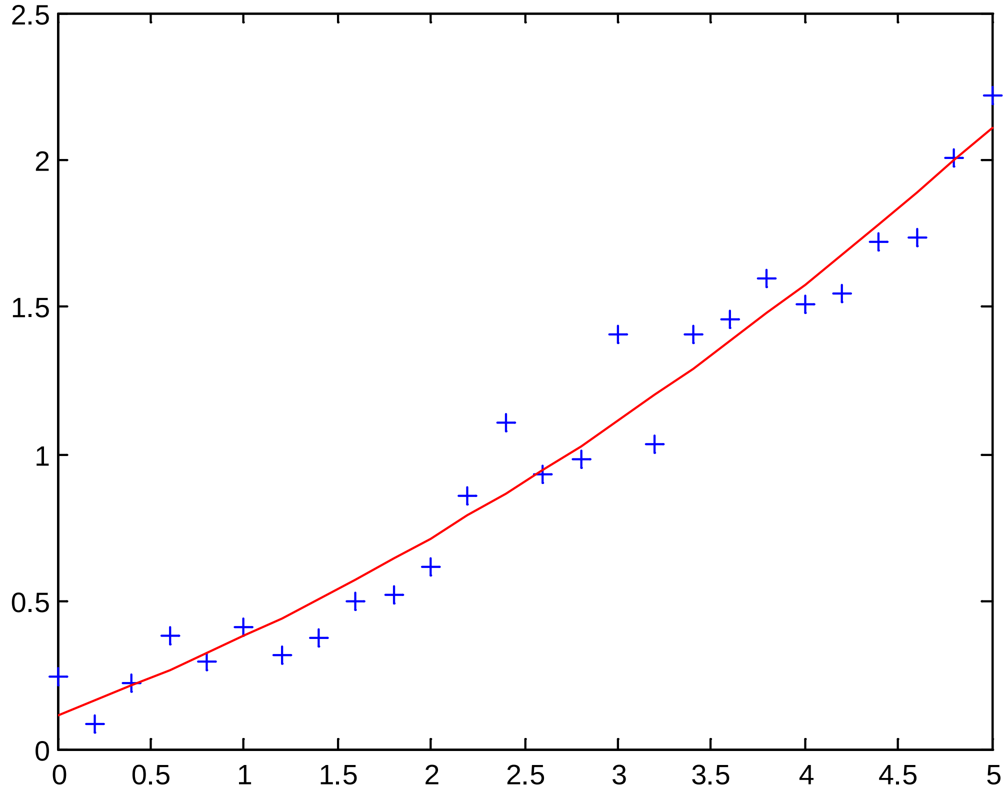


Occam's Razer: Dimensionality = 1



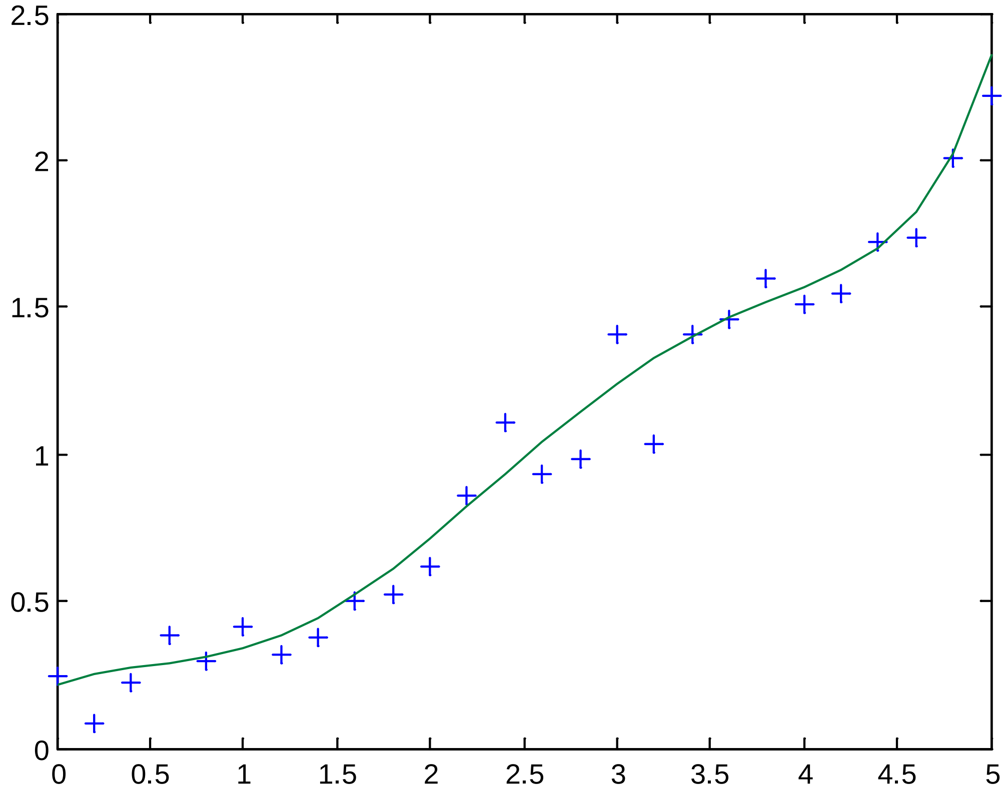
$$y = a_1 x$$

Occam's Razer: Dimensionality = 3



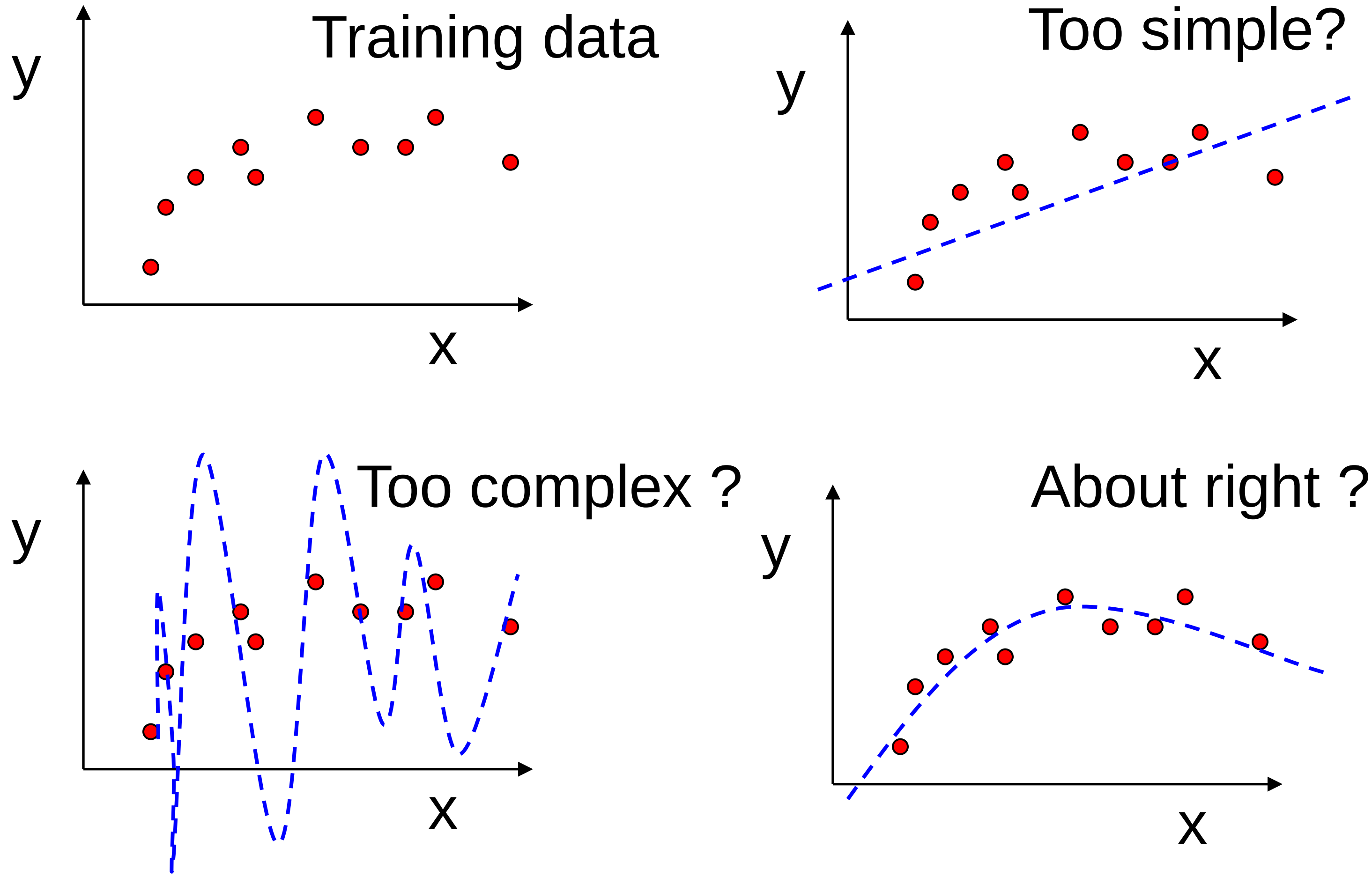
$$y = a_1 x + a_2 x^2 + a_3 x^3$$

Occam's Razor: Dimensionality = 10

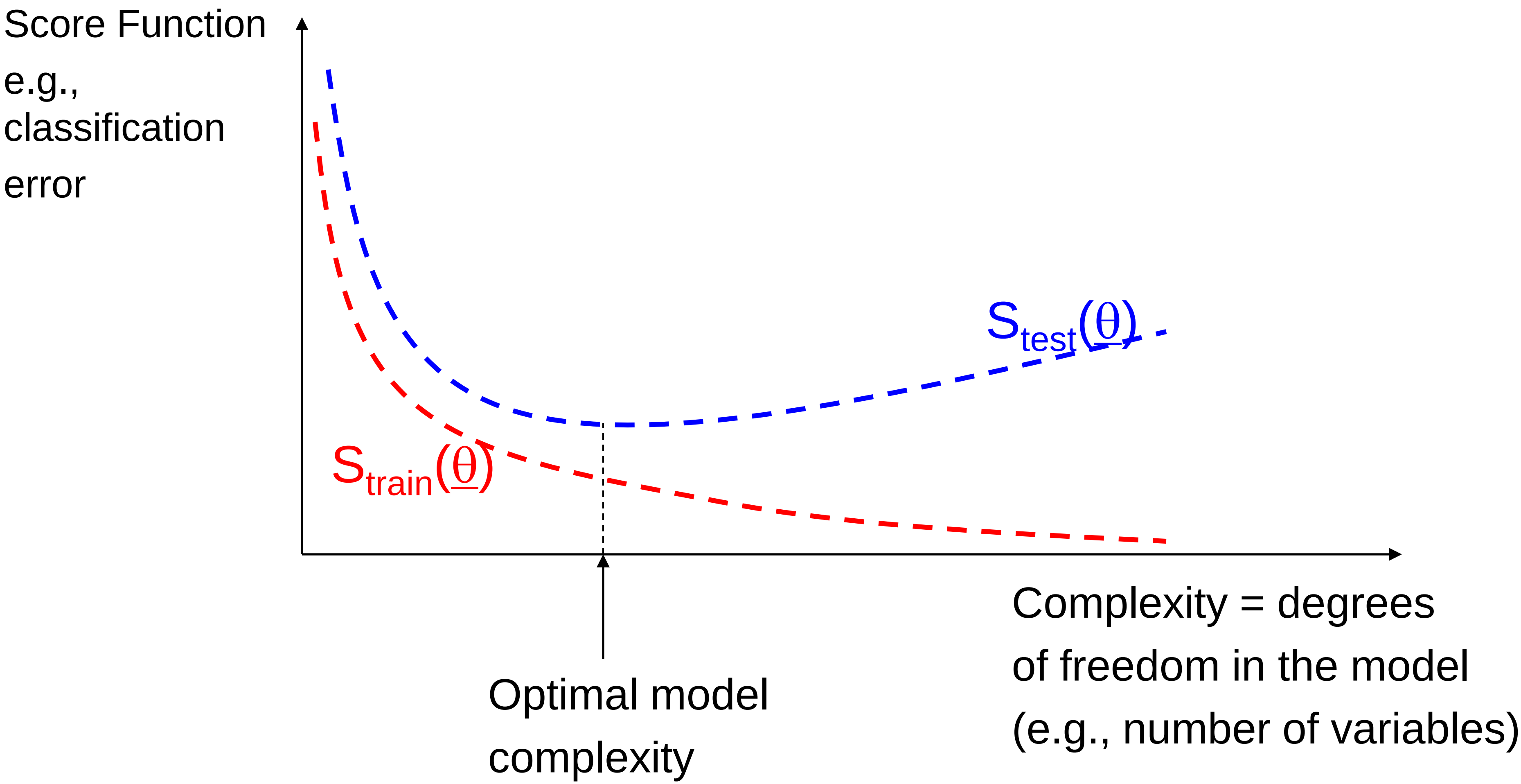


$$y = a_1 x + a_2 x^2 + \ldots + a_{10} x^{10}$$

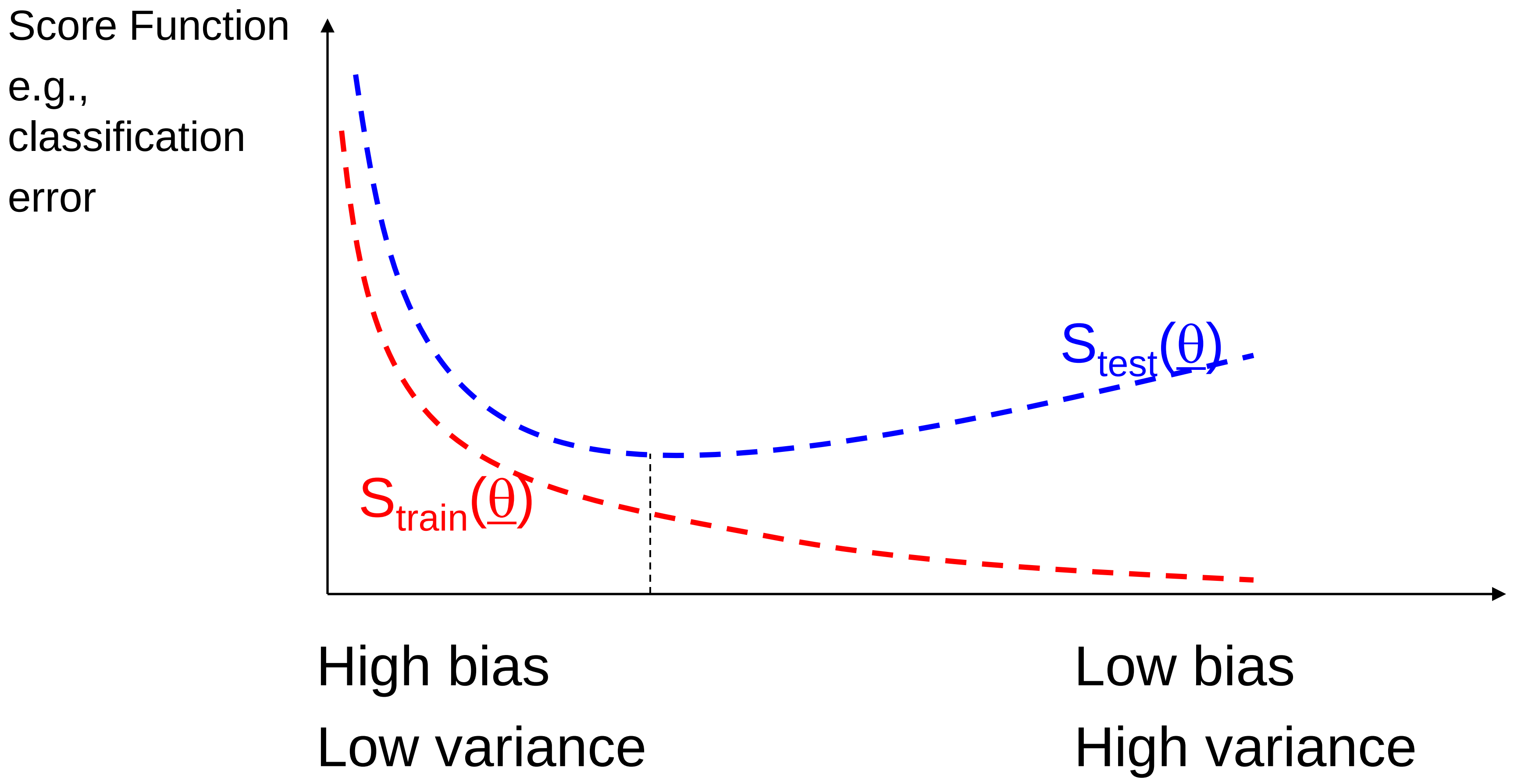
Complexity versus Goodness of Fit



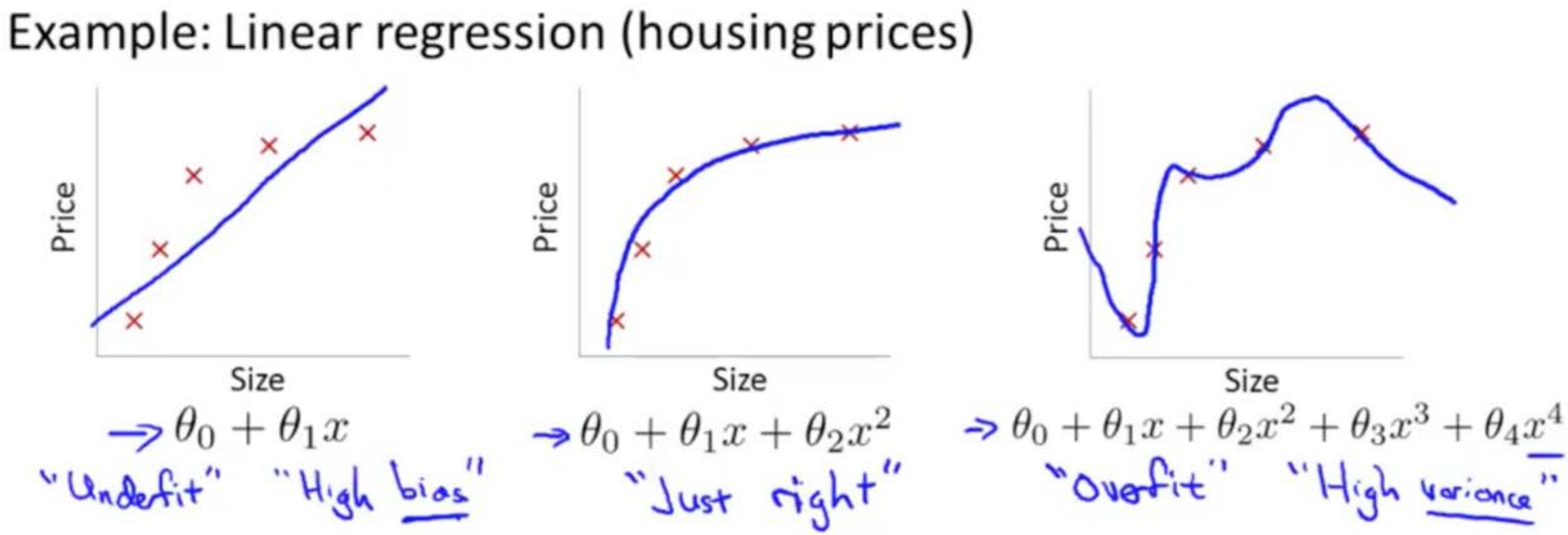
Complexity and Generalization



Complexity and Generalization



Complexity and Generalization

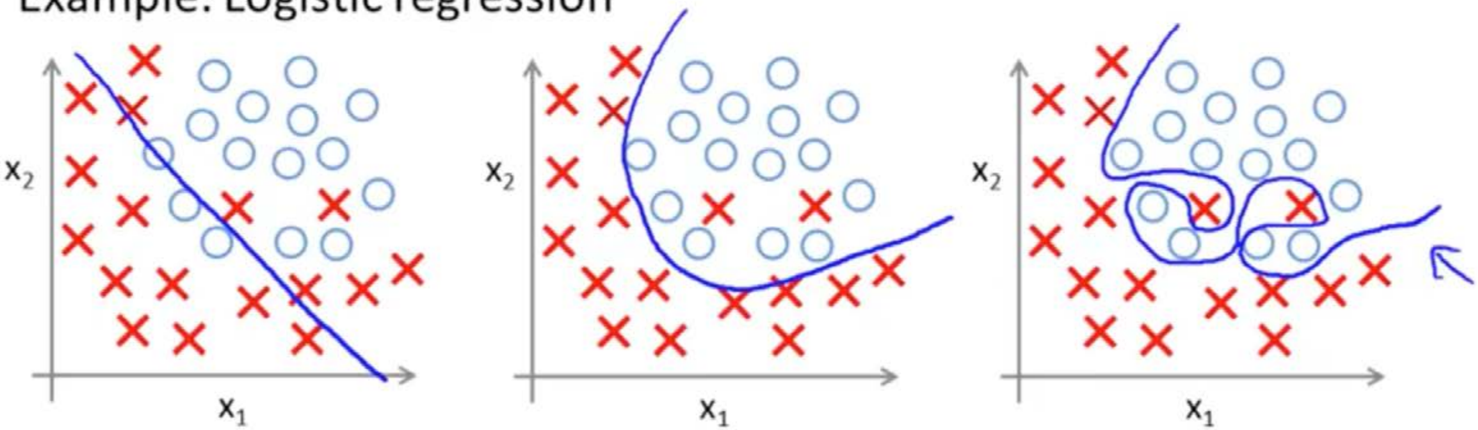


Overfit
- High Variance
- Too many features
- Fit well but fail to generalize new examples

Underfit
- High Bias

Complexity and Generalization

Example: Logistic regression



$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$

(g = sigmoid function)

"Underfit"

$g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2 + \theta_5 x_1 x_2)$

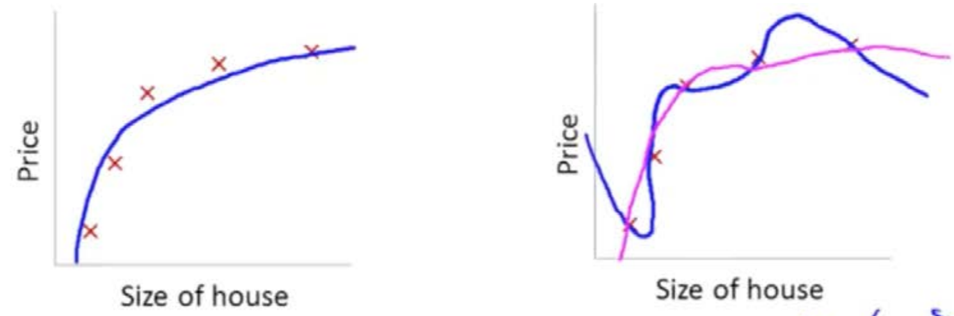
$g(\theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \theta_3 x_1^2 x_2 + \theta_4 x_1^2 x_2^2 + \theta_5 x_1^2 x_2^3 + \theta_6 x_1^3 x_2 + \ldots)$

"Overfit"

Solutions to Overfitting
- Reduce number of features
- Manually select features to keep
- Model selection algorithm

Regularization
- Keep all features, but reduce magnitude or values of coefficients
- Works well when we've a lot of features

Control Complexity and Generalization by Regularization



$\theta_0 + \theta_1 x + \theta_2 x^2$

$\theta_0 + \theta_1 x + \theta_2 x^2 + \cancel{\theta_3 x^3} + \cancel{\theta_4 x^4}$

Suppose we penalize and make θ_3, θ_4 really small.

$$\rightarrow \min_\theta \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 + 1000\,\theta_3^2 + 1000\,\theta_4^2$$

$\theta_3 \approx 0 \quad \theta_4 \approx 0$

Regularization

- Small values for coefficients
- "Simpler" hypothesis
- Less prone to overfitting

First goal: fit training set well (first term in green)
Second goal: keep parameters small (second term in yellow)

Revisited: L2 Logistic Regression

- Penalize by sum-of-squares of parameters

$$\hat{\mathbf{w}} = \arg\max_{\vec{w}} l_{reg}(D_{train})$$

$$= \arg\min_{\vec{w}} \sum_{i=1}^{n} \log(1 + \exp(-y_i[\vec{x}_i \cdot \vec{w} + c])) + \lambda \sum_{j=1}^{p} w_j^2$$

- Or

$$\hat{\mathbf{w}} = \arg\min_{\vec{w}} \sum_{i=1}^{n} \log(1 + \exp(-y_i[\vec{x}_i \cdot \vec{w} + c]))$$

$$\text{subject to } \sum_{j=1}^{p} w_j^2 \le t$$

Sparse Solution: L1 (Lasso) Logistic Regression

- Penalize by absolute value of parameter

$$\hat{\mathbf{w}} = \arg\min_{\vec{w}} \sum_{i=1}^{n} \log(1+\exp(-y_i[\vec{x}_i \cdot \vec{w} + c])) + \lambda \sum_{j=1}^{p} |w_j|$$

or

$$\hat{\mathbf{w}} = \arg\min_{\vec{w}} \sum_{i=1}^{n} \log(1+\exp(-y_i[\vec{x}_i \cdot \vec{w} + c]))$$

$$\text{subject to } \sum_{j=1}^{p} |w_j| \le t$$

- $\lambda|w|$ is called the L1 regularizer (Lasso for L1 linear regression)
  - Shrinks tiny weights to zero!

Why L1 is sparse?

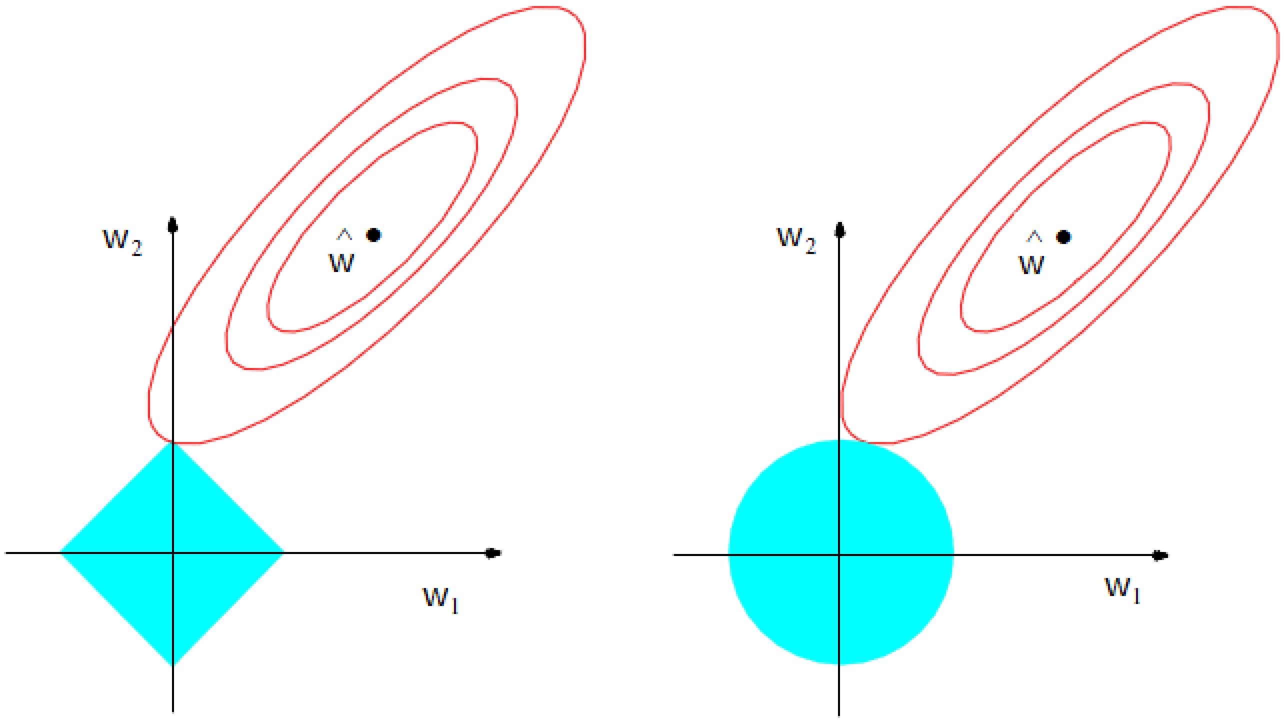


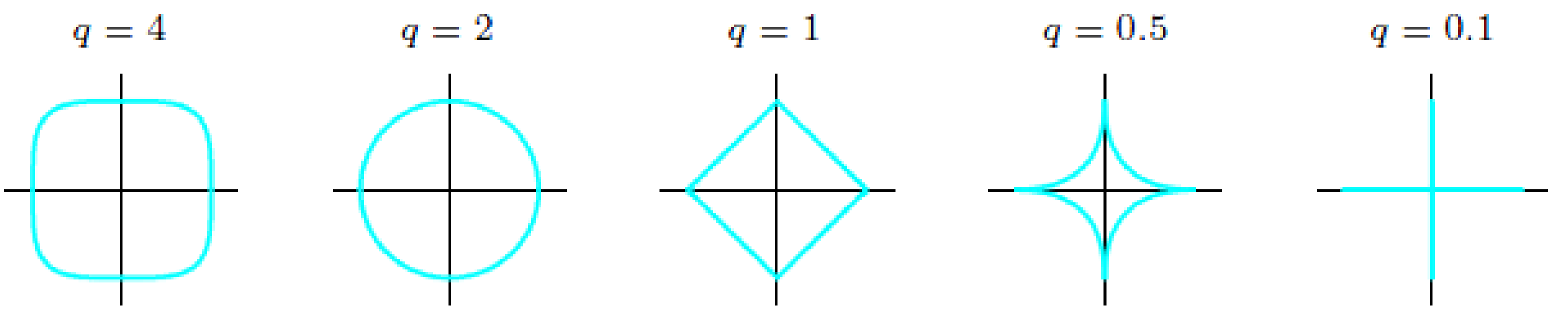
FIGURE 3.11. *Estimation picture for the lasso (left) and ridge regression (right). Shown are contours of the error and constraint functions. The solid blue areas are the constraint regions* $|w_1| + |w_2| \leq t$ *and* $w_1^2 + w_2^2 \leq t^2$, *respectively, while the red ellipses are the contours of the least squares error function.*

Lq regularization

- Regularized by $\lambda|w_j|^q$

$$\hat{\mathbf{w}} = \arg\min_{\vec{w}} \sum_{i=1}^{n} \log(1+\exp(-y_i[\vec{x}_i \cdot \vec{w} + c]))$$

$$\text{subject to } \sum_{j=1}^{p} | w_j |^q \le t$$



Applications of Penalization

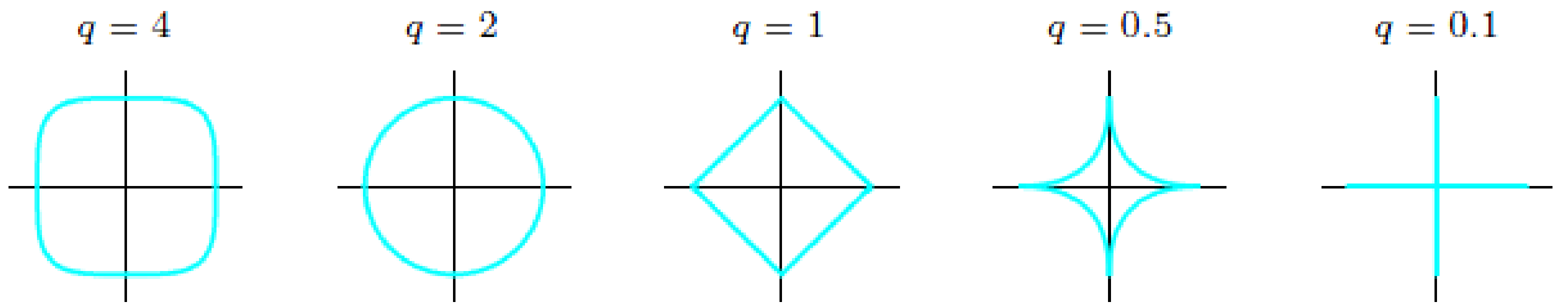
- Ridge: L2; Lasso: L1; Bridge: Lq, $0<q\leq 1$; Adaptive Lasso: weighted L1
- Elastic Net: Bridge + Ridge ; SCAD: piecewise L1+L2

Table 1: Published articles that use penalized classification methods for microarray data (incomplete list)

Author	Objective function	Penalty	Numerical study
Ghosh and Chinnaiyan [9]	LDA	Lasso	Simulation; microarray data
Zhang *et al*. [10]	SVM	SCAD	Simulation; microarray data; metabolism data
Liu *et al*. [11]	Likelihood	Elastic net/bridge	Simulation; methylation data; microarray data
Ma and Huang [32]	ROC	Lasso	Microarray data
Pan *et al*. [50]	Likelihood	Adaptive Lasso	Simulation; microarray data
Roth [43]	Likelihood	Lasso	Microarray data
Segal *et al*. [45]	Likelihood	Lasso	Microarray data
	SVM	Ridge	Microarray data
Shen and Tan [28]	Likelihood	Ridge	Microarray data
Zhu and Hastie [27]	Likelihood	Ridge	Microarray data
Zou and Hastie [54]	Likelihood	Elastic net	Microarray data

L1 and Bridge penalty

- For feature selection purpose $q \leq 1$
- Bridge: Lq, $0 < q \leq 1$;
- L1 is the only q being convex optimization problem → computationally feasible
- In general, L1 is not variable selection consistent in the sense that the whole Lasso path may not contain the true model when sample size → ∞
- For linear models with $n >> p$ the bridge penalty with $q < 1$ is feature selection consistent; when $n << p$, under certain conditions, consistency holds too

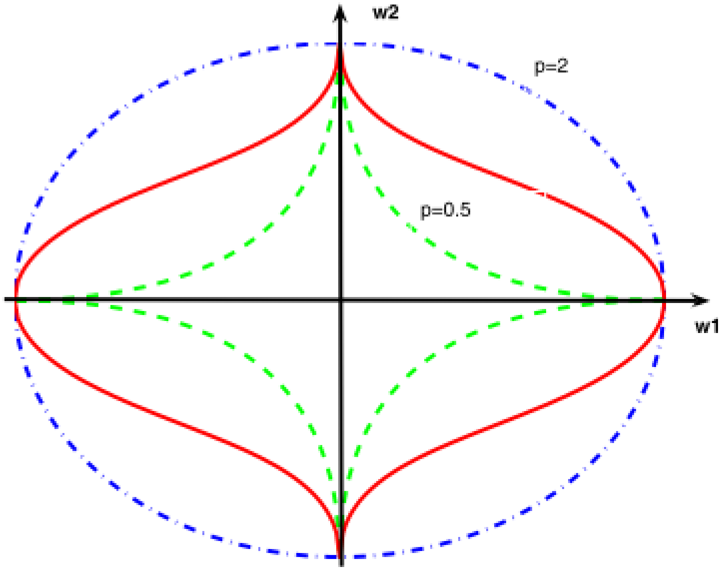


Elastic Net Penalty

- For linear models, when there exist highly correlated input variables, the Lasso (L1) tends to select only one of the correlated variables (Zou and Hastie, JRSSB, 2005)
- One penalty that can effectively deal with high correlations is the elastic net penalty:

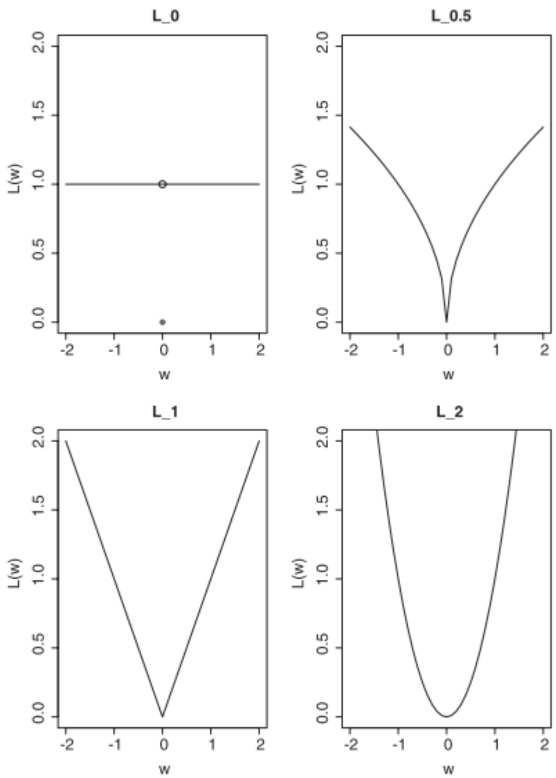
$$\text{pen}(\beta) = \sum_j |\beta_j|^\gamma + \left(\sum_j \beta_j^2\right)^\eta, \qquad (11)$$

with $0 < \gamma \leq 1$ and $\eta \geq 1$. That is, the elastic net is a mixture of bridge type penalties. Zou and Hastie [54] proposes $\gamma = 1$ and $\eta = 1$. In Liu *et al.* [11], it is extended to $\gamma < 1$ and $\eta = 1$. Applications of the elastic net in bioinformatics classification are considered in Liu *et al.* [11].



SCAD Penalty

- For L1 and Elastic net, the penalty increases with $|\beta|$ → biased estimates of large coefficients



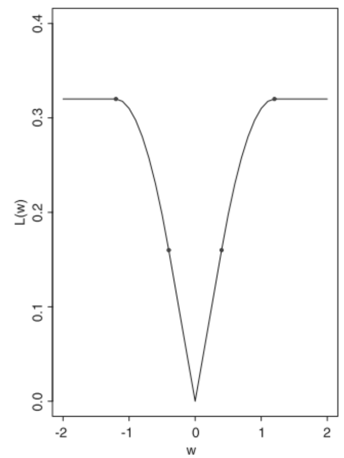


Fig. 2. The SCAD penalty function with $\lambda = 0.4$ and $a = 3$.

$$p_\lambda(|w|) = \begin{cases} \lambda|w| & \text{if } |w| \le \lambda, \\ -\frac{(|w|^2 - 2a\lambda|w| + \lambda^2)}{2(a-1)} & \text{if } \lambda < |w| \le a\lambda, \\ \frac{(a+1)\lambda^2}{2} & \text{if } |w| > a\lambda, \end{cases}$$

The L_1 & SCAD SVM

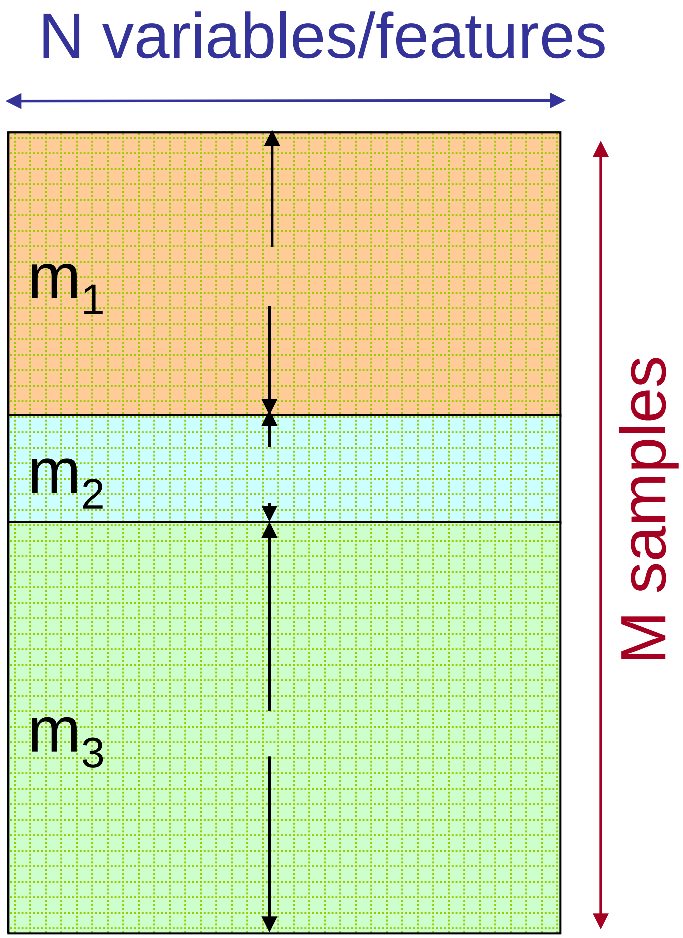
- The objection function of SVM can be written as

The SVM finds $f(\mathbf{x}) = b + \mathbf{w} \cdot h(\mathbf{x})$ by minimizing

$$\frac{1}{n}\sum_{i=1}^{n}[1 - y_i(b+\mathbf{w}\cdot h(\mathbf{x}_i))]_+ + \lambda\|w\|^2,$$

- A version of SVM where $\Omega(w)=||w||^2$ is replaced by the L_1 norm $\Omega(w)=\sum_i |w_i|$ *(Bi et al 2003, Zhu et al, 2003)*
- Also can be replaced by SCAD penalty , (*Zhang 2006*).
- Can be considered an embedded feature selection method:
  - Some weights will be drawn to zero (tend to remove redundant features)
  - Difference from the regular SVM where redundant features are included

Bilevel optimization (including tuning penalty parameter)



Split data into 3 sets:

training, validation, and test set.

1) For each feature subset, train predictor on training data.
2) Select the feature subset, which performs best on validation data.
   – Repeat and average if you want to reduce variance (cross-validation).
3) Test on test data.

A two–step strategy for ultrahigh dimensionality

- Fan, J. and Lv, J. (2008). Sure independence screening for ultrahigh dimensional feature space (with discussion). *Journal of the Royal Statistical Society Series B* 70, 849–911.
  - Step 1: Simple univariate method to reduce p to d (d<n)
  - Step 2: embedded method (regularization) to reduce d to d’

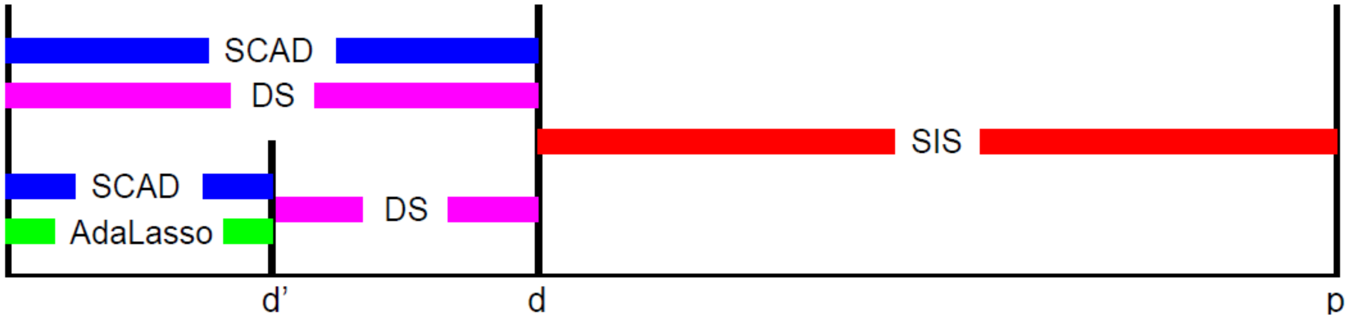


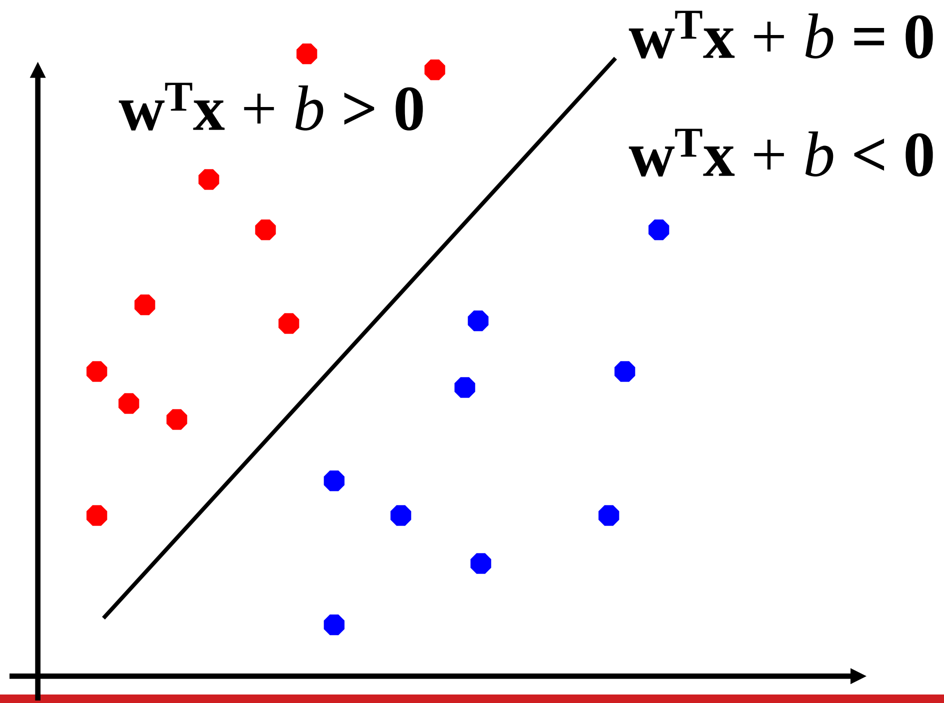
Figure 2: Methods of model selection with ultra high dimensionality.

Support Vector Machines

David Li

Linear Separators

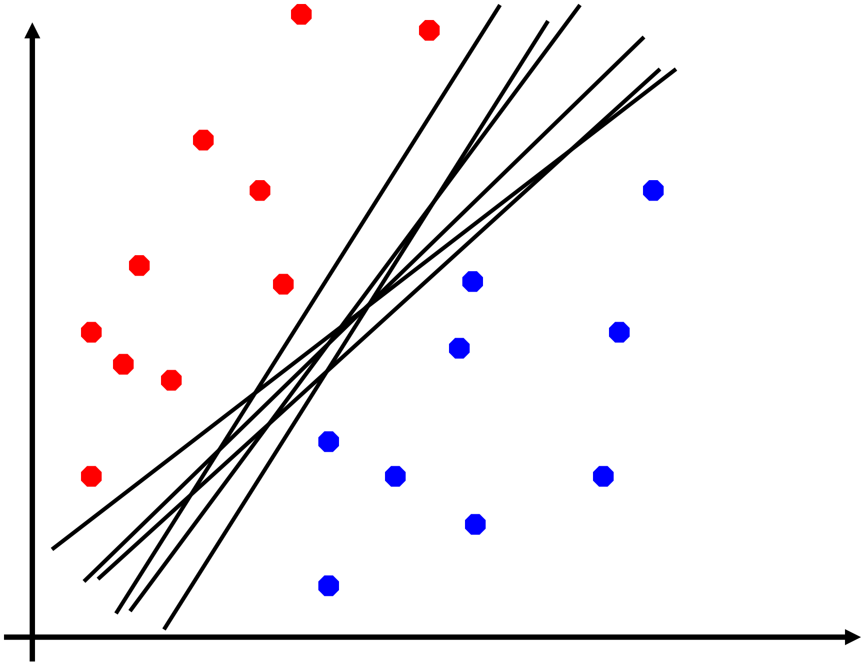
- Binary classification can be viewed as the task of separating classes in feature space:



$$f(\mathbf{x}) = \text{sign}(\mathbf{w}^T\mathbf{x} + b)$$

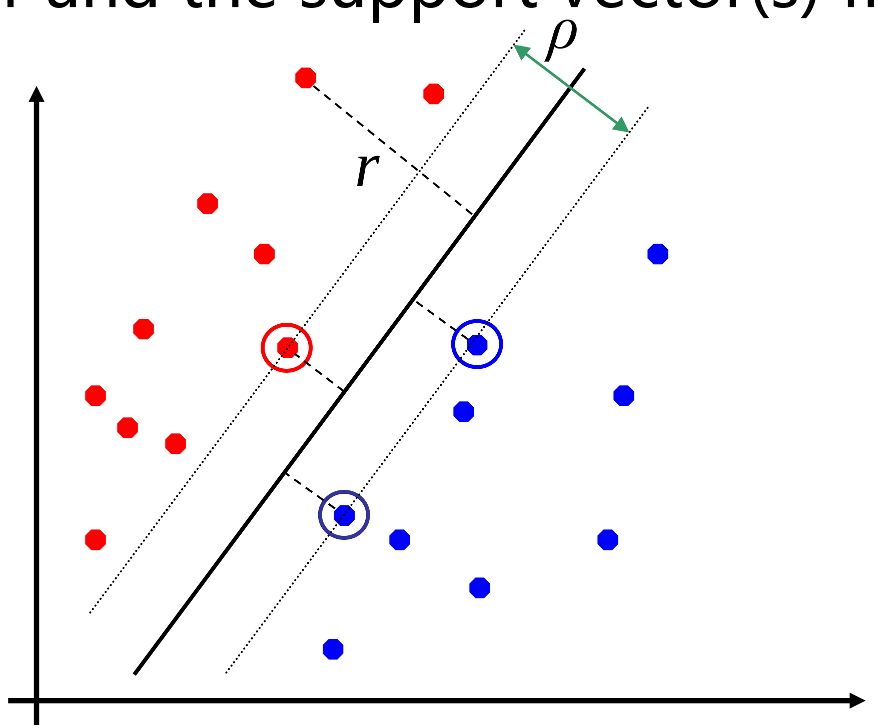
Linear Separators

- The problem in perceptron: which of the linear separators is optimal?



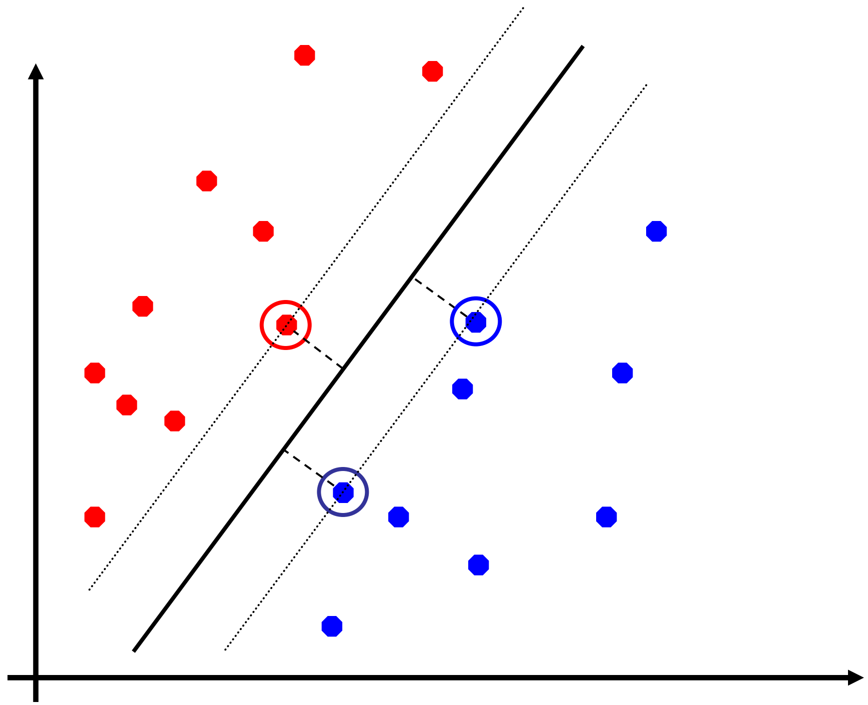
Classification Margin

- Distance from example $\mathbf{x}_i$ to the separator is $r = \frac{|\mathbf{w}^T\mathbf{x}_i + b|}{\|\mathbf{w}\|}$
- Examples closest to the hyperplane are ***support vectors***.
- *Margin* ρ of the separator is 2* the distance between the separator and the support vector(s) from either class.



Maximum Margin Classification

- Maximizing the margin is good according to intuition.
- Implies that only support vectors matter; other training examples are ignorable.



Linear SVM Mathematically

- Let training set $\{(\mathbf{x}_i, y_i)\}_{i=1..n}$, $\mathbf{x}_i \in \mathbf{R}^d$, $y_i \in \{-1, 1\}$ be separated by a hyperplane with margin ρ. Then for each training example $(\mathbf{x}_i, y_i)$:

$$\begin{array}{ll} (\mathbf{w}^T\mathbf{x}_i + b)/\|\mathbf{w}\| \leq -\rho/2 & \text{if } y_i = -1 \\ (\mathbf{w}^T\mathbf{x}_i + b)/\|\mathbf{w}\| \geq \rho/2 & \text{if } y_i = 1 \end{array} \quad \Leftrightarrow \quad y_i(\mathbf{w}^T\mathbf{x}_i + b)/\|\mathbf{w}\| \geq \rho/2$$

- For every support vector $\mathbf{x}_s$ the above inequality is an equality.
  - Scale **w** in the equality so that $\|w\|^* \rho/2=1$
- Then the margin can be expressed through (rescaled) **w** and b as:

$$\rho = \frac{2}{\|\mathbf{w}\|}$$

- And the constraints change to $y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1$

Linear SVMs Mathematically (cont.)

- Then we can formulate the *quadratic optimization problem:*

Find $\mathbf{w}$ and b such that

$$\rho = \frac{2}{\|\mathbf{w}\|} \text{ is maximized}$$

and for all $(\mathbf{x}_i, y_i)$, $i=1..n$: $\quad y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1$

Which can be reformulated as:

Find $\mathbf{w}$ and b such that

$$\mathbf{\Phi}(\mathbf{w}) = \tfrac{1}{2}*\|\mathbf{w}\|^2 = \tfrac{1}{2}*\mathbf{w}^T\mathbf{w} \text{ is minimized}$$

and for all $(\mathbf{x}_i, y_i)$, $i=1..n$: $\quad y_i (\mathbf{w}^T\mathbf{x}_i + b) \geq 1$

Solving the Optimization Problem

Find **w** and b such that
$\mathbf{\Phi}(\mathbf{w}) = ½*\mathbf{w}^T\mathbf{w}$ is minimized
and for all $(\mathbf{x}_i, y_i)$, i=1..n : $y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1$

- Need to optimize a *quadratic* function subject to *linear* constraints.
- Quadratic optimization problems are a well-known class of mathematical programming problems for which several (non-trivial) algorithms exist.
- The solution involves constructing a *dual problem* where a *Lagrange multiplier* α_i is associated with every inequality constraint in the primal (original) problem:

Find $\alpha_1 \ldots \alpha_n$ such that
$\mathbf{Q}(\boldsymbol{\alpha}) = \Sigma\alpha_i - ½\Sigma\Sigma\alpha_i\alpha_j y_i y_j \mathbf{x}_i^T\mathbf{x}_j$ is maximized and
(1) $\Sigma\alpha_i y_i = 0$
(2) $\alpha_i \geq 0$ for all α_i

The Optimization Problem Solution

- Given a solution $\alpha_1 \dots \alpha_n$ to the dual problem, solution to the primal is:

$$\boxed{\mathbf{w} = \Sigma \alpha_i y_i \mathbf{x}_i \qquad b = y_k - \Sigma \alpha_i y_i \mathbf{x}_i^{\mathbf{T}} \mathbf{x}_k \quad \text{for any } \alpha_k > 0}$$

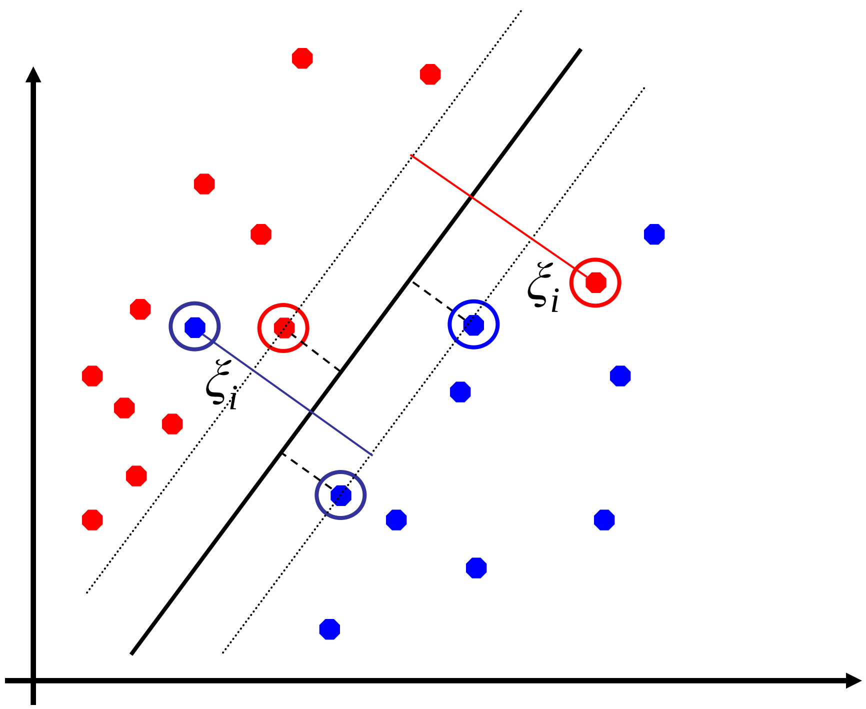
- Each non-zero α_i indicates that corresponding $\mathbf{x}_i$ is a support vector.
- Then the classifying function is (note that we don't need **w** explicitly):

$$\boxed{f(\mathbf{x}) = \Sigma \alpha_i y_i \mathbf{x}_i^{\mathbf{T}} \mathbf{x} + b}$$

- Notice that it relies on an *inner product* between the test point **x** and the support vectors $\mathbf{x}_i$ – we will return to this later.
- Also keep in mind that solving the optimization problem involved computing the inner products $\mathbf{x}_i^{\mathbf{T}}\mathbf{x}_j$ between all training points.

Soft Margin Classification

- What if the training set is not linearly separable?
- *Slack variables* ξ_i can be added to allow misclassification of difficult or noisy examples, resulting margin called *soft*.



Soft Margin Classification Mathematically

- The old formulation:

Find $\mathbf{w}$ and b such that
$\boldsymbol{\Phi}(\mathbf{w}) = \mathbf{w}^{\mathrm{T}}\mathbf{w}$ is minimized
and for all $(\mathbf{x}_i, y_i)$, $i=1..n$: $\quad y_i (\mathbf{w}^{\mathbf{T}}\mathbf{x}_i + b) \geq 1$

- Modified formulation incorporates slack variables:

Find $\mathbf{w}$ and b such that
$\boldsymbol{\Phi}(\mathbf{w}) = \mathbf{w}^{\mathrm{T}}\mathbf{w} + C\Sigma\xi_i$ is minimized
and for all $(\mathbf{x}_i, y_i)$, $i=1..n$: $\quad y_i (\mathbf{w}^{\mathbf{T}}\mathbf{x}_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0$

  - Parameter C can be viewed as a way to control overfitting: it "trades off" the relative importance of maximizing the margin and fitting the training data.

Soft Margin Classification – Solution

- Dual problem is identical to separable case (would *not* be identical if the 2-norm penalty for slack variables $C\Sigma\xi_i^2$ was used in primal objective, we would need additional Lagrange multipliers for slack variables):

> Find $\alpha_1 \ldots \alpha_N$ such that
> $\mathbf{Q}(\boldsymbol{\alpha}) = \Sigma\alpha_i - \frac{1}{2}\Sigma\Sigma\alpha_i\alpha_j y_i y_j \mathbf{x}_i^T\mathbf{x}_j$ is maximized and
> (1) $\Sigma\alpha_i y_i = 0$
> (2) $0 \leq \alpha_i \leq C$ for all α_i

- Again, $\mathbf{x}_i$ with non-zero α_i will be support vectors.
- Solution to the dual problem is:

> $\mathbf{w} = \Sigma\alpha_i y_i \mathbf{x}_i$
> $b = y_k(1 - \xi_k) - \Sigma\alpha_i y_i \mathbf{x}_i^T\mathbf{x}_k$ for any k s.t. $\alpha_k > 0$

Again, we don't need to compute **w** explicitly for classification:

> $f(\mathbf{x}) = \Sigma\alpha_i y_i \mathbf{x}_i^T\mathbf{x} + b$

Linear SVMs: Overview

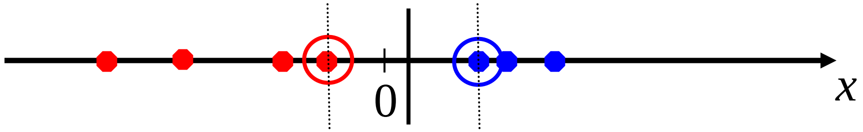
- The classifier is a *separating hyperplane.*
- Most "important" training points are support vectors; they define the hyperplane.
- Quadratic optimization algorithms can identify which training points $\mathbf{x}_i$ are support vectors with non-zero Lagrangian multipliers α_i.
- Both in the dual formulation of the problem and in the solution training points appear only inside inner products:

Find $\alpha_1 \ldots \alpha_N$ such that
$\mathbf{Q}(\boldsymbol{\alpha}) = \Sigma\alpha_i - \frac{1}{2}\Sigma\Sigma\alpha_i\alpha_j y_i y_j \boxed{\mathbf{x}_i^{\mathrm{T}}\mathbf{x}_j}$ is maximized and
(1) $\Sigma\alpha_i y_i = 0$
(2) $0 \leq \alpha_i \leq C$ for all α_i

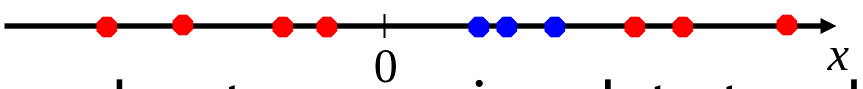
$$f(\mathbf{x}) = \Sigma\alpha_i y_i \boxed{\mathbf{x}_i^{\mathrm{T}}\mathbf{x}} + b$$

Non-linear SVMs

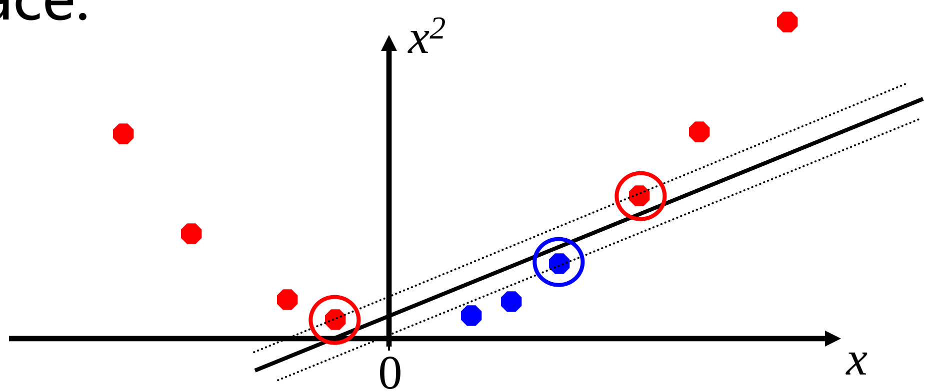
- Datasets that are linearly separable with some noise work out great:



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

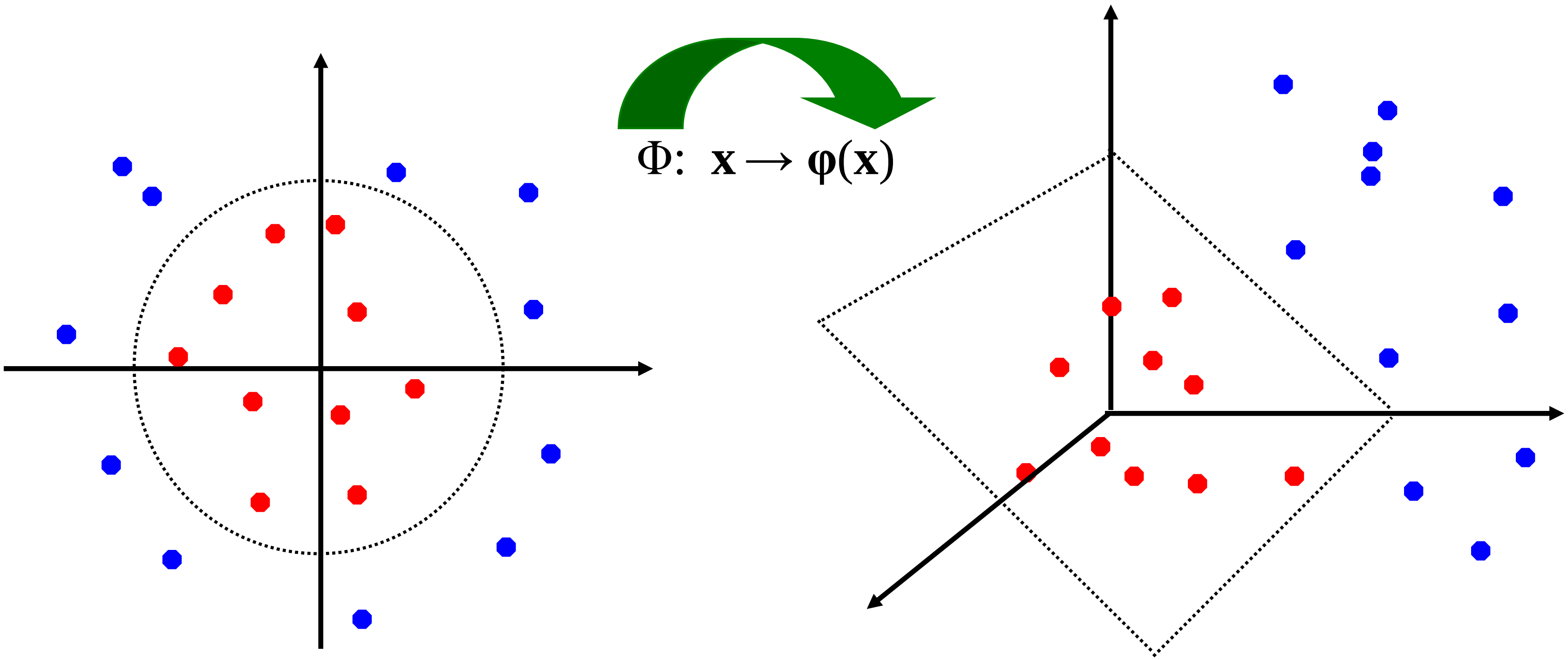


- How about… mapping data to a higher-dimensional space:



Non-linear SVMs: Feature spaces

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



The "Kernel Trick"

- The linear classifier relies on inner product between vectors $K(\mathbf{x}_i,\mathbf{x}_j)=\mathbf{x}_i^\mathsf{T}\mathbf{x}_j$
- If every datapoint is mapped into high-dimensional space via some transformation Φ: $\mathbf{x} \rightarrow \varphi(\mathbf{x})$, the inner product becomes:

$$K(\mathbf{x}_i,\mathbf{x}_j)= \boldsymbol{\varphi}(\mathbf{x}_i)^\mathsf{T}\boldsymbol{\varphi}(\mathbf{x}_j)$$

- A *kernel function* is a function that is eqiuvalent to an inner product in some feature space.
- Example:

  2-dimensional vectors $\mathbf{x}=[x_1\ \ x_2]$; let $K(\mathbf{x}_i,\mathbf{x}_j)=(1 + \mathbf{x}_i^\mathsf{T}\mathbf{x}_j)^2$,

  Need to show that $K(\mathbf{x}_i,\mathbf{x}_j)= \boldsymbol{\varphi}(\mathbf{x}_i)^\mathsf{T}\boldsymbol{\varphi}(\mathbf{x}_j)$:

  $K(\mathbf{x}_i,\mathbf{x}_j)=(1 + \mathbf{x}_i^\mathsf{T}\mathbf{x}_j)^2 = 1+ x_{i1}^2x_{j1}^2 + 2\, x_{i1}x_{j1}\, x_{i2}x_{j2}+ x_{i2}^2x_{j2}^2 + 2x_{i1}x_{j1} + 2x_{i2}x_{j2}=$

  $= [1\ \ x_{i1}^2\ \ \sqrt{2}\ x_{i1}x_{i2}\ \ x_{i2}^2\ \ \sqrt{2}x_{i1}\ \ \sqrt{2}x_{i2}]^\mathsf{T}\, [1\ \ x_{j1}^2\ \ \sqrt{2}\ x_{j1}x_{j2}\ \ x_{j2}^2\ \ \sqrt{2}x_{j1}\ \ \sqrt{2}x_{j2}]$

  $= \boldsymbol{\varphi}(\mathbf{x}_i)^\mathsf{T}\boldsymbol{\varphi}(\mathbf{x}_j)$, where $\boldsymbol{\varphi}(\mathbf{x}) = [1\ \ x_1^2\ \ \sqrt{2}\ x_1x_2\ \ x_2^2\ \ \sqrt{2}x_1\ \ \sqrt{2}x_2]$

- Thus, a kernel function *implicitly* maps data to a high-dimensional space (without the need to compute each $\boldsymbol{\varphi}(\mathbf{x})$ explicitly).

What Functions are Kernels?

- For some functions $K(\mathbf{x}_i,\mathbf{x}_j)$ checking that $K(\mathbf{x}_i,\mathbf{x}_j)= \varphi(\mathbf{x}_i)^T\varphi(\mathbf{x}_j)$ can be cumbersome.
- Mercer's theorem:

Every semi-positive definite symmetric function is a kernel

- Semi-positive definite symmetric functions correspond to a semi-positive definite symmetric Gram matrix:

K=

$K(\mathbf{x}_1,\mathbf{x}_1)$	$K(\mathbf{x}_1,\mathbf{x}_2)$	$K(\mathbf{x}_1,\mathbf{x}_3)$	…	$K(\mathbf{x}_1,\mathbf{x}_n)$
$K(\mathbf{x}_2,\mathbf{x}_1)$	$K(\mathbf{x}_2,\mathbf{x}_2)$	$K(\mathbf{x}_2,\mathbf{x}_3)$		$K(\mathbf{x}_2,\mathbf{x}_n)$
…	…	…	…	…
$K(\mathbf{x}_n,\mathbf{x}_1)$	$K(\mathbf{x}_n,\mathbf{x}_2)$	$K(\mathbf{x}_n,\mathbf{x}_3)$	…	$K(\mathbf{x}_n,\mathbf{x}_n)$

Examples of Kernel Functions

- Linear: $K(\mathbf{x}_i,\mathbf{x}_j)= \mathbf{x}_i^{\mathsf{T}}\mathbf{x}_j$
  - Mapping Φ: $\mathbf{x} \to \boldsymbol{\varphi}(\mathbf{x})$, where $\boldsymbol{\varphi}(\mathbf{x})$ is $\mathbf{x}$ itself

- Polynomial of power p: $K(\mathbf{x}_i,\mathbf{x}_j)= (1+ \mathbf{x}_i^{\mathsf{T}}\mathbf{x}_j)^p$
  - Mapping Φ: $\mathbf{x} \to \boldsymbol{\varphi}(\mathbf{x})$, where $\boldsymbol{\varphi}(\mathbf{x})$ has $\binom{d+p}{p}$ dimensions

- Gaussian (radial-basis function): $K(\mathbf{x}_i,\mathbf{x}_j) = e^{-\frac{\|\mathbf{x}_i-\mathbf{x}_j\|^2}{2\sigma^2}}$
  - Mapping Φ: $\mathbf{x} \to \boldsymbol{\varphi}(\mathbf{x})$, where $\boldsymbol{\varphi}(\mathbf{x})$ is *infinite-dimensional*: every point is mapped to *a function* (a Gaussian); combination of functions for support vectors is the separator.

- Higher-dimensional space still has *intrinsic* dimensionality d, but linear separators in it correspond to *non-linear* separators in original space.

Non-linear SVMs Mathematically

- Dual problem formulation:

Find $\alpha_1 \ldots \alpha_n$ such that
$\mathbf{Q}(\boldsymbol{\alpha}) = \Sigma\alpha_i - \frac{1}{2}\Sigma\Sigma\alpha_i\alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$ is maximized and
(1) $\Sigma\alpha_i y_i = 0$
(2) $\alpha_i \geq 0$ for all α_i

- The solution is:

$$f(\mathbf{x}) = \Sigma\alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b$$

- Optimization techniques for finding α_i′ s remain the same!

SVM applications

- SVMs were originally proposed by Boser, Guyon and Vapnik in 1992 and gained increasing popularity in late 1990s.
- SVMs are currently among the best performers for a number of classification tasks ranging from text to genomic data.
- SVMs can be applied to complex data types beyond feature vectors (e.g. graphs, sequences, relational data) by designing kernel functions for such data.
- SVM techniques have been extended to a number of tasks such as regression [Vapnik *et al.* '97], principal component analysis [Schölkopf *et al.* '99], etc.
- Most popular optimization algorithms for SVMs use *decomposition* to hill-climb over a subset of α_i's at a time, e.g. SMO [Platt '99] and [Joachims '99]
- Tuning SVMs remains a black art: selecting a specific kernel and parameters is usually done in a try-and-see manner.

R Packages

- Linear models with regularization (feature selection): glmnet
  - https://cran.r-project.org/web/packages/glmnet/index.html
- SVM: e1071
  - https://cran.r-project.org/web/packages/e1071/index.html

Trees, Neural Networks and other classifiers

David Li

Other Representative Classifiers

- Tree-based classifiers
  - Decision Trees
  - Random Forest, Boosting
- KNN (K-nearest neighbors)
- Neural Network

Other Representative Classifiers

- Tree-based classifiers
  - Decision Trees
  - Random Forest, Boosting
- KNN (K-nearest neighbors)
- Neural Network

When to play tennis?

- 4 discrete-valued attributes
- "Yes/No" classification

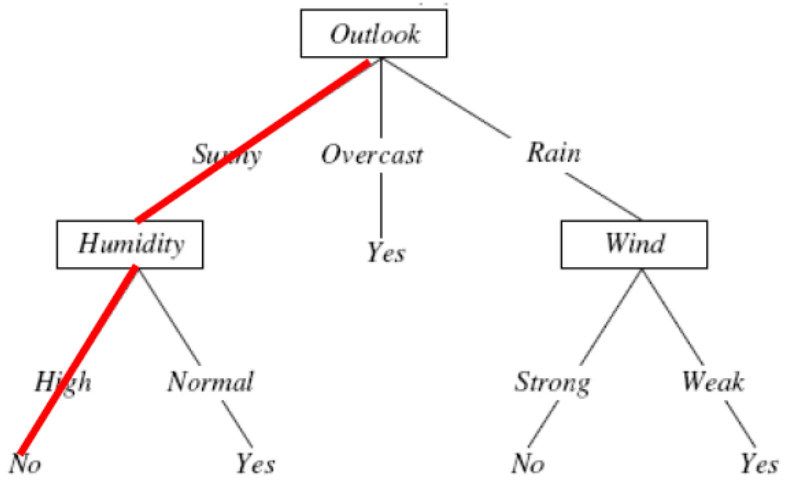
Day	Outlook	Temp.	Humidity	Wind	P.Tennis
d_1	Sunny	Hot	High	Weak	No
d_2	Sunny	Hot	High	Strong	No
d_3	Overcast	Hot	High	Weak	Yes
d_4	Rain	Mild	High	Weak	Yes
d_5	Rain	Cool	Normal	Weak	Yes
d_6	Rain	Cool	Normal	Strong	No
d_7	Overcast	Cool	Normal	Strong	Yes
d_8	Sunny	Mild	High	Weak	No
d_9	Sunny	Cool	Normal	Weak	Yes
d_{10}	Rain	Mild	Normal	Weak	Yes
d_{11}	Sunny	Mild	Normal	Strong	Yes
d_{12}	Overcast	Mild	High	Strong	Yes
d_{13}	Overcast	Hot	Normal	Weak	Yes
d_{14}	Rain	Mild	High	Strong	No

Decision Tree

- Internal nodes labeled with some feature x_j
- Arc (from x_j) labeled with results of test x_j
- Leaf nodes specify class $h(x)$
- Instance:

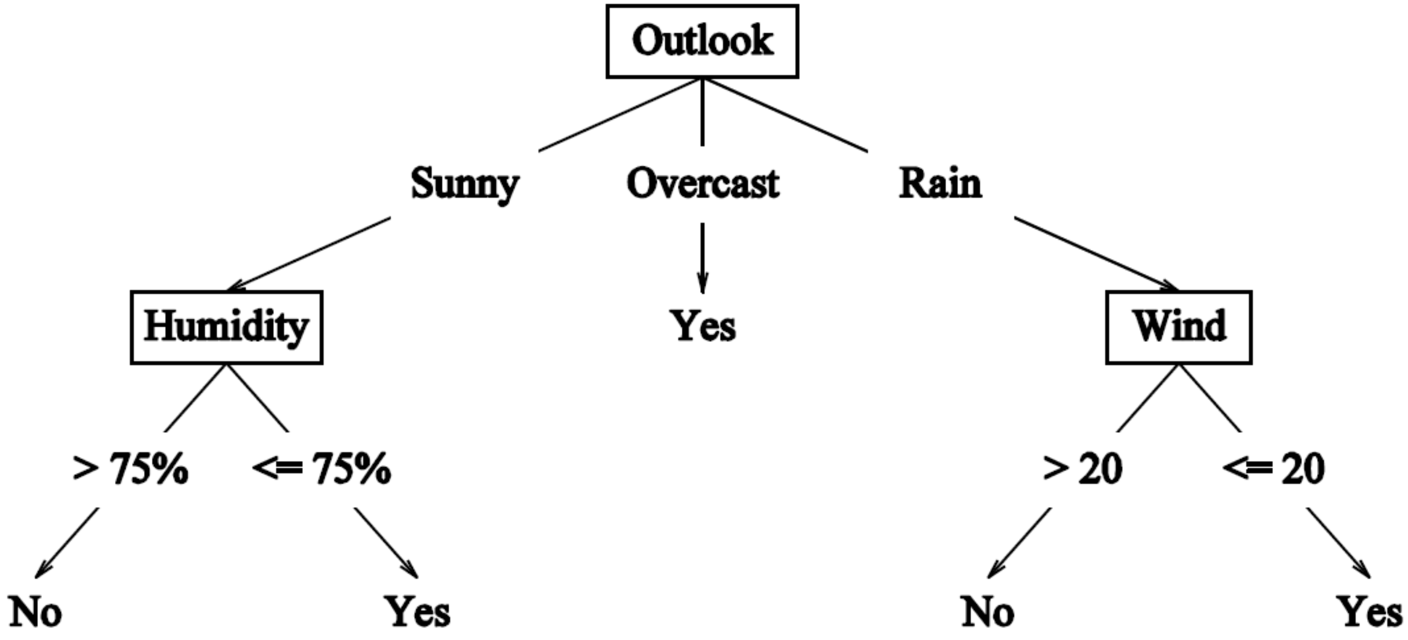
Outlook	= Sunny
Temperature	= Hot
Humidity	= High
Wind	= Strong

  classified as "No"
  - (Temperature, Wind: irrelevant)
- Easy to use in Classification
  - Answer short series of questions…



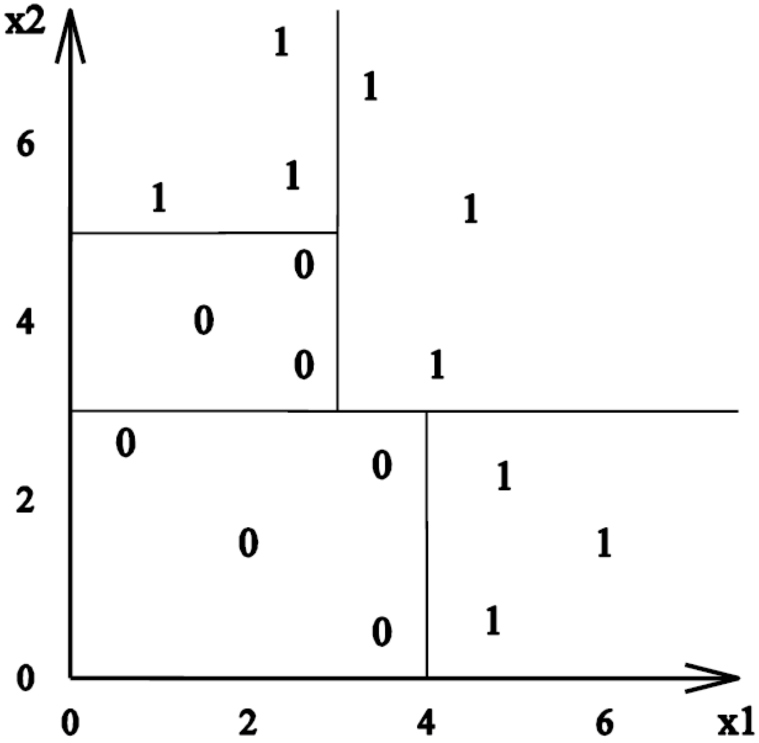
Continuous Features

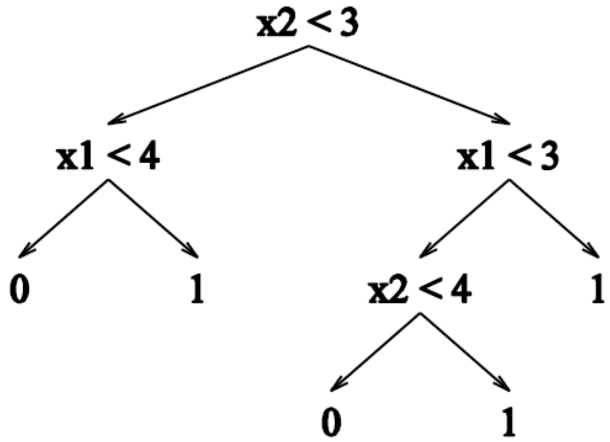
- If feature is continuous, internal nodes may test value against threshold



Decision Tree Decision Boundaries

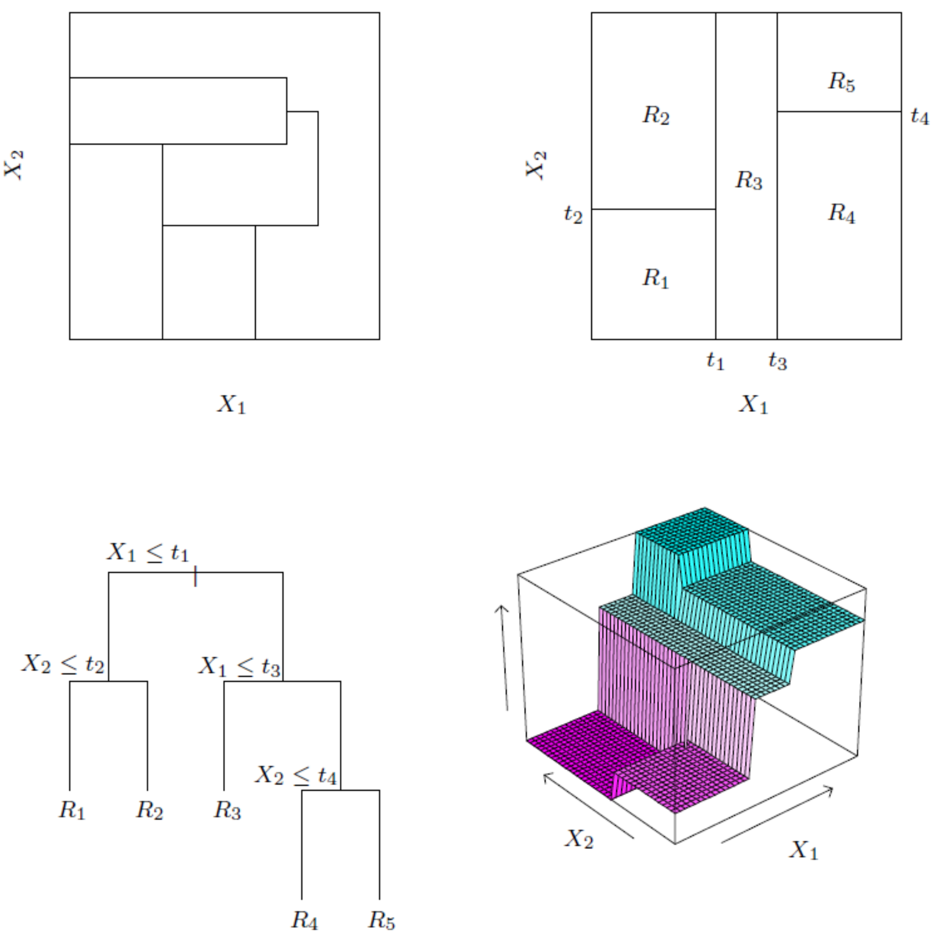
- Decision Trees divide the feature space into axis-parallel rectangles and label each rectangle with one of the K classes





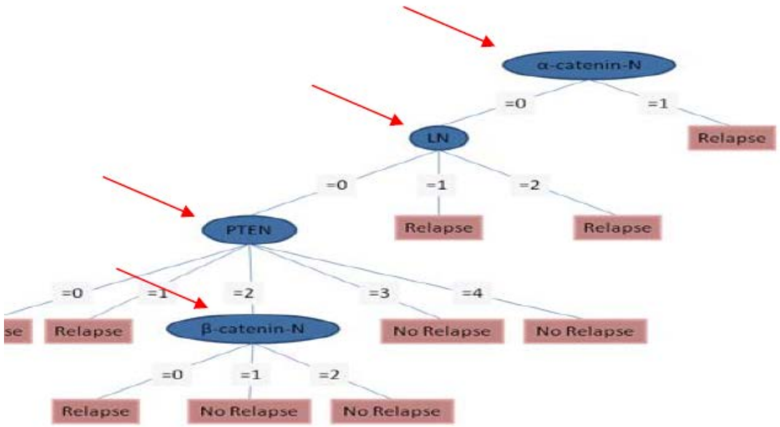
Space Partition

- **Top right:** a partition of a two-dimensional feature space by recursive binary splitting applied to some fake data.
- **Top left:** a general partition that *cannot* be obtained from recursive binary splitting.
- **Bottom left:** the tree corresponding to the partition in the top right panel
- **Bottom right:** a perspective plot of the prediction surface.

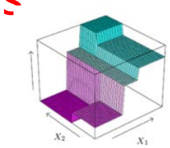


Remarks of Trees

- Well known Tree implementations: CART (Classification And Regression Tree), ID3 → C4.5 → C5.0
- Good interpretability
  - Concentration of
    α-catenin in nucleus is very important:
    - If >0, probably relapse
  - If =0, then #lymph_nodes is important:
    - If >0, probably relaps
  - If =0, then concentration of pten is important:
    - If <2, probably relapse
    - If >2, probably NO relapse
  - If =2, then concentration of β-catenin in nucleus is important:
    - If =0, probably relapse
    - If >0, probably NO relapse



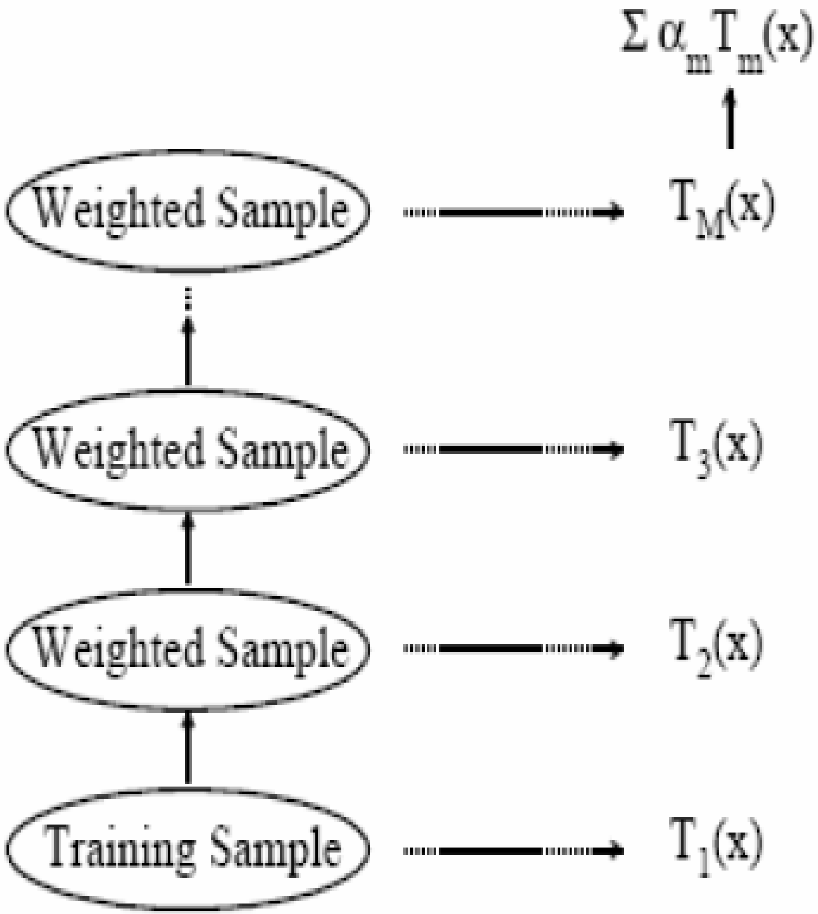
- Instability
  – A small change in the data can result a very different tree, due to the hierarchical nature of the tree building procedure.
- Lack of smoothness



- Difficulty in capturing additive structure: X1 + X2  vs X1*X2 

Boosting the Decision Tree

- Decision trees have been known for some time, but often are unstable; a small change in the training sample can produce a large difference
- Boosting the decision tree
  - Give the training events misclassified under this procedure a higher weight.
  - Continuing build perhaps 1000 trees and average the results.



Radom Forest

- Random forests (Breiman, 2001) is a substantial modification of bagging that builds a large collection of **de-correlated** trees, and then averages them.
- The authors make grand claims about the success of random forests: “most accurate”, “ most interpretable” and the like.
- On many problems the performance of random forests is very similar to boosting, and they are simpler to train and tune. As a consequence, random forests are popular, and are implemented in a variety of packages
  – R package: randomForest
  – http://www.math.usu.edu/~adele/forests/

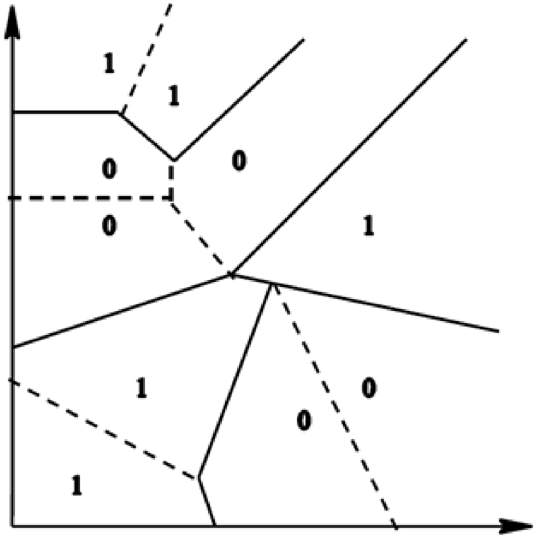
Other Representative Classifiers

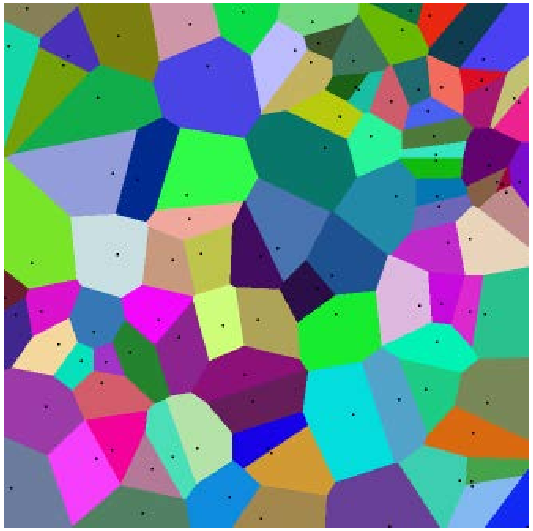
- Tree-based classifiers
  - Decision Trees
  - Random Forest, Boosting
- KNN (K-nearest neighbors)
- Neural Network

Basic k-nearest neighbor (KNN) classification

- Training method:
  - Save the training examples
- At prediction time:
  - Determine parameter *k*
  - Find the *k* training examples $(x_1,y_1),\ldots(x_k,y_k)$ that are closest to the test example *x*
  - Predict the most frequent class among those y_i' s.
- Properties:
  - A "lazy" classifier. No training.
  - Feature selection and distance measure are crucial.

Decision Boundaries of 1-NN: the Voronoi Diagram

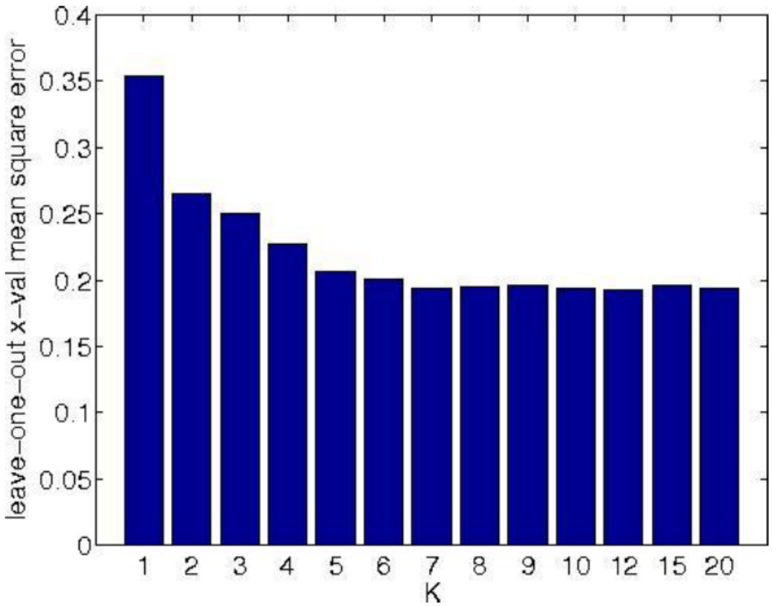


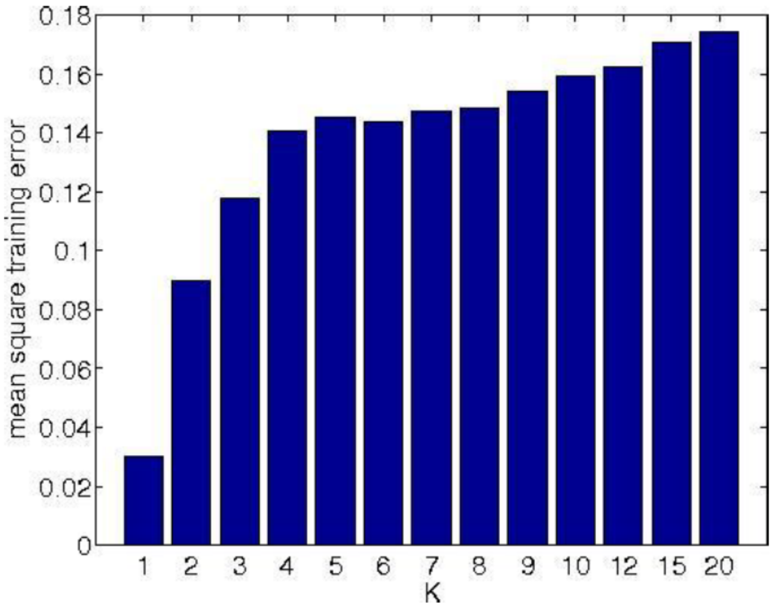


- Nearest Neighbor does not explicitly compute decision boundaries. However, the boundaries form a subset of the Voronoi diagram of the training data
- Each line segment is equidistant between two points of opposite class. The more examples that are stored, the more complex the decision boundaries can become.

Picking K

- Value of *k* is typically odd to avoid ties
- Use V-fold cross validation: pick the one that minimizes cross validation error.





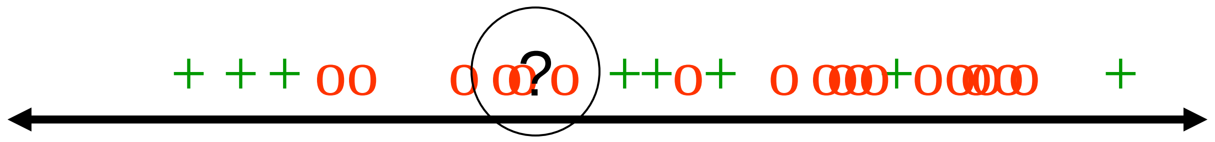
Pick best value according to the error on the validation set

Normalizing attribute values

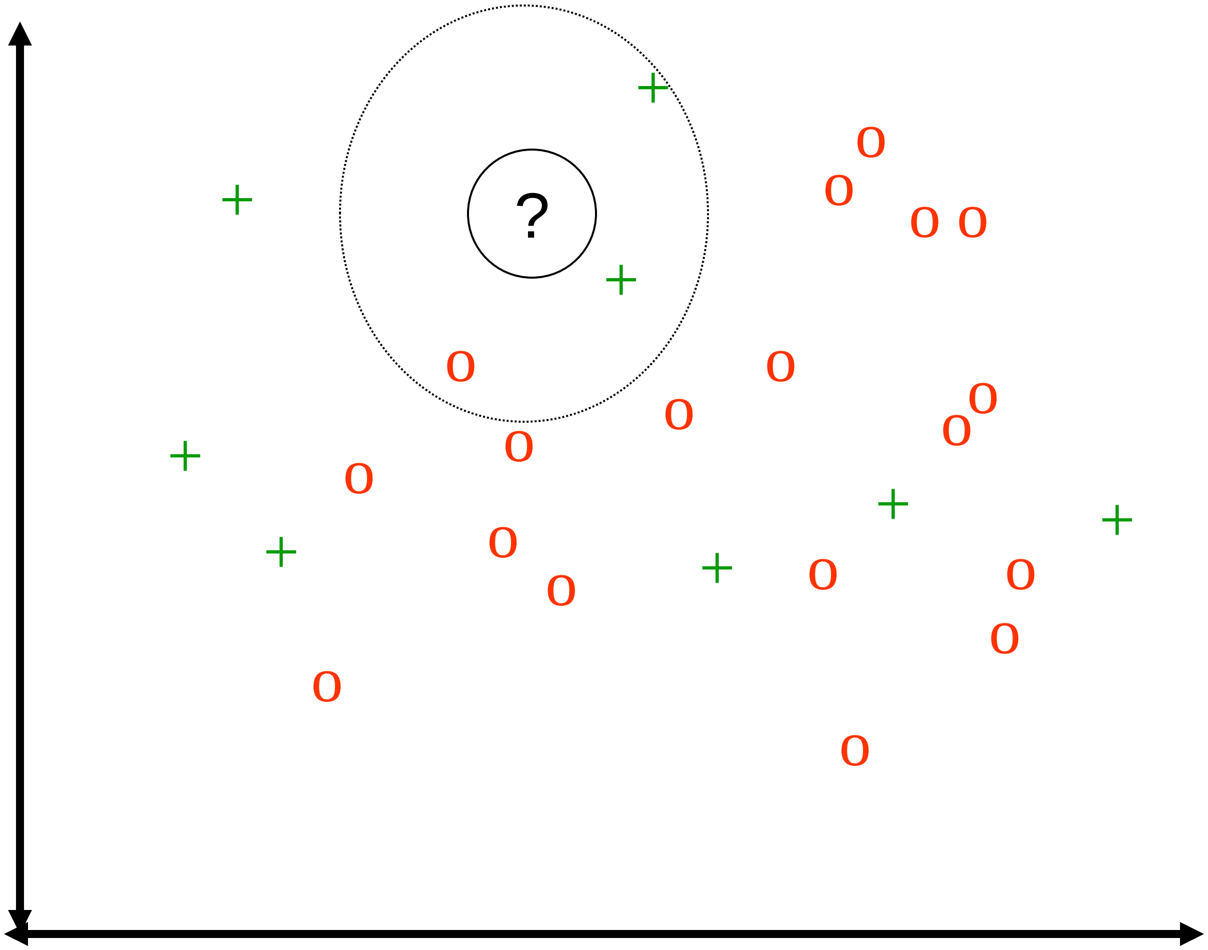
- Distance could be dominated by some attributes with large numbers:
  - Ex: features: age, income
  - Original data: x_1=(35, 76K), x_2=(36, 80K), x_3=(70, 79K)
  - Assume: age in [0,100], income in [0, 200K]
  - After normalization:

    x_1=(0.35, 0.38), x_2=(0.36, 0.40), x_3 = (0.70, 0.395).

K-NN and irrelevant features



K-NN and irrelevant features



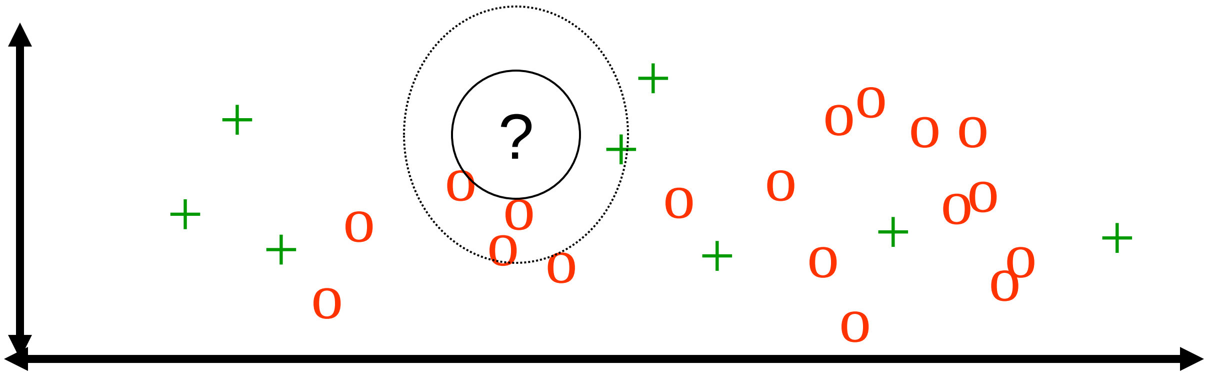
The Choice of Features

- Imagine there are 100 features, and only 2 of them are relevant to the target label.
- kNN is easily misled in high-dimensional space.

➔ Feature weighting or feature selection

Feature weighting

- Stretch j-th axis by weight w_j,
- Use cross-validation to automatically choose weights $w_1, \ldots, w_n$
- Setting w_j to zero eliminates this dimension altogether.



Similarity measure

- Euclidean distance:

$$dist(d_i, d_j) = \sqrt{\sum_k (a_{i,k} - a_{j,k})^2}$$

- Weighted Euclidean distance:

$$dist(d_i, d_j) = \sqrt{\sum_k w_k (a_{i,k} - a_{j,k})^2}$$

- Similarity measure: cosine

$$cos(d_i, d_j) = \frac{\sum_k a_{i,k} a_{j,k}}{\sqrt{\sum_k a_{i,k}^2} \sqrt{\sum_k a_{j,k}^2}}$$

Weighted kNN

- Inputs: Training data $\{(\mathbf{x}_i, y_i)\}_{i=1}^{m}$ distance metric d on $\mathcal{X}$, weighting function

$$w : \Re \mapsto \Re$$

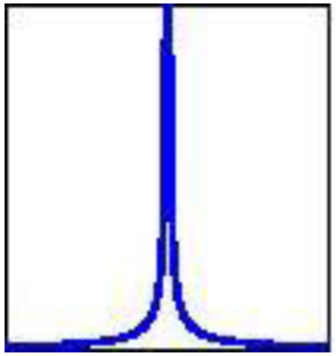
- Learning: Nothing to do!
- Prediction: On input **x**,
  - For each i compute $w_i = w(d(\mathbf{x}_i, \mathbf{x}))$.
  - Predict weighted majority or mean. For example,

$$\mathbf{y} = \frac{\sum_i w_i \mathbf{y_i}}{\sum_i w_i}$$

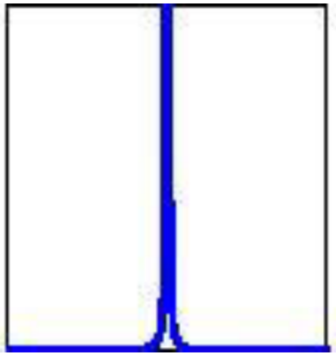
How to weight distances?

Some weighting functions

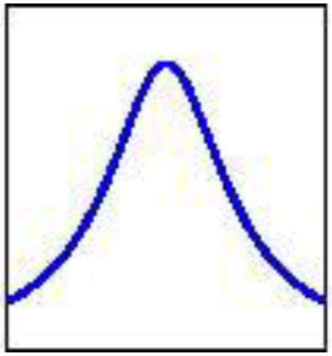
$$\frac{1}{d(\mathbf{x}_i,\mathbf{x})}$$



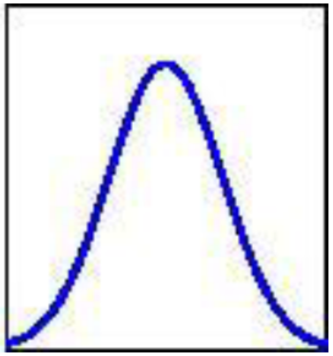
$$\frac{1}{d(\mathbf{x}_i,\mathbf{x})^2}$$



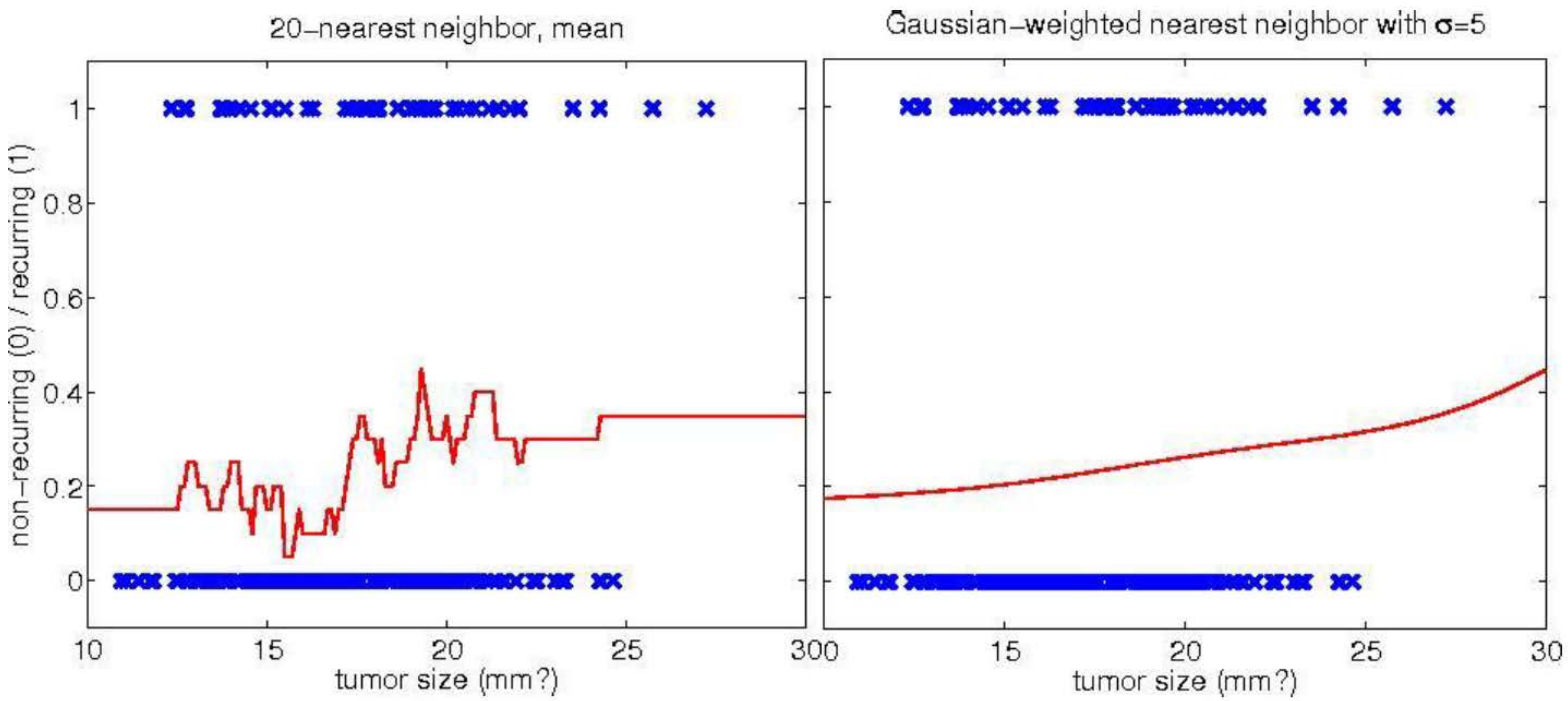
$$\frac{1}{c+d(\mathbf{x}_i,\mathbf{x})^2}$$



$$\exp\left(-\frac{d(\mathbf{x}_i,\mathbf{x})^2}{\sigma^2}\right)$$

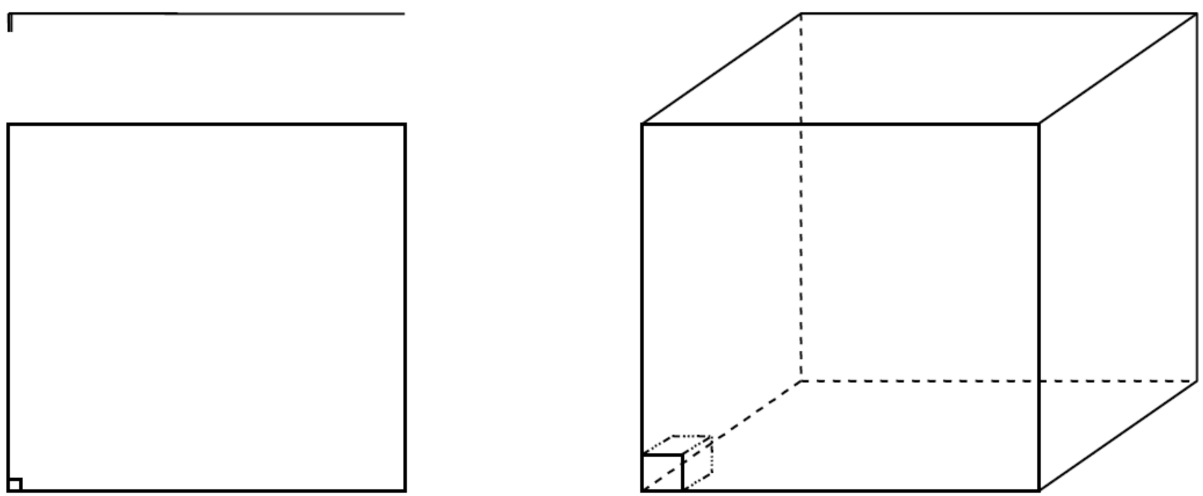


Example: Gaussian weighting



The Curse of Dimensionality

- Nearest neighbor breaks down in high-dimensional spaces, because the "neighborhood" becomes very large.
- Suppose we have 5000 points uniformly distributed in the unit hypercube and we want to apply the 5-nearest neighbor algorithm. Suppose our query point is at the origin.
- Then on the 1-dimensional line, we must go a distance of 5/5000 = 0.001 on the average to capture the 5 nearest neighbors
- In 2 dimensions, we must go $\sqrt{0.001} = 0.0316$ to get a square that contains 0.001 of the volume.
- In D dimensions, we must go $(0.001)^{1/d}$ [$(0.001)^{1/30} = 0.794$!]



Summary of kNN

- Strengths:
  - Simplicity (conceptual)
  - Efficiency at training: no training
  - Handling multi-class
  - Very flexible decision boundaries
- Weakness:
  - Theoretical validity
  - It is not clear which types of distance measure
  - Irrelevant features must be eliminated
  - typically cannot handle more than 30 features
  - computational costs: memory to store all training samples; and testing-time: need to calculate all distances

Other Representative Classifiers

- Tree-based classifiers
  - Decision Trees
  - Random Forest, Boosting
- KNN (K-nearest neighbors)
- Neural Network

Artificial Neural Network

Information flow is unidirectional

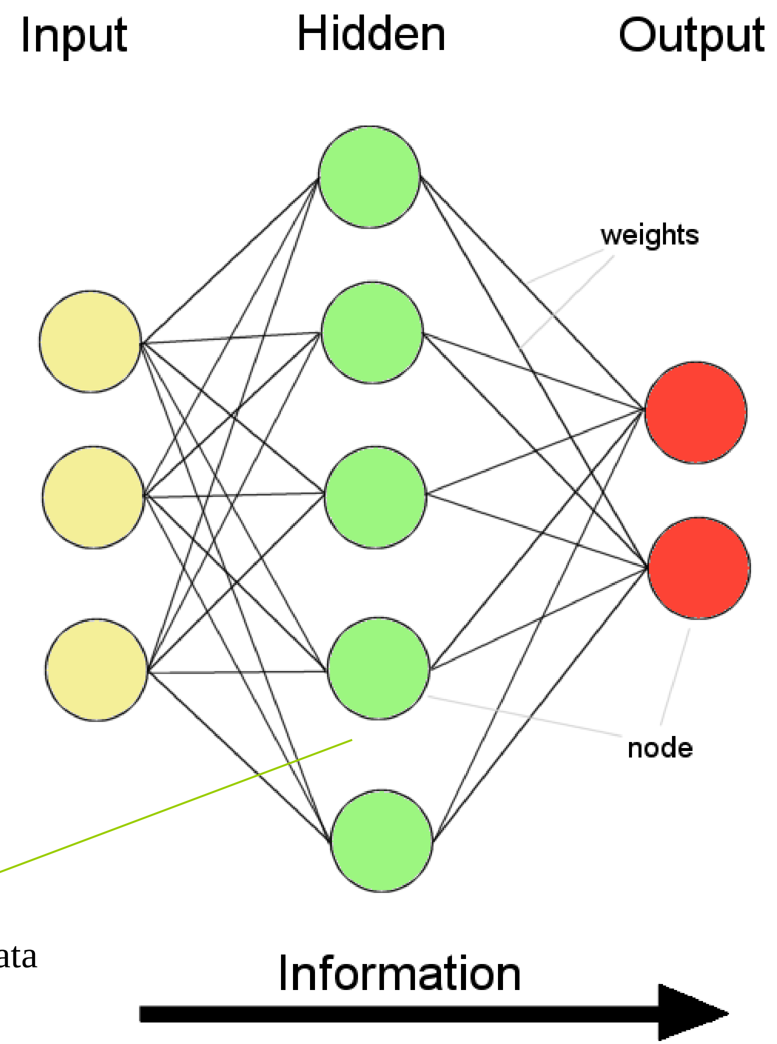
Data is presented to *Input layer*

Passed on to *Hidden Layer*

Passed on to *Output layer*

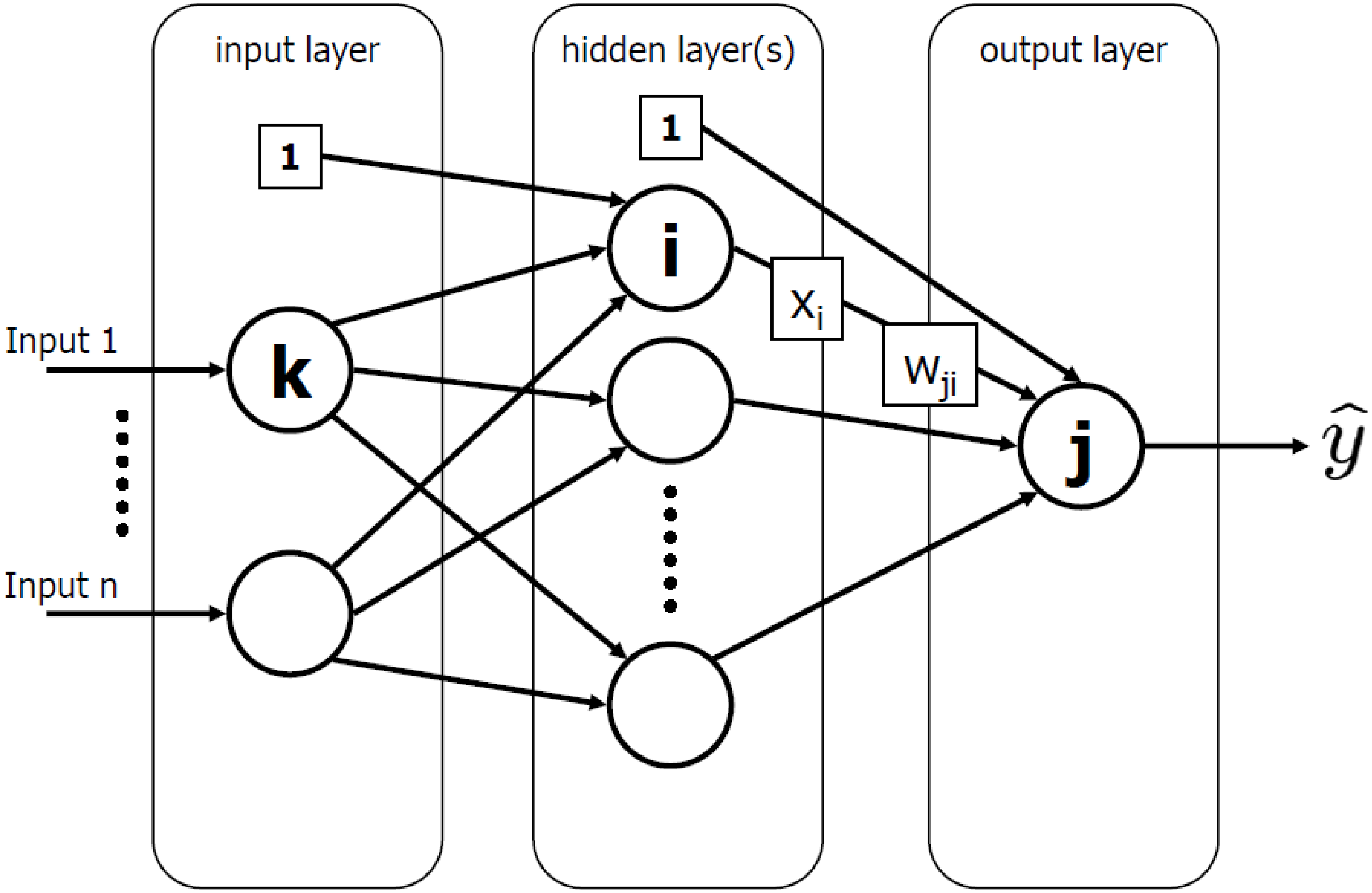
Information is distributed

Information processing is parallel

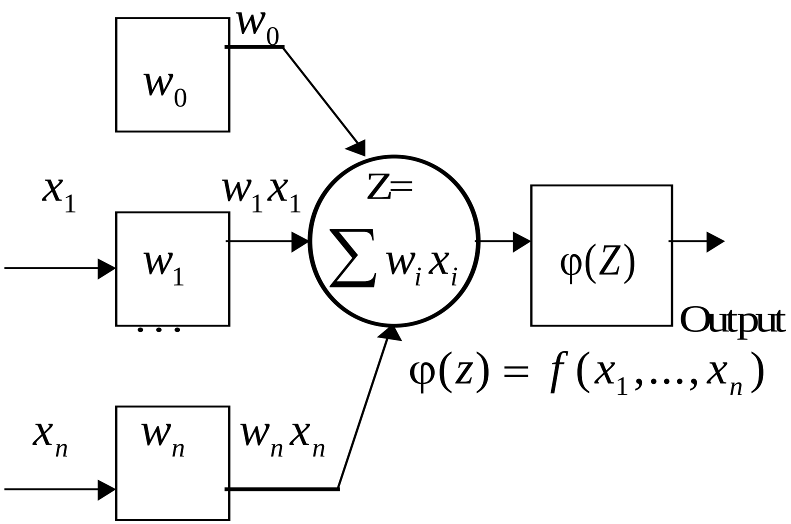


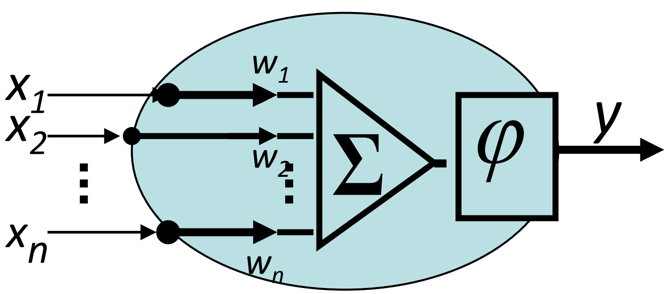
Internal representation (interpretation) of data

Feed-forward (neural) network



Artificial Neuron





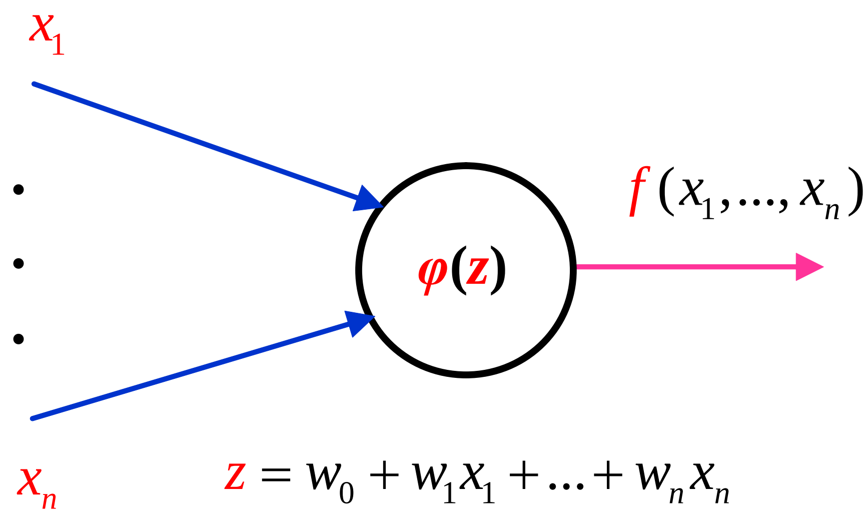
- A neuron has a set of *n synapses* associated to the *inputs*. Each of them is characterized by a weight .
- A signal $x_i, \; i = 1, \dots, n$ at the i^{th} input is multiplied (weighted) by the weight $w_i, \; i = 1, \dots, n$
- The weighted input signals are summed. Thus, a linear combination of the input signals $w_1 x_1 + \dots + w_n x_n$ is obtained. A "free weight" (or bias) w_0 , which does not correspond to any input, is added to this linear combination and this forms a *weighted sum* $z = w_0 + w_1 x_1 + \dots + w_n x_n$
- A (nonlinear) **activation function** $\boldsymbol{\varphi}$ is applied to the weighted sum. A value of the activation function $y = \phi(z)$ is the neuron's output.

A Neuron

- Neurons' functionality is determined by the nature of its activation function, that will specify its main properties, its plasticity and flexibility, its ability to approximate a function to be learned

$x_1, \ldots, x_n$ are the inputs, f is a function to be learned

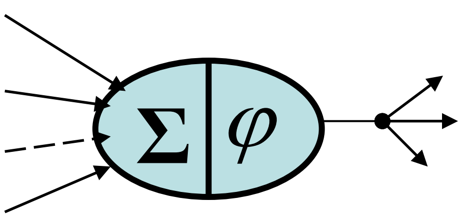
φ is the activation function



$z = w_0 + w_1 x_1 + \ldots + w_n x_n$

Z is the weighted sum

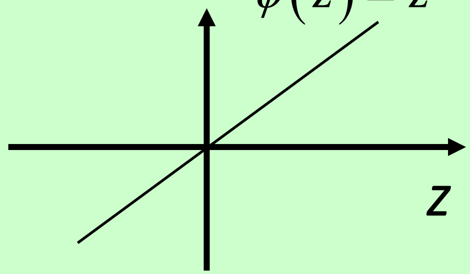
Artificial Neuron: Classical Activation Functions



One simple perctron

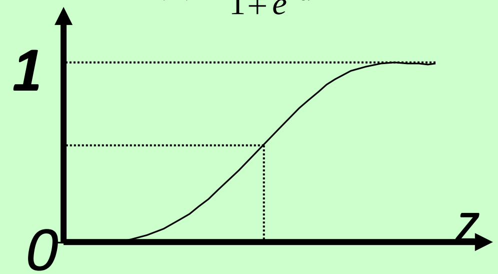
Linear activation

$$\phi(z) = z$$



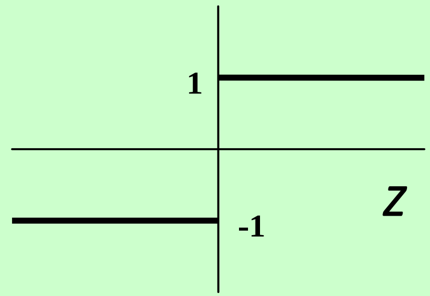
Logistic activation

$$\phi(z) = \frac{1}{1+e^{-\alpha z}}$$



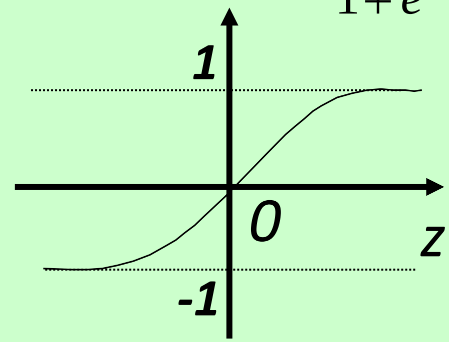
Threshold activation

$$\phi(z) = \text{sign}(z) = \begin{cases} 1, & if \quad z \geq 0, \\ -1, & if \quad z < 0. \end{cases}$$

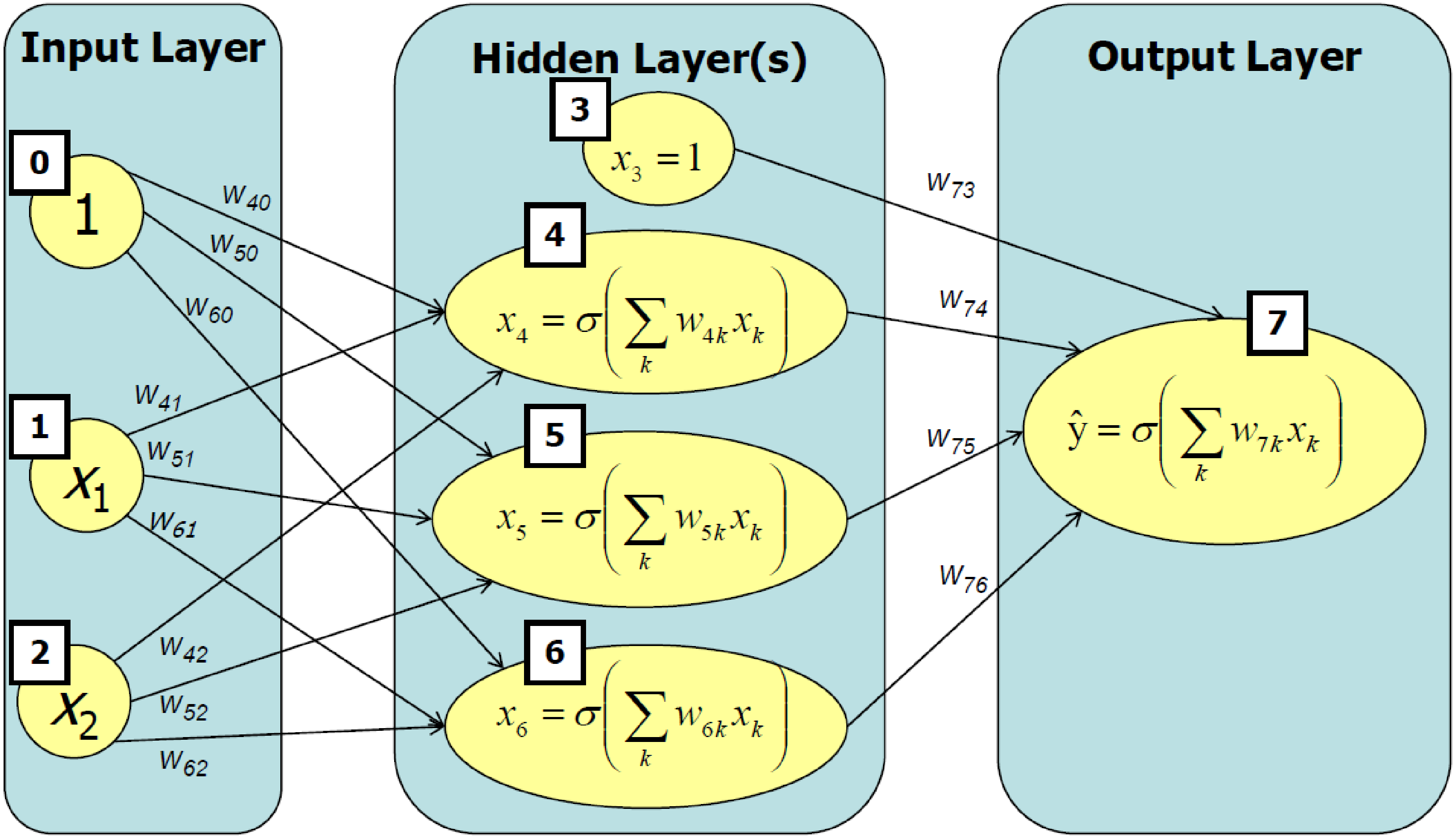


Hyperbolic tangent activation

$$\varphi(u) = tanh(\gamma u) = \frac{1-e^{-2\gamma u}}{1+e^{-2\gamma u}}$$

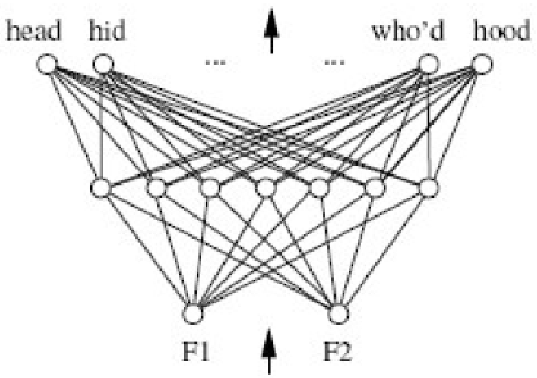


A 1-HIDDEN LAYER NET



When to Consider Neural Networks

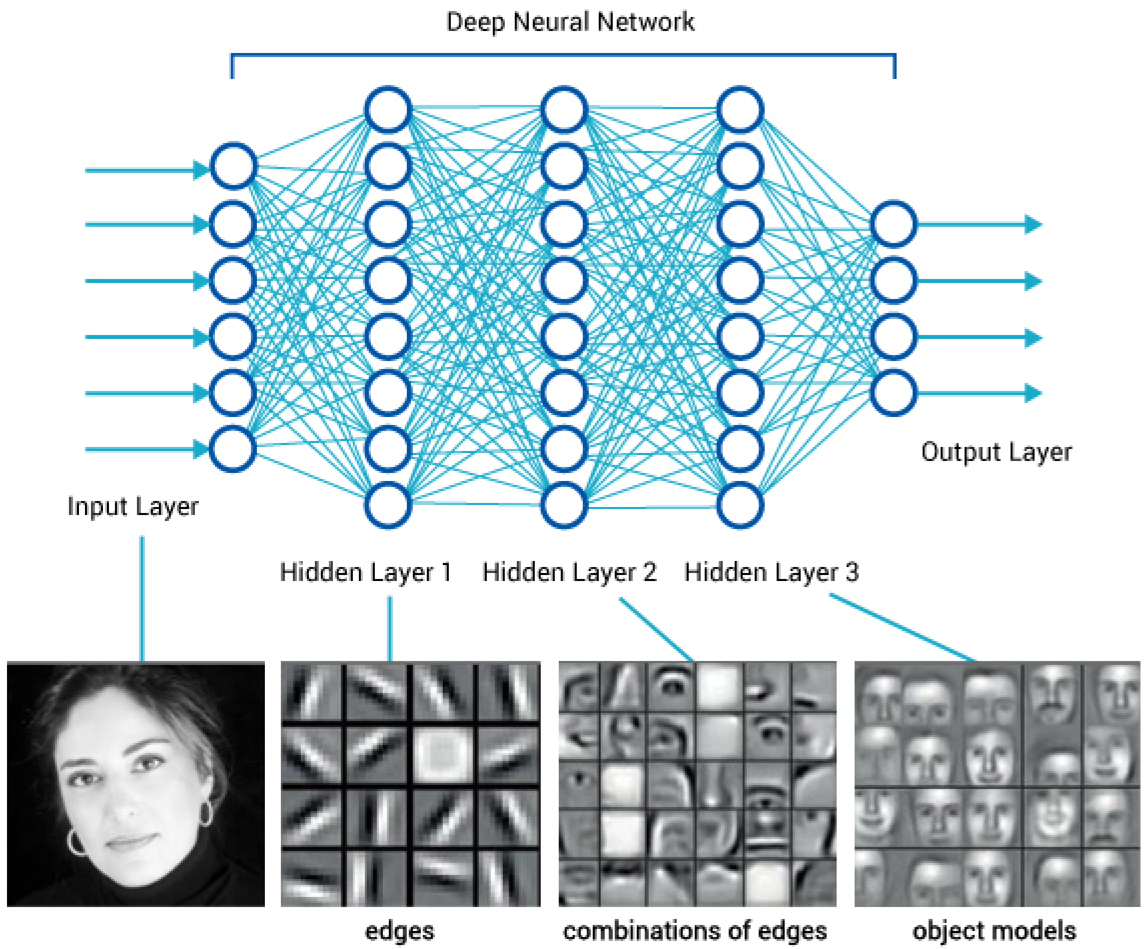
- Input is
  - high-dimensional (attribute-value pairs)
  - discrete or real-valued
  - possibly noisy [training, testing]
  - complete
  - (eg, raw sensor input)
- Output is
  - vector of values
  - discrete or real valued
  - “linear ordering"



$\Rightarrow \Re^n \to \Re$

- . . . have LOTS OF TIME to train (performance is fast)
- Form of target function is unknown
- Human readability / Explanability is NOT important

Deep Neural Network (Deep Learning)



R Packages

- Linear models with regularization (feature selection): glmnet
  - https://cran.r-project.org/web/packages/glmnet/index.html
- SVM: e1071
  - https://cran.r-project.org/web/packages/e1071/index.html
- Trees: rpart
  - https://cran.r-project.org/web/packages/rpart/index.html
- Random Forest: randomForest
  - https://cran.r-project.org/web/packages/randomForest/index.html
- Boosting: gbm
  - https://cran.r-project.org/web/packages/gbm/
- K-Nearest Neighbor: knn
  - https://cran.r-project.org/web/packages/kknn
- Neural Network: nnet
  - https://cran.r-project.org/web/packages/nnet/index.html
- Deep learning: http://deeplearning.net/software_links/