Data Mining: Foundation, Techniques and Applications

Lesson 6,7: Classification and Regression

Li Cuiping(李翠平)
School of Information
Renmin University of China

Anthony Tung(鄧锦浩)
School of Computing
National University of Singapore

Outline

- Introduction
- Data Preparation
- Linear Discriminant
- Decision Tree Building
- Support Vector Machine(SVM)
- Bayesian Learning
- Other Classification Methods
- Combining Classifiers
- Validation Methods
- Regression

Predictive Modeling

Goal: learn a mapping: $y = f(\boldsymbol{x};\theta)$

Need: 1. A model structure

2. A score function

3. An optimization strategy

Categorical $y \in \{c_1,\ldots,c_m\}$: classification

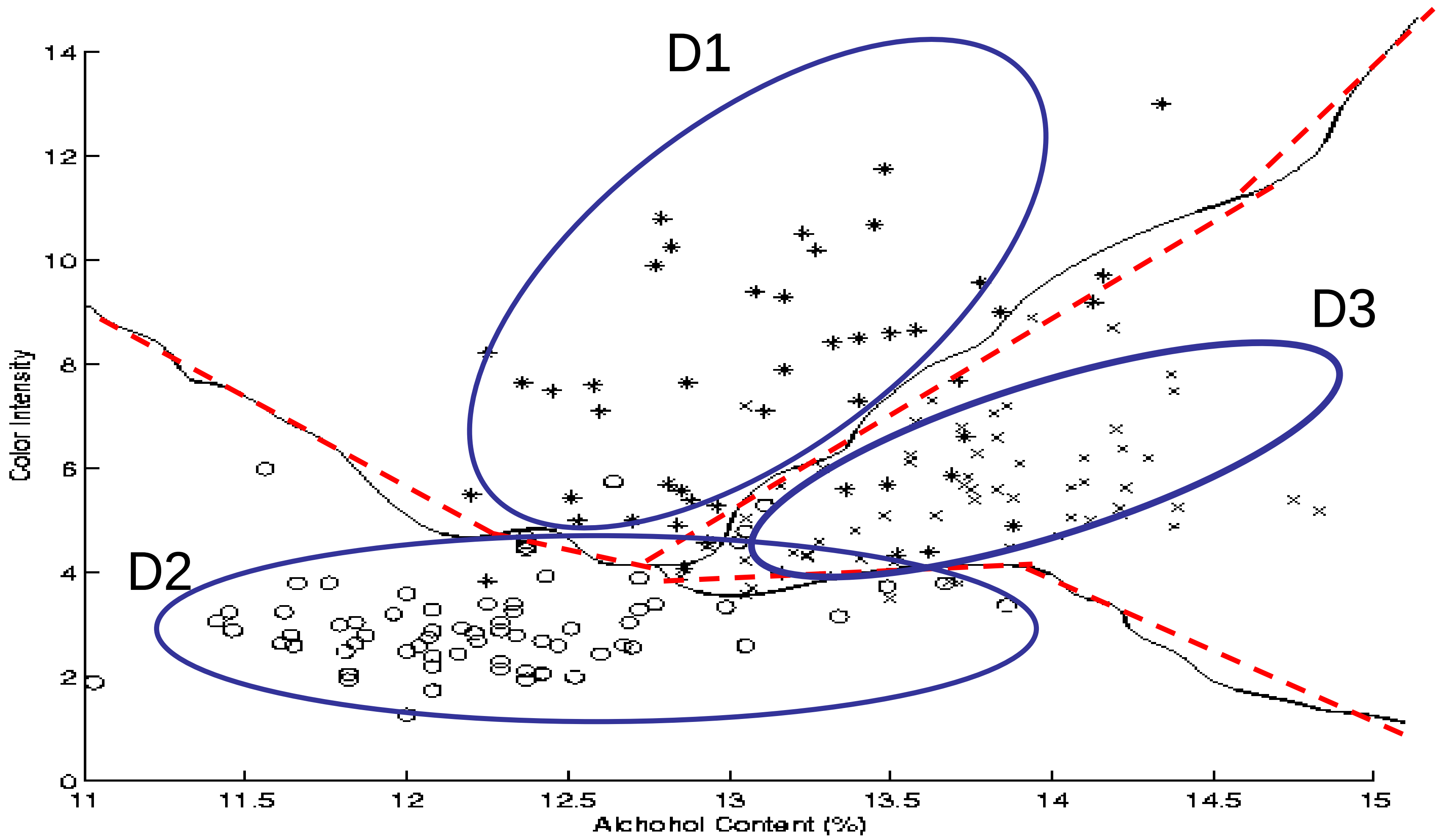
Real-valued y: regression

Note: usually assume $\{c_1,\ldots,c_m\}$ are mutually exclusive and exhaustive

Two class of classifiers

- Discriminative Classification
  - Provide decision surface or boundary to separate out the different classes
  - In real life, it is often impossible to separate out the classes perfectly
  - Instead, seek function $f(x;\theta)$ that maximizes some measure of separation between the classes. We call $f(x; \theta)$ the discriminant function.
- Probabilistic Classification
  - Focus on modeling the probability distribution of each class

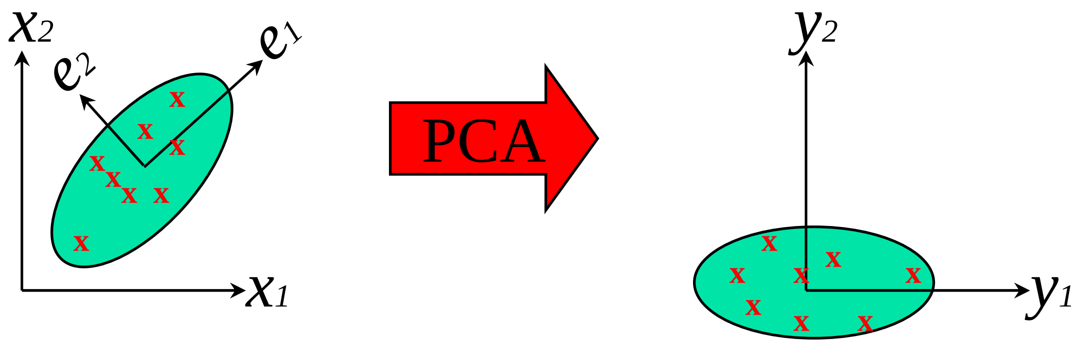
Discriminative vs Probabilistic Classification



Outline

- Introduction
- Data Preparation
- Linear Discriminant
- Decision Tree Building
- Support Vector Machine(SVM)
- Bayesian Learning
- Other Classification Methods
- Combining Classifiers
- Validation Methods
- Regression

PCA: Theory



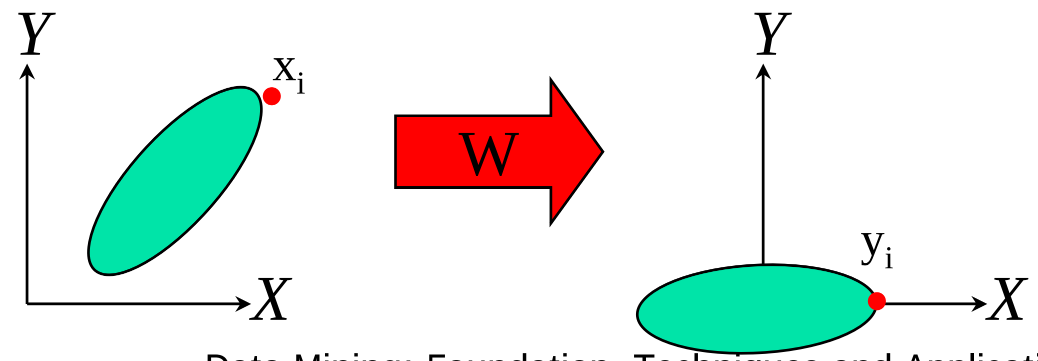
- Rotate the data so that its primary axes lie along the axes of the coordinate space and move it so that its center of mass lies on the origin.

PCA: Goal - Formally Stated

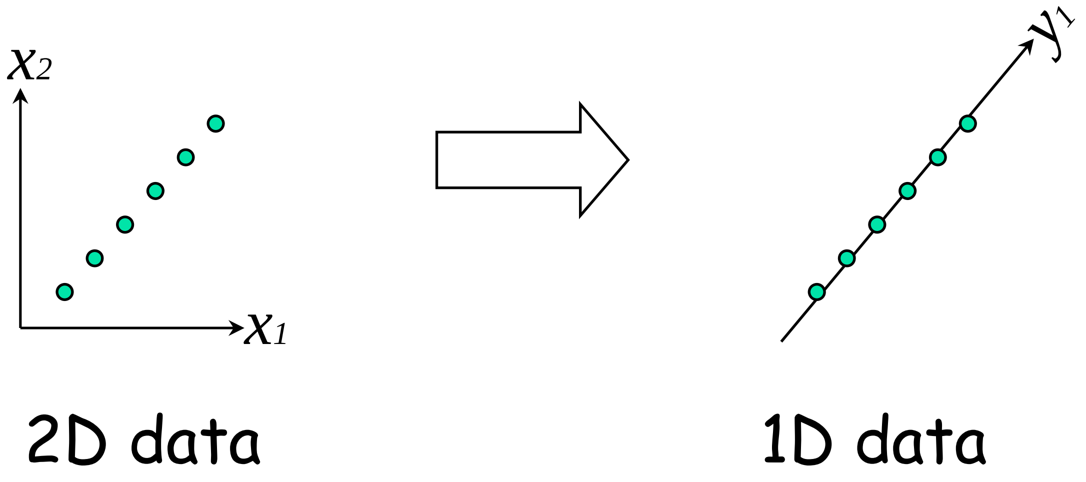
Problem formulation

- Input: $x=[x_1|\dots|x_N]_{d\times N}$ points in d-dimensional space
- Look for: W, a $d\times m$ projection matrix ($m\leq d$)
- S.t.: $y=[y_1|\dots|y_N]_{m\times N} = W^T [x_1|\dots|x_N]$...

...And correlation is minimized



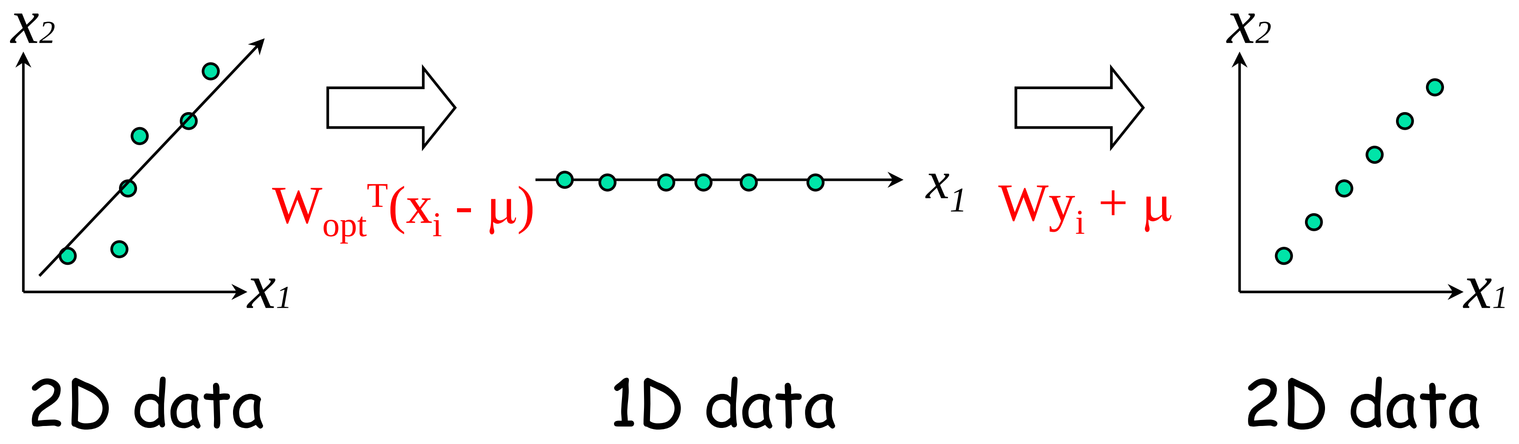
Dimension Reduction



Since there is no variance along one dimension, we only need a single dimension !!!

Data Loss

- Sample points can still be projected via the new m×d projection matrix W_{opt} and can still be reconstructed, but some information will be lost.



Basic: What is Covariance ?

- Given a d-dimensional space with N data points, the covariance between the i-th and j-th dimensions, represented as Cov_{ij} defined as follow:

$$Cov_{ij}(x) = \frac{1}{N}\sum_{k=1}^{N}(x_k[i]-\mu_i)(x_k[j]-\mu_j)$$

- where $x_k[i]$ represent the i-th dimension values for the kth data point and μ_i and μ_j represent the mean values of the data points along dimensions i and j respectively

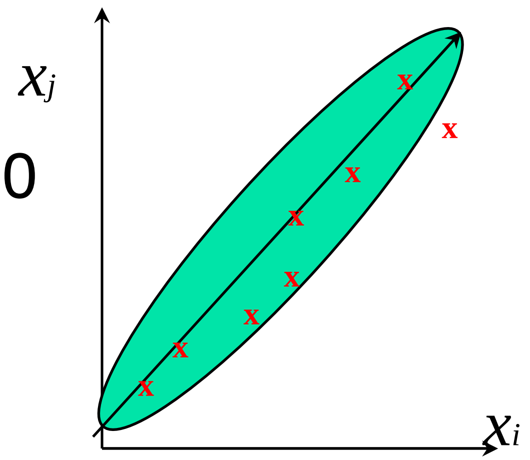
Example: Covariance

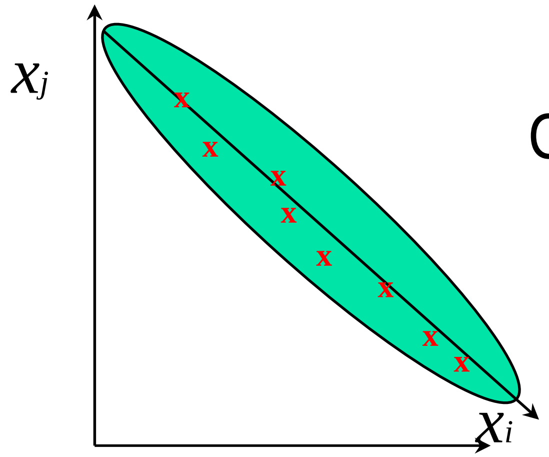
X_i	X_j
1	2
2	4
3	7
4	9
5	9

- μ_i = (1+2+3+4+5)/5 = 3
- μ_j = (2+4+7+9+9)/5 =6.2
- $Cov_{ij}(x)$= 1/5 * ((1-3)(2-6.2) + (2-3)(4-6.2) + (3-3)(7-6.2) + (4-3)(9-6.2) + (5-3)(9-6.2))
- = **4.04**

Covariance: Intuition

$Cov_{ij}>0$

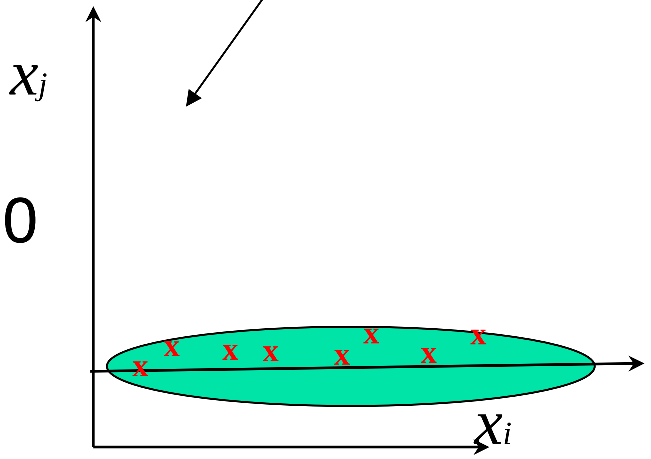


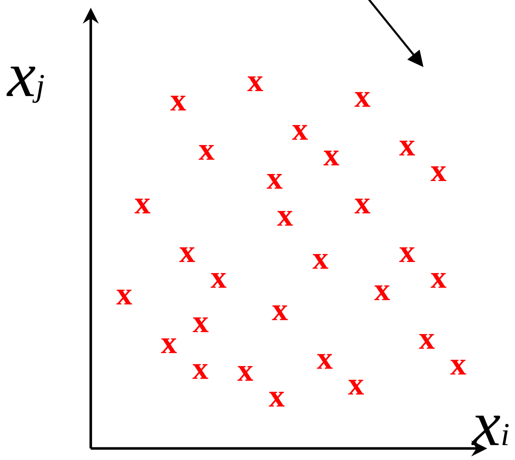


$Cov_{ij}<0$

How to separate these two cases ?

$Cov_{ij}=0$





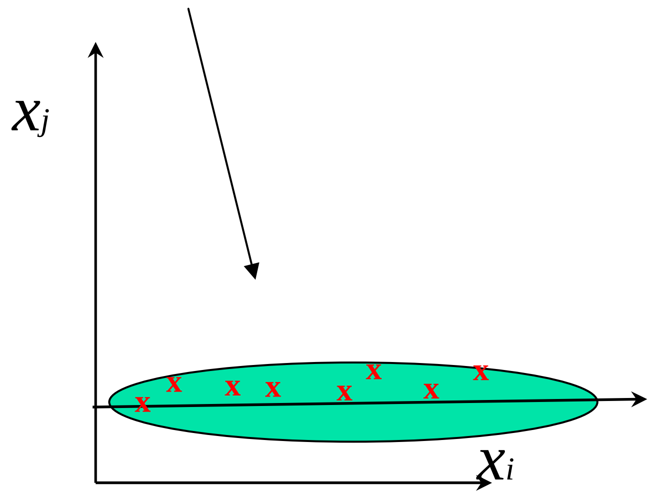
$Cov_{ij}= 0$

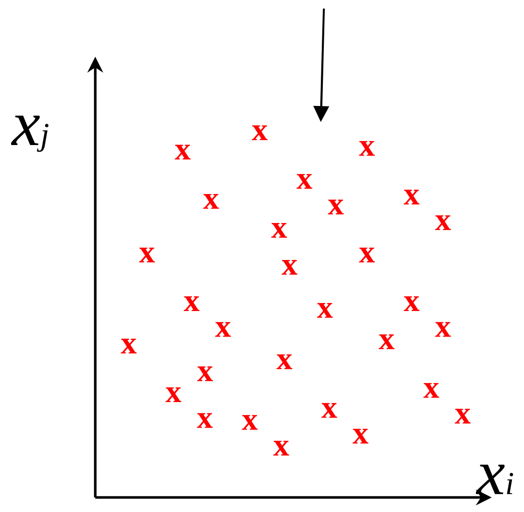
PCA: The Covariance Matrix

- Given a d-dimensional space with N data points, the covariance matrix Cov(x) is a d x d matrix in which the element at row i and column j contain the value $Cov_{ij}(x)$.

$Cov_{jj}(x)$ will be low

$Cov_{jj}(x)$ will be high





Properties of the Covariance Matrix(I)

- The ith column can be seen as a vector that represent the covariance interaction of the ith dimension with the rest of the dimensions

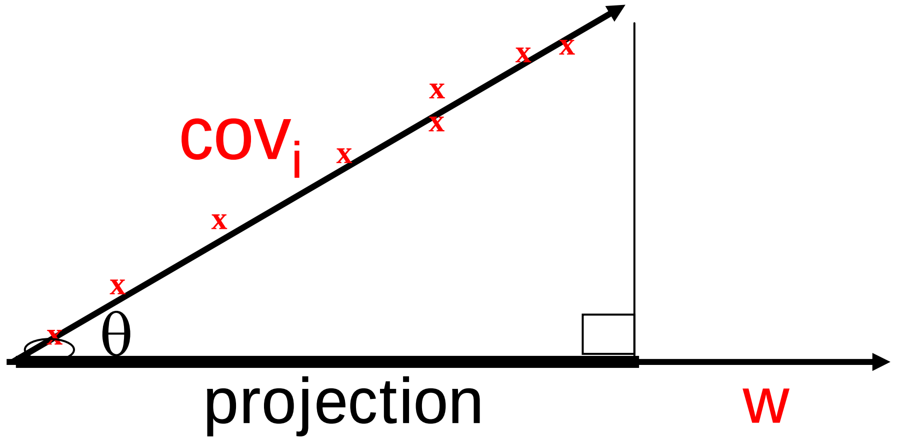
$$Cov(x) = \begin{bmatrix} \mathrm{cov}_{11} & \mathrm{cov}_{12} & \cdots & \mathrm{cov}_{1d} \\ \mathrm{cov}_{21} & \mathrm{cov}_{22} & \cdots & \mathrm{cov}_{2d} \\ \vdots & \vdots & \vdots & \vdots \\ \mathrm{cov}_{d1} & \mathrm{cov}_{d2} & \cdots & \mathrm{cov}_{dd} \end{bmatrix} = \begin{bmatrix} \mathbf{cov_1} & \mathbf{cov_2} & \cdots & \mathbf{cov}_d \end{bmatrix}$$

where

$$\mathbf{cov_1} = \begin{bmatrix} \mathrm{cov}_{11} \\ \mathrm{cov}_{21} \\ \vdots \\ \mathrm{cov}_{n1} \end{bmatrix} \qquad \mathbf{cov_2} = \begin{bmatrix} cov_{12} \\ cov_{22} \\ \vdots \\ cov_{d2} \end{bmatrix} \qquad \cdots \qquad \mathbf{cov}_d = \begin{bmatrix} cov_{1d} \\ cov_{2d} \\ \vdots \\ cov_{dd} \end{bmatrix}$$

Properties of the Covariance Matrix(II)

- Given the covariance vector of dimension i, $\text{cov}_i(x)$. We can compute the variance of it's projection along a **unit** vector w as $w.\text{cov}_i$.



Recall that $w.\text{cov}_i = |w||\text{cov}_i| \cos\theta$. Since $|w|$ is 1, we will have $|\text{cov}_i| \cos\theta$ which is the length of the projection of cov_i along w.

Properties of the Covariance Matrix(III)

- Assuming now we have $W = [w_1|\ldots|w_d]$, then $W^T \, Cov(x)$ give the magnitude of the projection that each covariance vector in Cov(x) have on $w_1,\ldots,w_d$.
- $$W^T \, Cov(x) = \begin{bmatrix} w_1{}^T \\ . \\ . \\ w_d{}^T \end{bmatrix} \begin{bmatrix} \mathbf{cov}_1 & \mathbf{cov}_2 & \cdots & \mathbf{cov}_d \end{bmatrix}$$

where

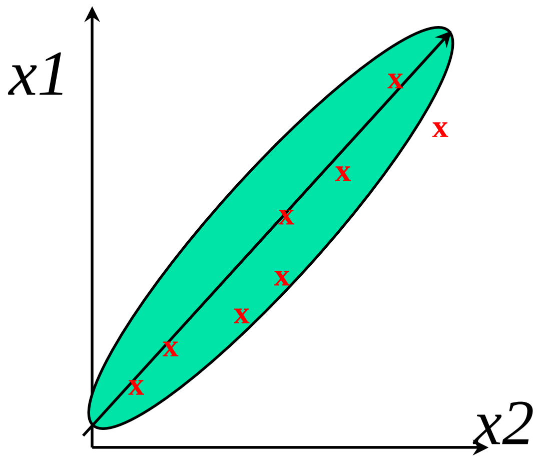
$$\mathbf{cov}_1 = \begin{bmatrix} cov_{11} \\ cov_{21} \\ \vdots \\ cov_{n1} \end{bmatrix} \qquad \mathbf{cov}_2 = \begin{bmatrix} cov_{12} \\ cov_{22} \\ \vdots \\ cov_{d2} \end{bmatrix} \qquad \cdots \qquad \mathbf{cov}_d = \begin{bmatrix} cov_{1d} \\ cov_{2d} \\ \vdots \\ cov_{dd} \end{bmatrix}$$

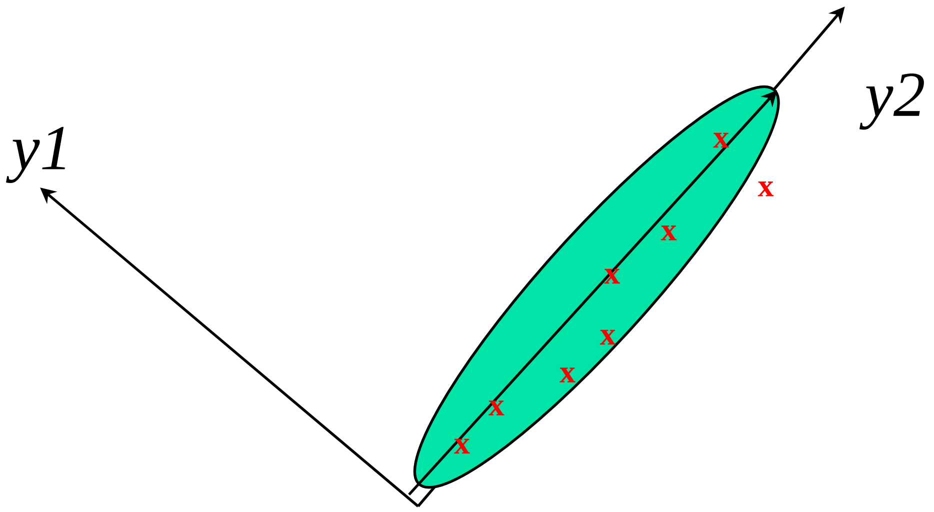
Properties of the Covariance Matrix(IV)

- In addition, W^T Cov(x) W give the variance of each covariance vector in Cov(x) when they are projected on $w_1,\ldots,w_d$.

PCA: Goal Revisited

- We want each of the data points $x_1,\ldots,x_N$ are transformed to $y_1,\ldots,y_N$ based on $W^T x_i$ for $1 <= i <= N$
- Look for: W s.t.
  - $[y_1|\ldots|y_N] = W^T [x_1|\ldots|x_N]$, and
  - correlation is minimized => **Cov(y) is diagonal!**





Selecting the Optimal W

- Note that Cov(y) can be expressed via Cov(x) and W as Cov(y) = W^T Cov(x) W. How do we find such W ?

$$\text{Cov}(y) = \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & \lambda_d \end{bmatrix} = W^T \text{ Cov}(x)\ W$$

$$\lambda_i = w_i^T \text{ Cov}(x)\ w_i \longrightarrow w^i . \lambda_i = \overset{1!}{\boxed{w_i . w_i^T}} \text{ Cov}(x)\ w_i$$

Selecting the Optimal W(II)

We thus have

$$\lambda_i w_i = \mathrm{Cov}(x)\ w_i$$

Therefore :

Choose W_{opt} to be the eigenvectors matrix:

$$W_{opt} = [w_1 | \ldots | w_d]$$

Where $\{w_i | i=1,\ldots,d\}$ is the set of the d-dimensional eigenvectors of Cov(x)

So...to sum up

- To find a more convenient coordinate system one needs to :

Calculate mean sample μ

Subtract it from all samples x_i

Calculate Covariance matrix for resulting samples

Find the set of eigenvectors for the covariance matrix

Create W_{opt}, the projection matrix, by taking as columns the eigenvectors calculated !

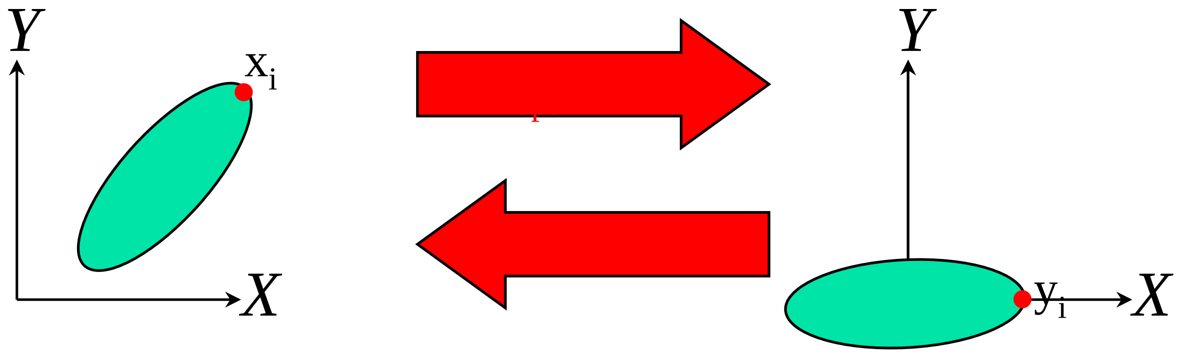
So...to sum up (cont.)

- Now we have that any point x_i can be projected to an appropriate point y_i by :

$$y_i = W_{opt}^T(x_i - \mu)$$

- and conversely (since $W^{-1} = W^T$)

$$Wy_i + \mu = x_i$$



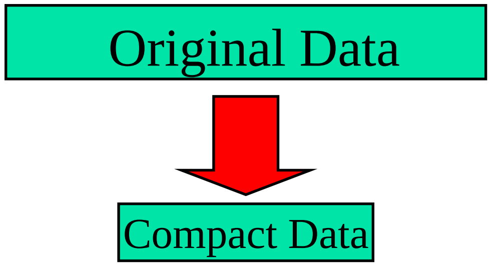
Data Reduction: Theory

- Each eigenvalue represents the the total variance in its dimension.
- So...
- Throwing away the least significant eigenvectors in W_{opt} means throwing away the least significant variance information !

Data Reduction: Practice

- Sort the d columns of the projection matrix W_{opt} in descending order of appropriate eigenvalues.
- Select the first m columns thus creating a new projection matrix of dimension $d \times m$

This will now be a projection from a d-dimensional space to an m-dimensional space (m < d) !



Data Loss

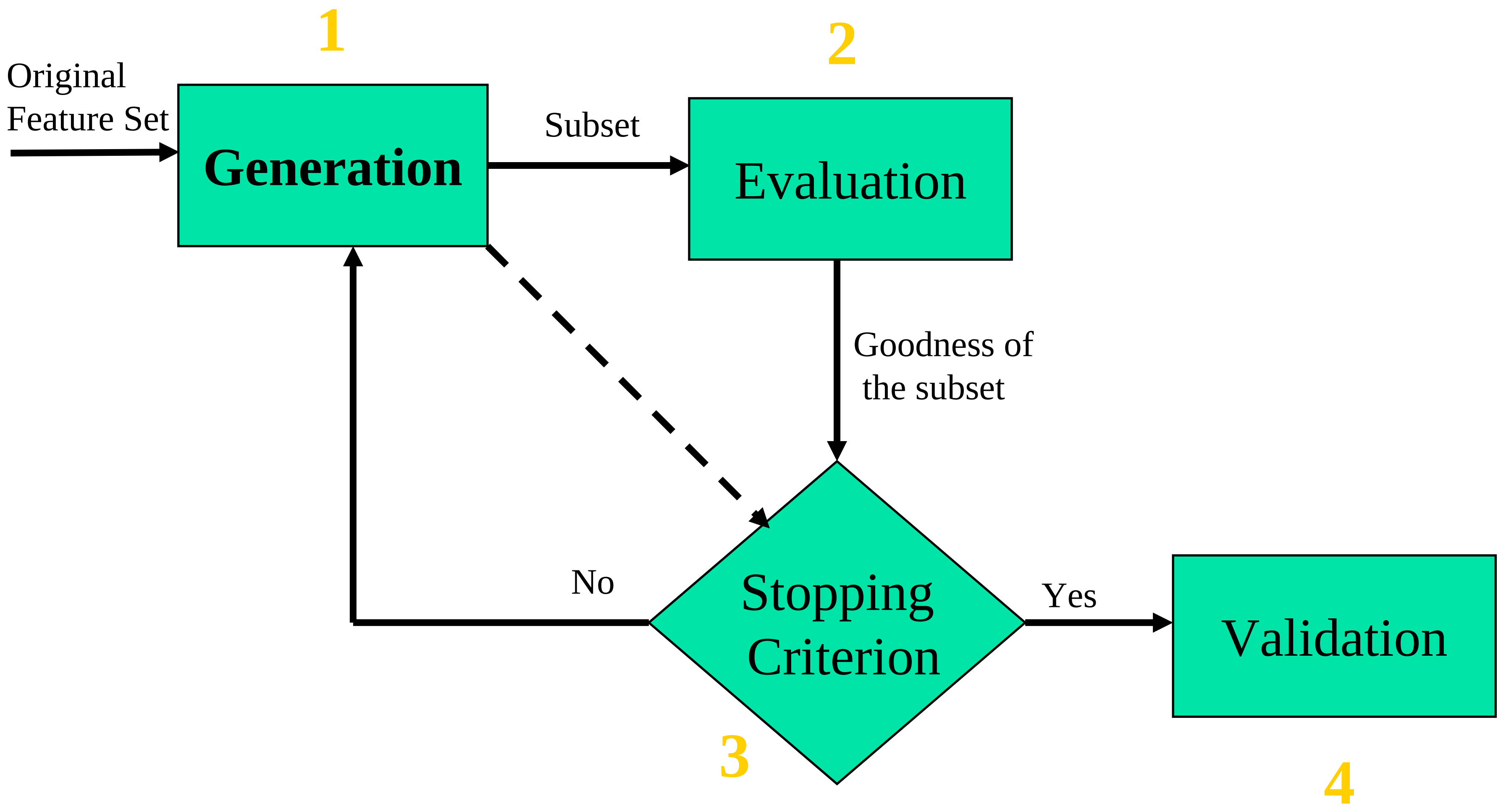
- It can be shown that the mean square error between x_i and its reconstruction using only m principle eigenvectors is given by the expression :

$$\boxed{\sum_{j=1}^{N} \lambda_j - \sum_{j=1}^{m} \lambda_j = \sum_{j=m+1}^{N} \lambda_j}$$

Feature Selection - Definition

- Attempt to select the <u>minimal</u> size subset of features according to <u>certain criteria</u>:
  - <u>classification accuracy</u> is at least <u>not</u> significantly dropped
  - Resulting class distribution given only values for the <u>selected features</u> is <u>as close as possible</u> to the original class distribution given all features

Four basic steps



Search *Procedure - Approaches*

- **Complete**
  - complete search for optimal subset
  - Guaranteed optimality
  - Able to backtrack
- **Random**
  - Setting a maximum number of iterations possible
  - Optimality depends on values of some parameters
- **Heuristic**
  - “hill-climbing”
  - iterate, then remaining features yet to be selected/rejected are considered for selection/rejection
  - simple to implement
  - fast in producing results
  - produce sub-optimal results

Evaluation Functions - Method Types

Filter methods

- Distance Measures
- Information Measures
- Dependence Measures
- Consistency Measures

Wrapper Methods

- Classifier Error Rate Measures

Relief :Underlying Concept

- A statistical method to select the relevant features
- It is a feature weight based algorithm
- It first chooses a sample of instances from the training set instances, & user must provide the no of instances
- It randomly pick the sample & find the near-hit and near-miss instances
- Negative weights are for irrelevant features & positive weights are for relevant and redundant features

Example

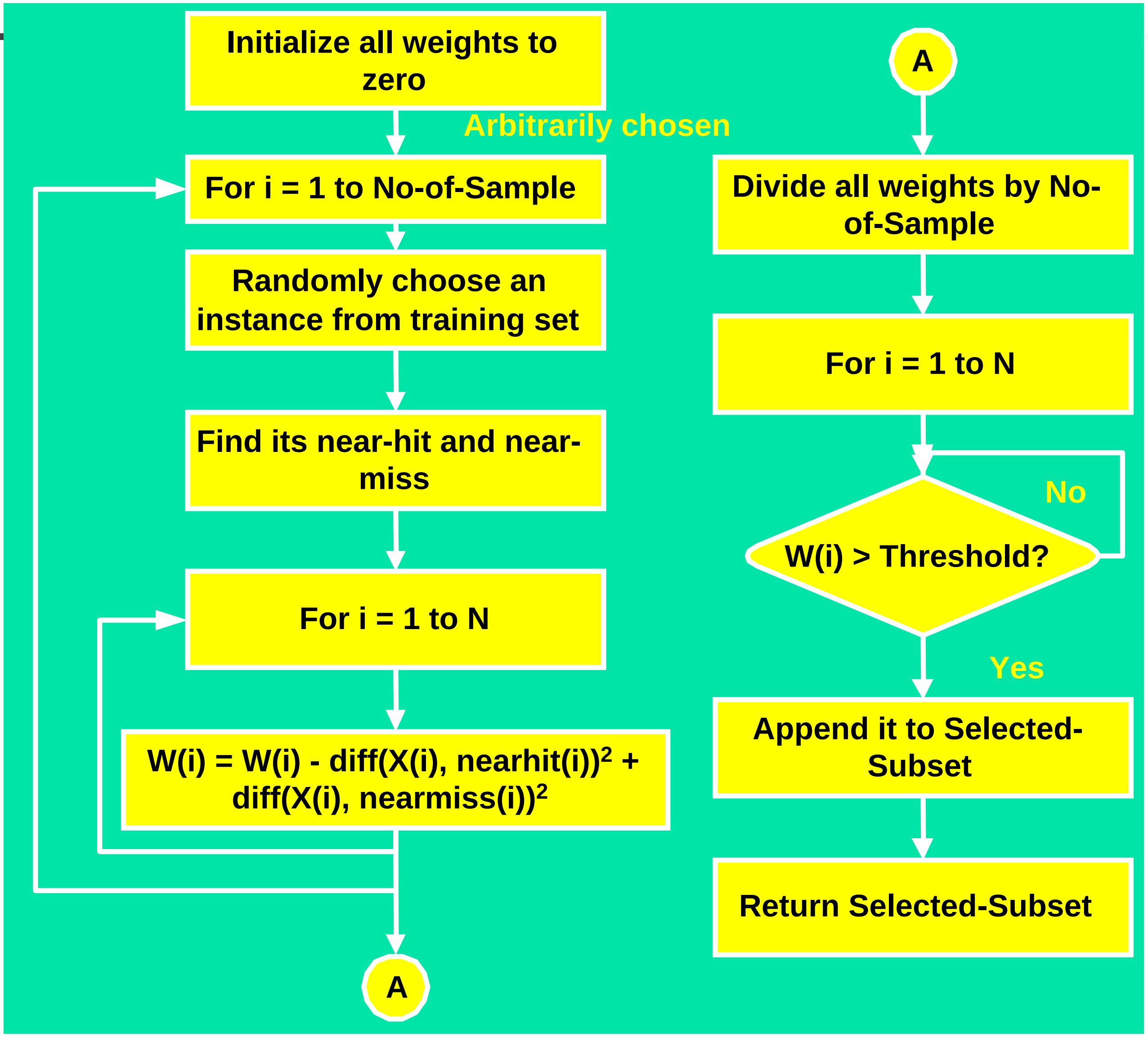
near hit

near miss

#	A0	A1	B0	B1	I	C	Class
1	0	0	0	0	0	1	0
2	0	0	0	1	1	1	0
3	0	0	1	0	0	1	0
4	0	0	1	1	0	0	1
5	0	1	0	0	0	1	0
6	0	1	0	1	1	1	0
7	0	1	1	0	1	0	0
8	0	1	1	1	0	0	1
9	1	0	0	0	1	1	0
10	1	0	0	1	1	0	0
11	1	0	1	0	0	1	0
12	1	0	1	1	0	0	1
13	1	1	0	0	0	0	1
14	1	1	0	1	0	1	1
15	1	1	1	0	1	0	1
16	1	1	1	1	0	0	1

sample

Relief(S, No-of-Sample, Threshold) - Algorithm


Initialize all weights to zero
Arbitrarily chosen
For i = 1 to No-of-Sample
Randomly choose an instance from training set
Find its near-hit and near-miss
For i = 1 to N
W(i) = W(i) - diff(X(i), nearhit(i))^2 + diff(X(i), nearmiss(i))^2
A
A
Divide all weights by No-of-Sample
For i = 1 to N
W(i) > Threshold?
No
Yes
Append it to Selected-Subset
Return Selected-Subset

Relief: Advantages and Disadvantages

- Advantages
  - Relief works for noisy and correlated features
  - It requires only linear time in the number of given features and no of instances (No-of-Sample)
  - It works both for nominal and continuous data
  - The procedure is very simple to implement & very fast
- Disadvantages
  - It often produces sub-optimal result because it does not remove redundant features
  - It works only for binary classes
  - User may find difficulty in choosing a proper No-of-Sample

LVF :Underlying Concept

- It randomly searches the space of instances which makes probabilistic choices faster to an optimal solution
- For each candidate subset, it calculates an inconsistency count based on the intuition
- An inconsistency threshold is fixed in the beginning (0 by default)
- Any subset with inconsistency rate greater than the threshold, is rejected

Example:LVF

Current={A0,B0,C},
Mixed Class value comb.= (0,1,0), (1,0,0), (1,0,1)
Class Distribution= (1,2), (1,1), (1,1)
Inconsistency= (3-2)+(2-1)+(2-1) = 3

#	A0	A1	B0	B1	I	C	Class	#	A0	A1	B0	B1	I	C	Class
1	0	0	0	0	0	1	0	9	1	0	0	0	1	1	0
2	0	0	0	1	1	1	0	10	1	0	0	1	1	0	0
3	0	0	1	0	0	1	0	11	1	0	1	0	0	1	0
4	0	0	1	1	0	0	1	12	1	0	1	1	0	0	1
5	0	1	0	0	0	1	0	13	1	1	0	0	0	0	1
6	0	1	0	1	1	1	0	14	1	1	0	1	0	1	1
7	0	1	1	0	1	0	0	15	1	1	1	0	1	0	1
8	0	1	1	1	0	0	1	16	1	1	1	1	0	0	1

LVF(S, MAX-TRIES, Incon-Threshold)

```
1. Initialize Selected-Subset to the original feature set

2. For i = 1 to MAX-TRIES
       Randomly choose a subset of feature, CURRENT
       If cardinality of CURRENT <= cardinality of Selected-Subset
          Calculate inconsistency rate of CURRENT
          If inconsistency rate < Incon-Threshold
             If cardinality < cardinality of Selected-Subset
                Selected-Subset=CURRENT
                Output CURRENT
             else
                Output CURRENT as 'yet another solution'

3. Return Selected-Subset
```

LVF: Advantages and Disadvantages

- *Advantages*
  - It is able to find the optimal subset even for databases with noise
  - User does not have to wait too long for a good subset
  - It is efficient and simple to implement and guarantee to find the optimal subset if resources permit
- Disadvantages
  - It may take more time to find the optimal subset (whether the data-set is consistent or not)

Problem with the Filter model

- [Recap] Feature selection:
  - Pre-processing step to classification
  - Classification accuracy not dropped
  - Accuracy is always wrt to the classification algorithm
- Assess merits of features from only the data and *ignores* the classification algorithm
  - generated feature subset may not be optimal for the target classification algorithm

Wrapper Model

- Use actual target classification algorithm to evaluate accuracy of each candidate subset

- Evaluation Criteria
  - Classifier error rate

- Generation method can be *heuristic, complete* or *random*

Wrapper compared to Filter

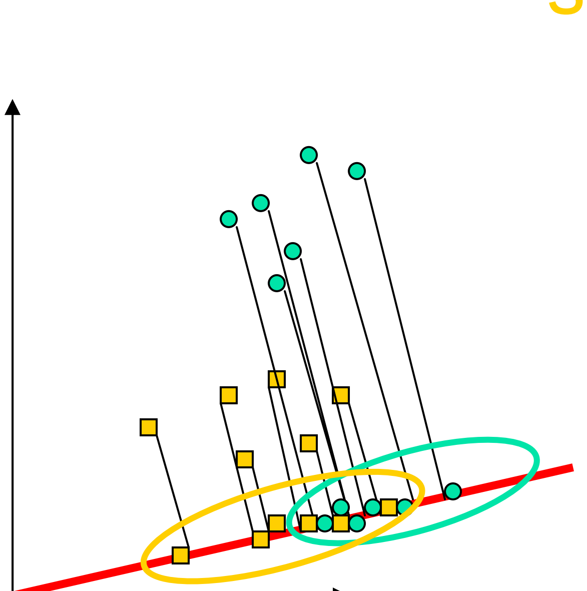
- 👍 higher accuracy
- 👎 higher computation cost
- 👎 lack generality

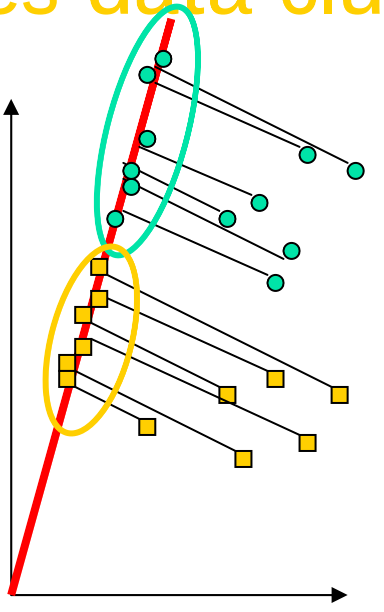
Outline

- Introduction
- Data Preparation
- Linear Discriminant
- Decision Tree Building
- Support Vector Machine(SVM)
- Bayesian Learning
- Other Classification Methods
- Combining Classifiers
- Validation Methods
- Regression

Fisher's Linear Discriminant

- Objective: Find a projection which separates data clusters


Poor separation


Good separation

FLD: Problem formulation

- Maximize the between-class variance while minimizing the within-class variance
- Data points: $\{\boldsymbol{x}_1, \cdots, \boldsymbol{x}_N\}$
- 2 classes: $\{c_1, c_2\}$
- Average of each class:

$$\mu_1 = \frac{1}{n_1} \sum_{x_k \in c_1} x_k, \quad \mu_2 = \frac{1}{n_2} \sum_{x_k \in c_2} x_k$$

- Total average: $\mu = \frac{1}{N} \sum_{k=1}^{N} x_k$

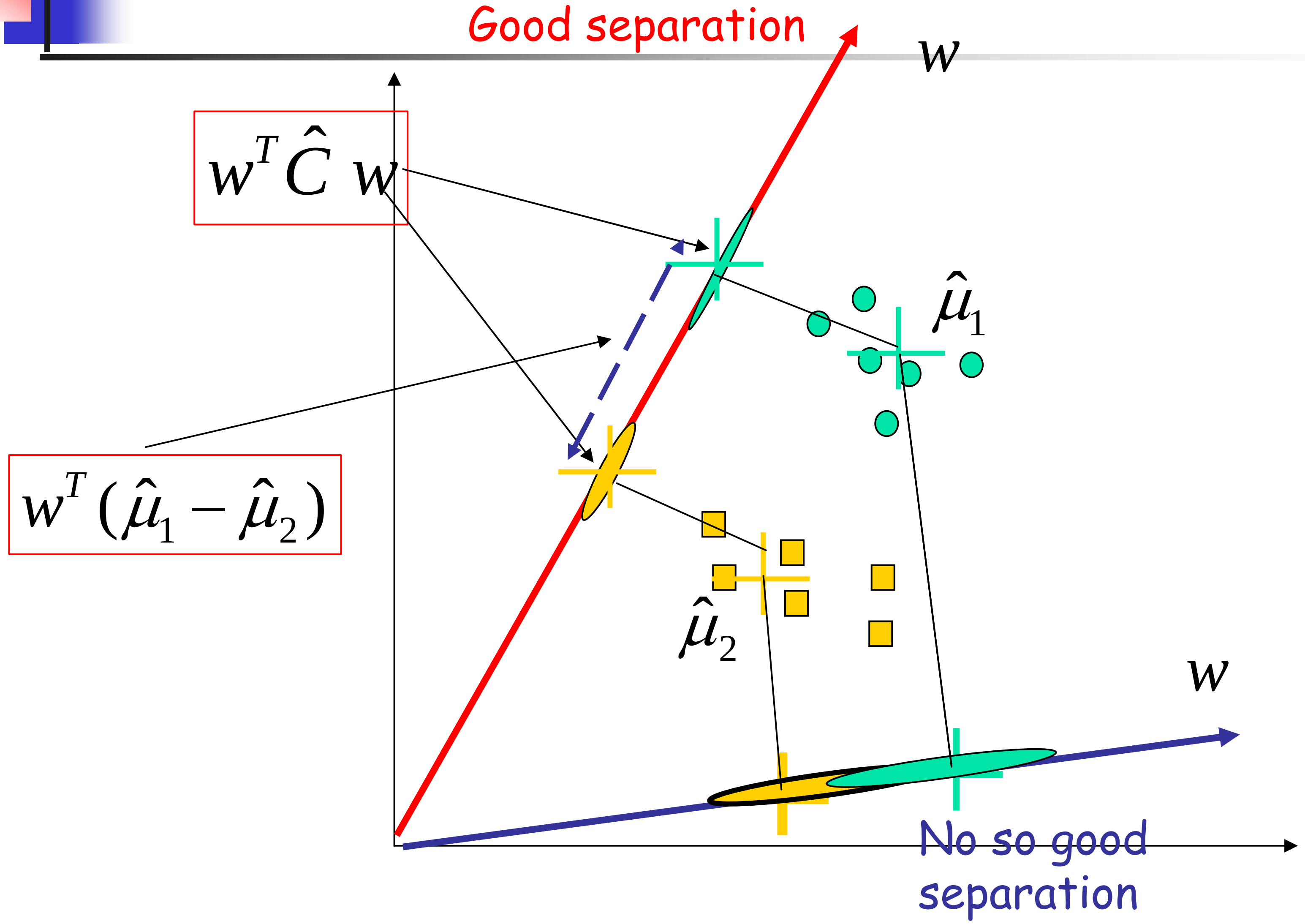
FLD: Scoring Function

- Let $\hat{C}_1$ and $\hat{C}_2$ be the covariance matrix of the two classes of points, their pooled covariance matrix will be computed as follow:

$$\hat{C} = \frac{1}{n_1 + n_2}\left(n_1\hat{C}_1 + n_2\hat{C}_2\right)$$

- Given a vector w, we measure the separability along w using the following score function

$$S(w) = \frac{w^T(\hat{\mu}_1 - \hat{\mu}_2)}{w^T\hat{C}\ w} = \frac{w.(\hat{\mu}_1 - \hat{\mu}_2)}{w^T\hat{C}\ w}$$


Good separation
w
$w^T \hat{C} w$
$\hat{\mu}_1$
$w^T(\hat{\mu}_1 - \hat{\mu}_2)$
$\hat{\mu}_2$
w
No so good separation

Solution for Maximizing S(w)

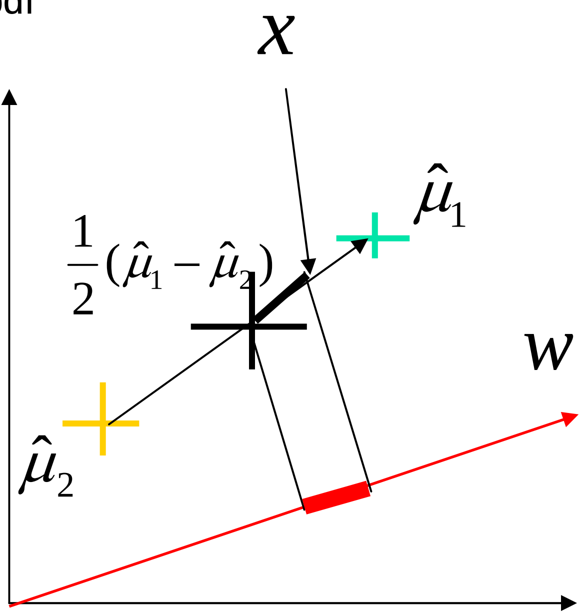
- Optimal solution for w is given by

$$\hat{w}_{lda} = \hat{C}^{-1}(\hat{\mu}_1 - \hat{\mu}_2)$$

http://www.stat.ucla.edu/~sczhu/Courses/UCLA/Stat_231/Lect7_Fisher.pdf

- Classify a point to class 1 if

$$\hat{w}_{lda}\left(x - \frac{1}{2}(\hat{\mu}_1 - \hat{\mu}_2)\right) > \log \frac{p(c_1)}{p(c_2)}$$

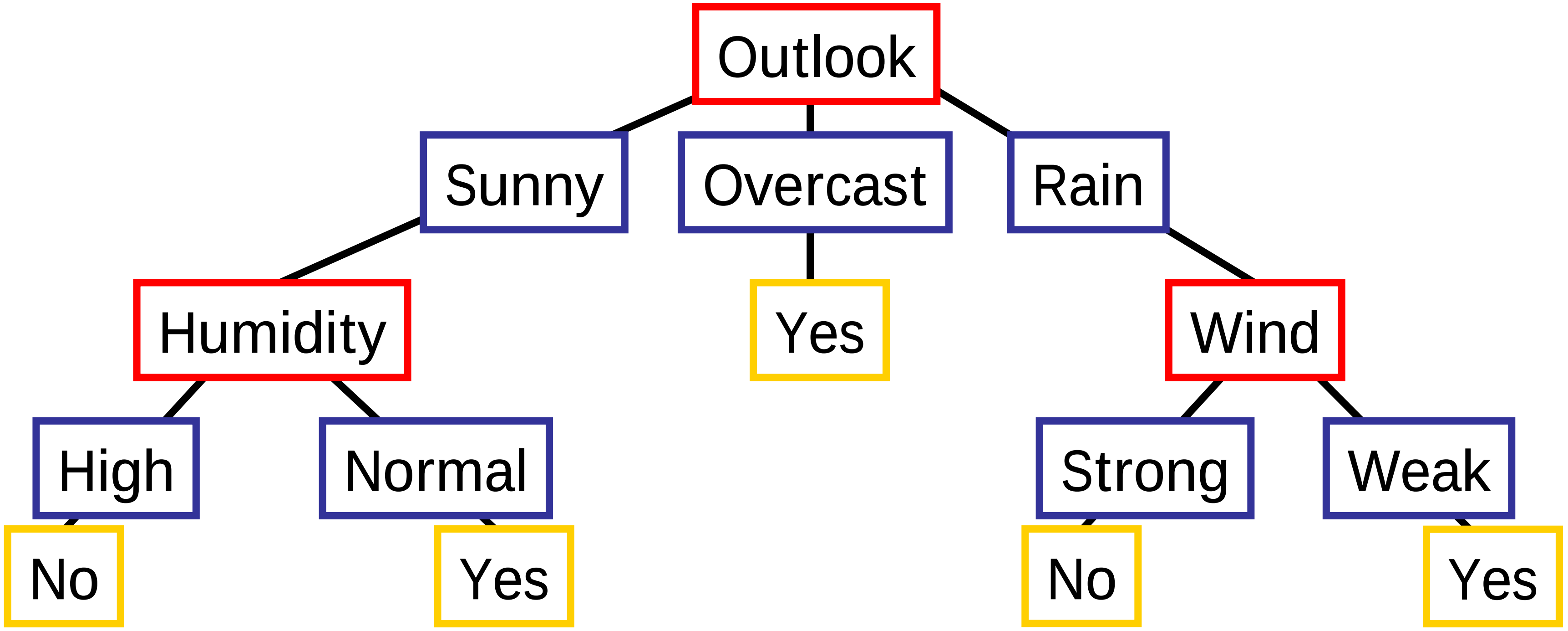


Outline

- Introduction
- Data Preparation
- Linear Discriminant
- Decision Tree Building
- Support Vector Machine(SVM)
- Bayesian Learning
- Other Classification Methods
- Combining Classifiers
- Validation Methods
- Regression

Decision Tree

- decision trees represent disjunctions of conjunctions

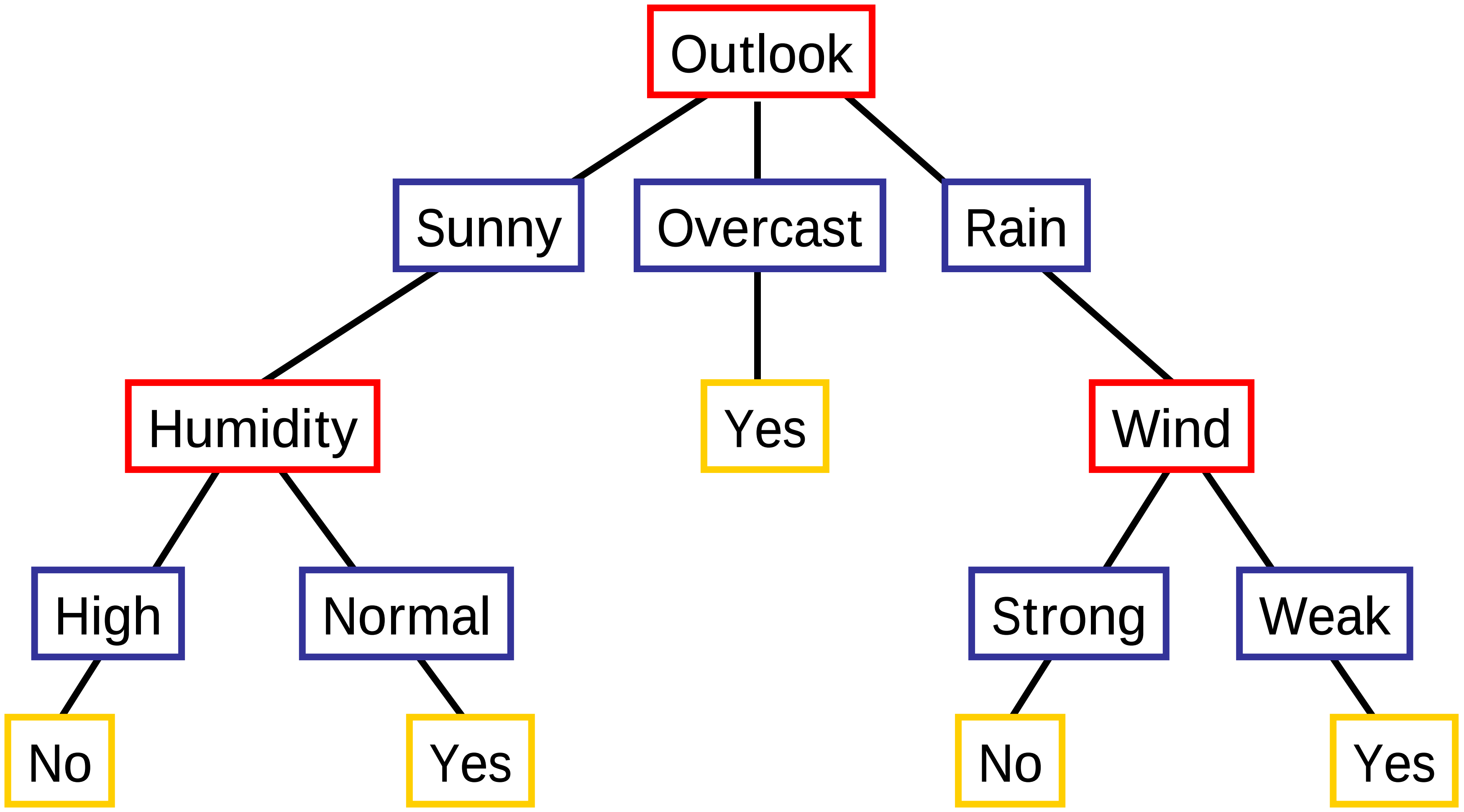


(Outlook=Sunny $\wedge$ Humidity=Normal)
$\vee$ (Outlook=Overcast)
$\vee$ (Outlook=Rain $\wedge$ Wind=Weak)

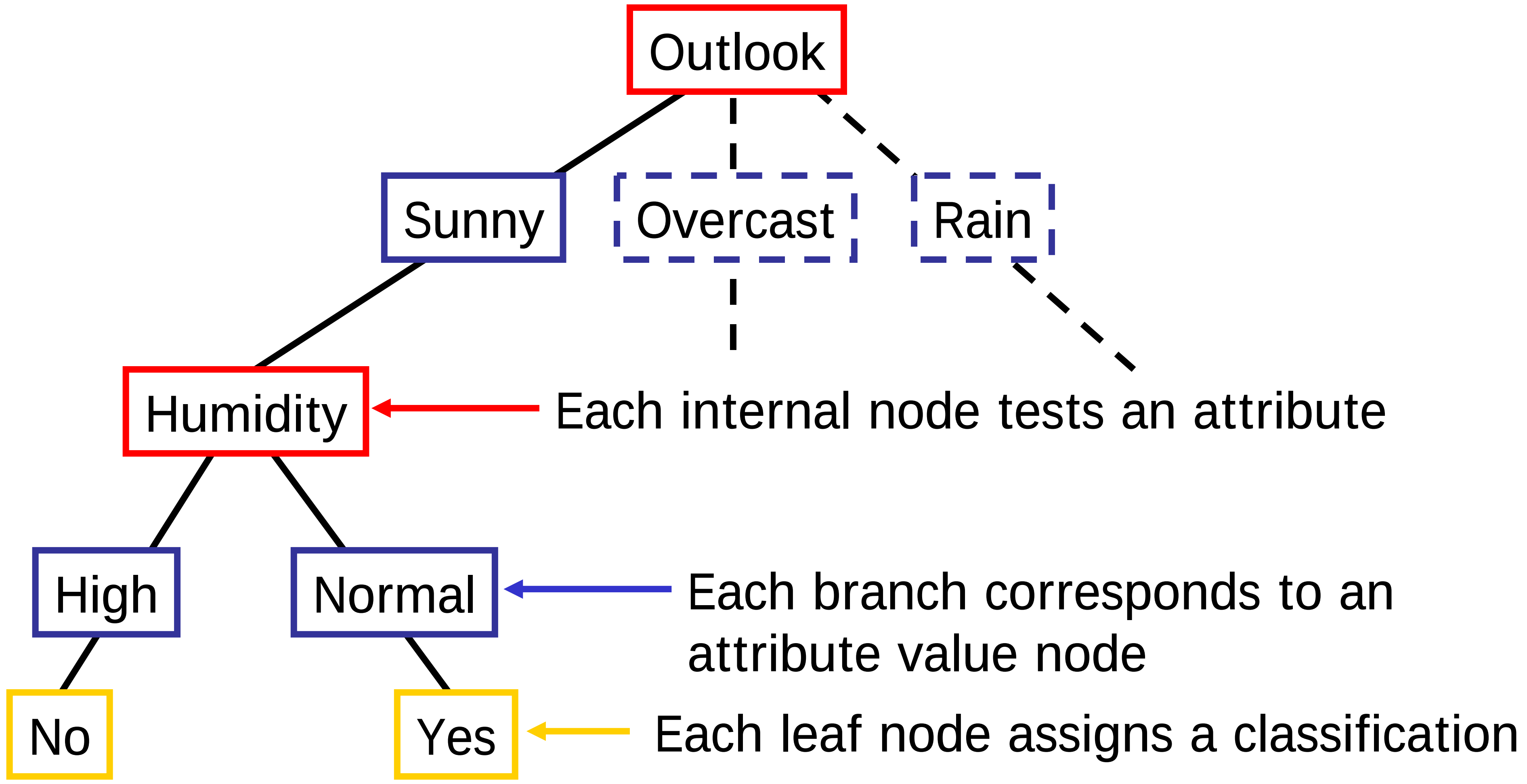
Training Dataset

Outlook	*Temp*	*Humid*	*Wind*	*PlayTennis*
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Overcast	Hot	High	Weak	Yes
Rain	Mild	High	Weak	Yes
Rain	Cool	Normal	Weak	Yes
Rain	Cool	Normal	Strong	No
Overcast	Cool	Normal	Strong	Yes
Sunny	Mild	High	Weak	No
Sunny	Cool	Normal	Weak	Yes
Rain	Mild	Normal	Weak	Yes
Sunny	Mild	Normal	Strong	Yes
Overcast	Mild	High	Strong	Yes
Overcast	Hot	Normal	Weak	Yes
Rain	Mild	High	Strong	No

Decision Tree for PlayTennis

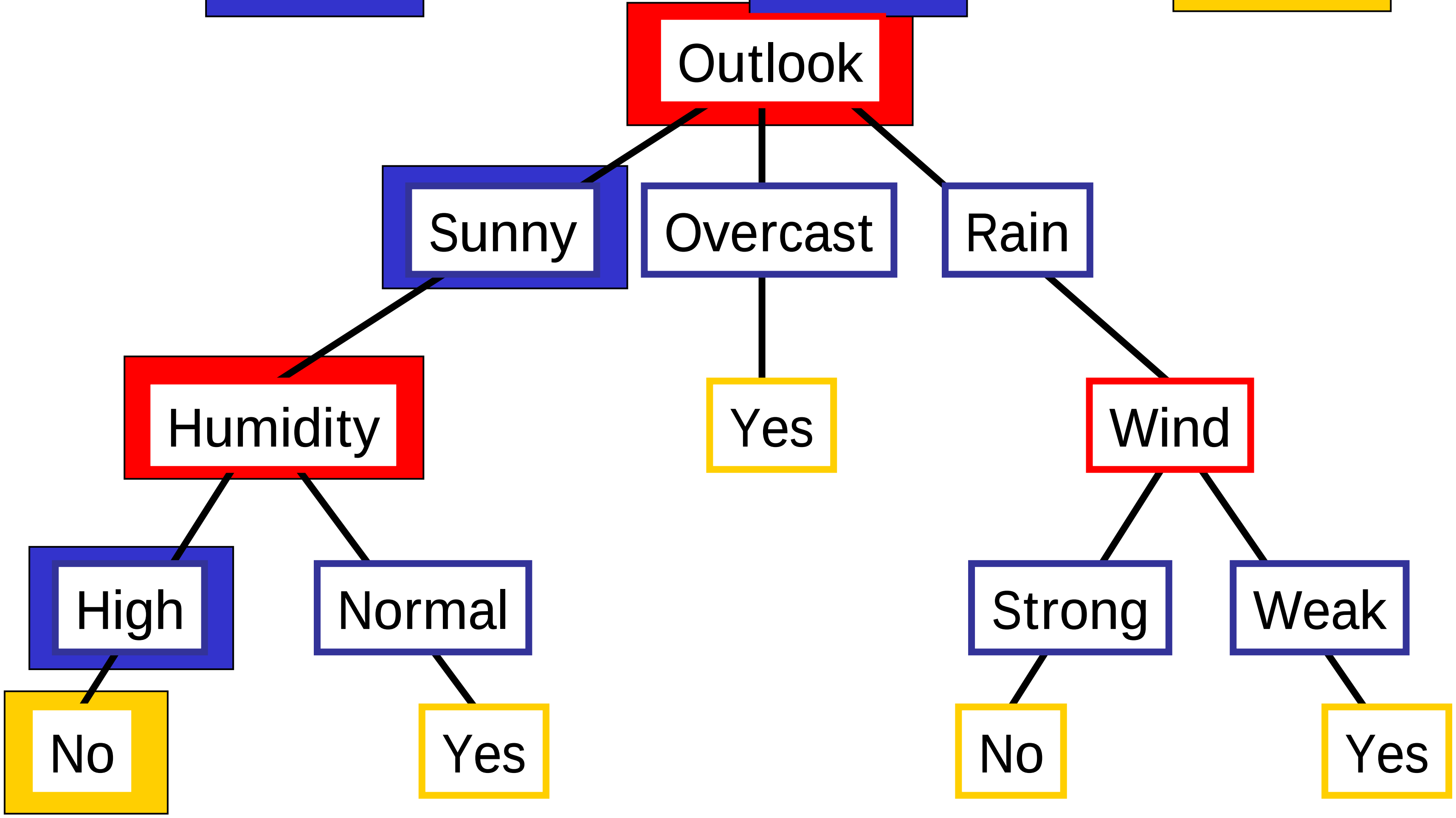


Decision Tree for PlayTennis


Outlook
Sunny
Overcast
Rain
Humidity
High
Normal
No
Yes
Each internal node tests an attribute
Each branch corresponds to an attribute value node
Each leaf node assigns a classification

Decision Tree for PlayTennis

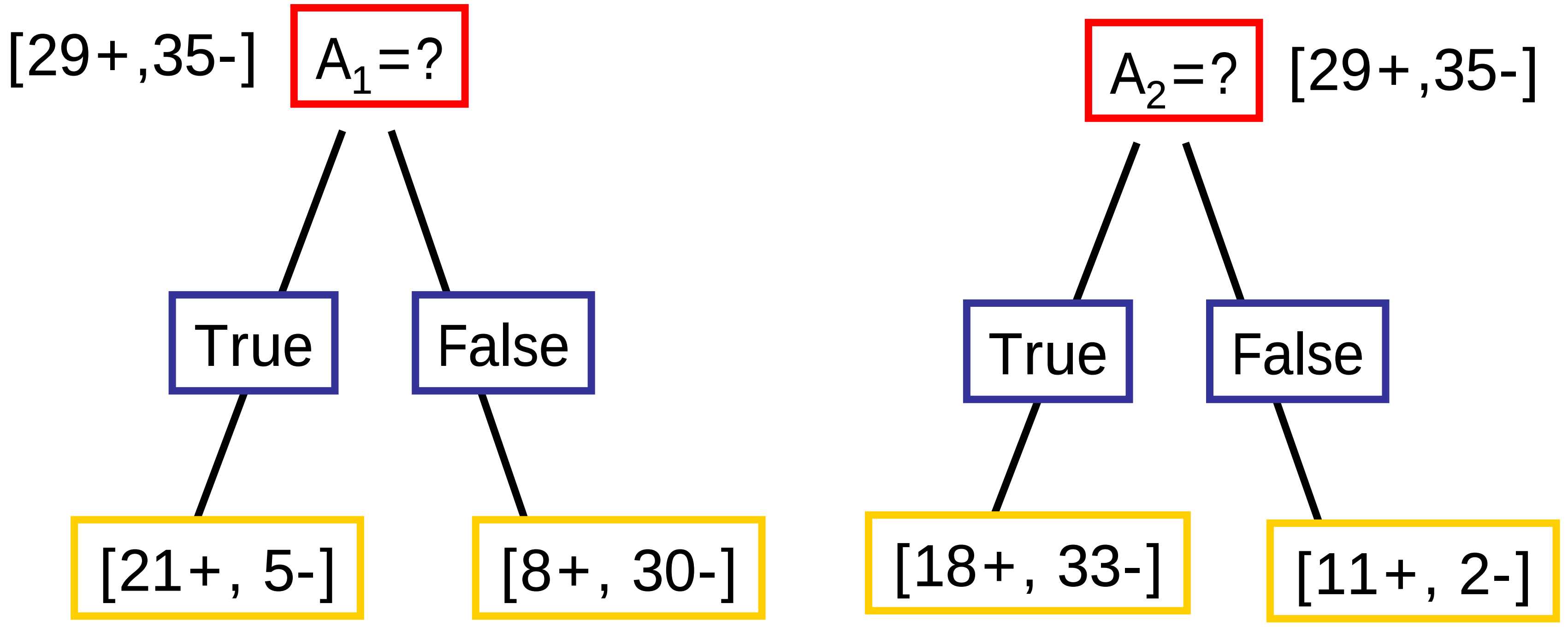
Outlook	Temperature	Humidity	Wind	PlayTennis
Sunny	Hot	High	Weak	?No



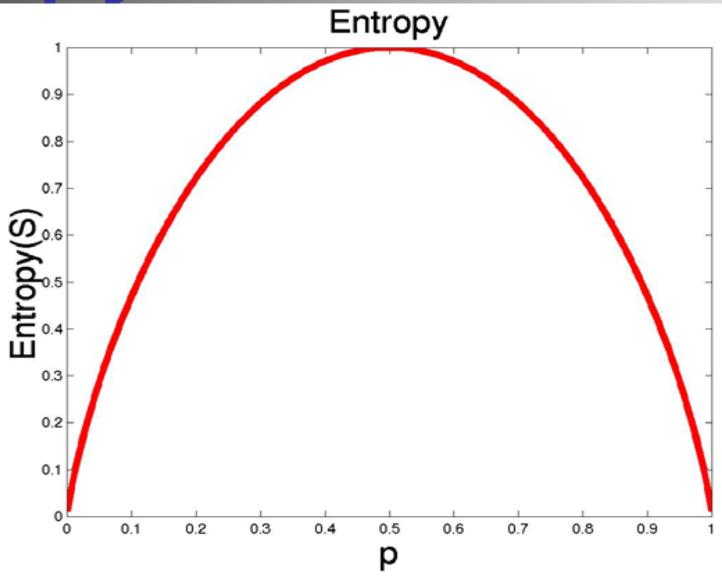
Top-Down Induction of Decision Trees

1. A ← the "best" decision attribute for next *node*
2. Assign A as decision attribute for *node*
3. For each value of A create new descendant
4. Sort training examples to leaf node according to the attribute value of the branch
5. If all training examples are perfectly classified (same value of target attribute) stop, else iterate over new leaf nodes.

Which Attribute is ”best”?



Entropy



- S is a sample of training examples
- p_+ is the proportion of positive examples
- p_- is the proportion of negative examples
- Entropy measures the impurity of S

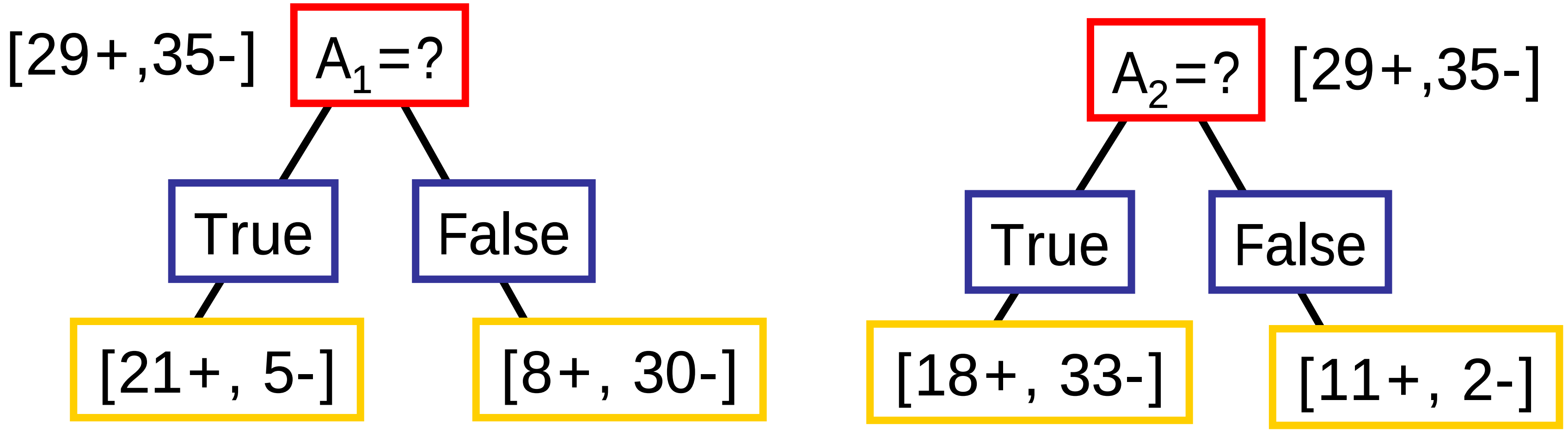
$$\text{Entropy}(S) = -p_+ \log_2 p_+ - p_- \log_2 p_-$$

Information Gain

- Gain(S,A): expected reduction in entropy due to sorting S on attribute A

$$\text{Gain}(S,A)=\text{Entropy}(S) - \sum_{v\in \text{values}(A)} |S_v|/|S| \ \text{Entropy}(S_v)$$

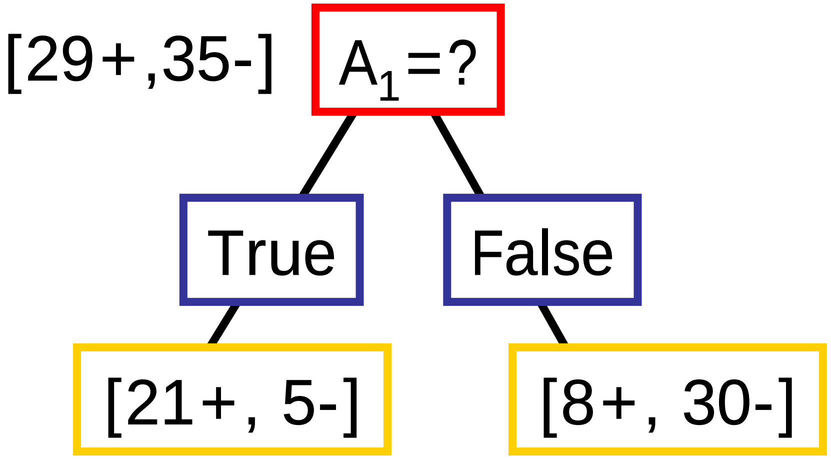
$$\text{Entropy}([29+,35-]) = -29/64 \log_2 29/64 - 35/64 \log_2 35/64 = 0.99$$

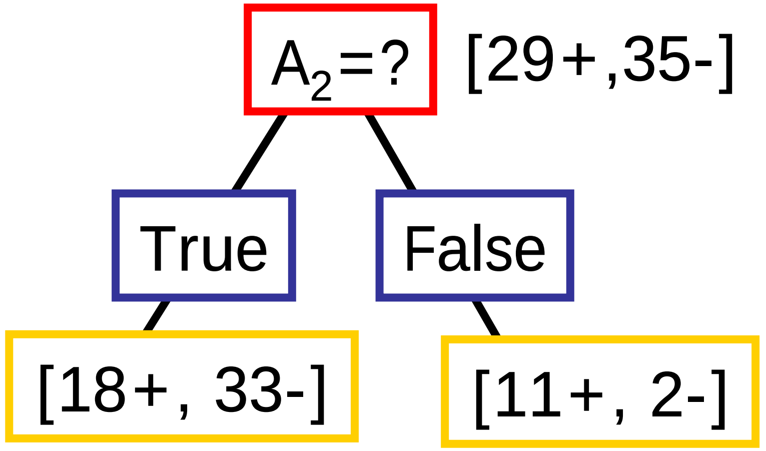


Information Gain

Entropy([21+,5-]) = 0.71
Entropy([8+,30-]) = 0.74
Gain(S,A_1)=Entropy(S)
-26/64*Entropy([21+,5-])
-38/64*Entropy([8+,30-])
=0.27

Entropy([18+,33-]) = 0.94
Entropy([8+,30-]) = 0.62
Gain(S,A_2)=Entropy(S)
-51/64*Entropy([18+,33-])
-13/64*Entropy([11+,2-])
=0.12





Entropy

- Entropy(S)= expected number of bits needed to encode class (+ or -) of randomly drawn members of S (under the optimal, shortest length-code)

Why?

- Information theory optimal length code assign

  $-\log_2 p$ bits to messages having probability p.
- So the expected number of bits to encode

  (+ or -) of random member of S:

$$-p_+ \log_2 p_+ - p_- \log_2 p_-$$

Alternative Measures

- Gain ratio: penalize attributes like date by incorporating split information
  - Split information is sensitive to how broadly and uniformly the attribute splits the data

$$SplitInformation(S, A) \equiv -\sum_{i=1}^{c} \frac{|S_i|}{|S|} \log_2 \frac{|S_i|}{|S|}$$

- Gain ratio can be undefined or very large
  - Only test attributes with above average Gain

$$GainRatio(S, A) \equiv \frac{Gain(S, A)}{SplitInformation(S, A)}$$

Gini Index

- A data set S contains examples from n classes where p_j is the relative frequency of class j in S

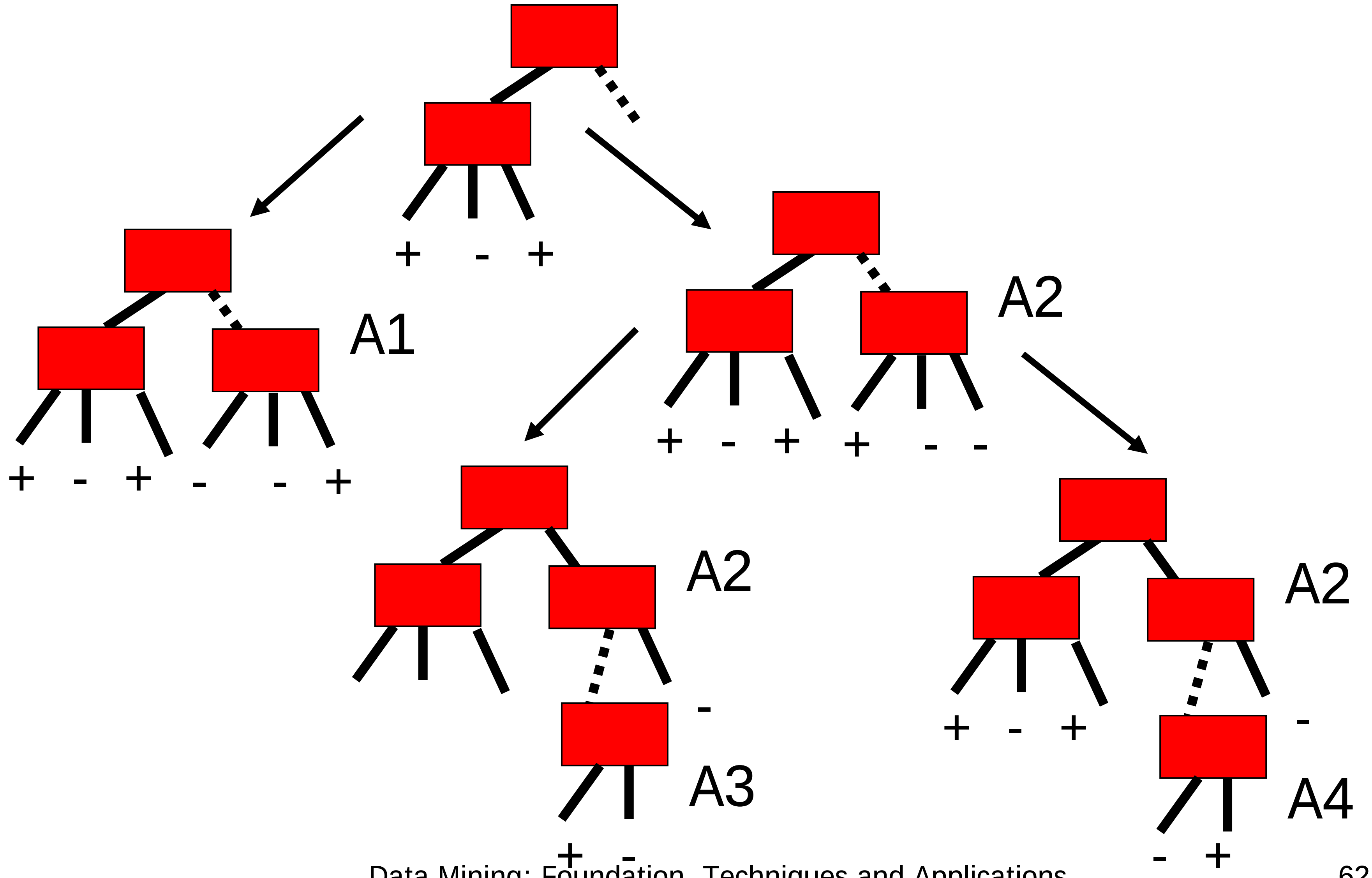
$$gini(T) = 1 - \sum_{j=1}^{n} p_j^2$$

- A data set S is split into two subsets S_1 and S_2 with sizes N_1 and N_2 respectively

$$gini_{split}(T) = \frac{N_1}{N} gini(T_1) + \frac{N_2}{N} gini(T_2)$$

- The attribute provides the smallest $gini_{split}(T)$ is chosen to split the node

Hypothesis Space Search ID3



Inductive Bias

- Preference for short trees, and for those with high information gain attributes near the root
- Bias is a *preference* for some model, rather than a *restriction* of the model space
- Occam's razor: prefer the shortest (simplest) hypothesis that fits the data

Occam’s Razor

Why prefer short hypotheses?

Argument in favor:

- Fewer short hypotheses than long hypotheses
- A short hypothesis that fits the data is unlikely to be a coincidence
- A long hypothesis that fits the data might be a coincidence

Argument opposed:

- There are many ways to define small sets of hypotheses
- E.g. All trees with a prime number of nodes that use attributes beginning with ”Z”
- What is so special about small sets based on *size* of hypothesis

Overfitting

- A decision tree T overfits the training data if ∃ alternative tree T’ s.t. T has a higher accuracy than T’ over the training examples, but T’ has a higher accuracy than T over the entire distribution of data

Avoid Overfitting

- Prepruning: stop growing the tree earlier
  - Difficult to choose an appropriate threshold
- Postpruning: remove branches from a "fully grown" tree
  - Use an independent set of data to prune
- Key: how to determine the correct final tree size

Discretizing Continuous Values

- Turn continuous values into discrete values
- Sort the examples according to their values for A
- For each ordered pair X_i, X_{i+1} in the sorted list,
  If the category of X_i and X_{i+1} are different,
  Then use the midpoint between their values as a candidate threshold

Value:	10		15		21		28		32		40		50
Class:	No		Yes		Yes		No		Yes		Yes		No
Threshold:		12.5				24.5		30				45	

Unknown Attribute Values

What if some examples have missing values for A?

Use training example anyway

- If node n tests A, assign most common value of A among other examples
- Assign most common value of A among other examples with same target value
- Assign probability pi to each possible value vi of A
  - Assign fraction pi of example to each descendant in tree
- Classify new examples in the same fashion

Classification in Large Databases

- What about the training data not in main memory?
- Scalability: build classifiers for large data sets with many attributes in a reasonable speed
- Why decision tree induction in data mining?
  - Relatively faster learning speed (than other classification methods)
  - Convertible to simple and easy to understand classification rules
  - Can use SQL queries for database accesses
  - Comparable classification accuracy with other methods

SLIQ

- Assumption: the training data set cannot be held in memory
  - Bottleneck: determining the best split for each attribute
  - Have to sort examples by attributes repeatedly
- Presorted attribute lists and class list
- Breadth-first growth of decision trees
  - Grow one level with a single, complete pass over the data

Attribute Lists in SLIQ

Training data

Age	Salary	Class
30	65	G
23	15	B
40	75	G
55	40	B
55	100	G
45	60	G

Attribute lists

Age	Class list index
23	2
30	1
40	3
45	6
55	5
55	4

Salary	Class list index
15	2
40	4
60	6
65	1
75	3
100	5

Class list

Index	Class	Leaf
1	G	N1
2	B	N1
3	G	N1
4	B	N1
5	G	N1
6	G	N1

From SLIQ to SPRINT

- The class list in SLIQ must stay in memory
  - Bottleneck: the class list can be huge
- SPRINT: put class information in attribute lists
  - No class list anymore
- Parallelizing classification
  - Partition the attribute lists

Example of Attribute Lists

Training data

Age	Salary	Class
30	65	G
23	15	B
40	75	G
55	40	B
55	100	G
45	60	G

Attribute lists

Age	Class	rid
23	B	2
30	G	1
40	G	3
45	G	6
55	G	5
55	B	4

Salary	Class	rid
15	B	2
40	B	4
60	G	6
65	G	1
75	G	3
100	G	5

RainForest: A Generic Framework

- What is the bottleneck of scalability?
  - Computing the attribute-value, class label (AVC-group) for each node
- RainForest: separate quality and scalability designs, focus on scalability
  - A set of algorithms for fast AVC-group computation

Things to ponder

- Is a decision tree a discriminant classifier or a probabilistic classifier ?
- In discriminant classifier, it seems that we have an assumption that all predictive attributes are numerical attributes. Is it true ? What happen when there are categorical attributes ?

Outline

- Introduction
- Data Preparation
- Linear Discriminant
- Decision Tree Building
- Support Vector Machine(SVM)
- Bayesian Learning
- Other Classification Methods
- Combining Classifiers
- Validation Methods
- Regression

SVM: Introduction

- Widely used method for learning classifiers and regression models
- Has some theoretical support from Statistical Learning Theory
- Empirically works very well, at least for some classes of problems

VC Dimension

l observations consisting of a pair: $\boldsymbol{x}_i \in \mathrm{R}^n$, $i=1,\ldots,l$ and the associated "label" $y_i \in$ **{-1,1}**

Assume the observations are iid from $P(\boldsymbol{x},y)$

Have a "machine" whose task is to learn the mapping $\boldsymbol{x}_i \rightarrow y_i$

Machine is defined by a set of mappings $\boldsymbol{x} \rightarrow f(\boldsymbol{x},\alpha)$

Expected test error of the machine (risk):

unknown

$$R(\alpha) = \int \frac{1}{2} |y - f(x,\alpha)| \underbrace{dP(x,y)}_{\text{unknown}}$$

Empirical risk(from training data):

$$R_{emp}(\alpha) = \frac{1}{2l} \sum_{i=1}^{l} |y - f(x_i,\alpha)|$$

VC Dimension (cont.)

Choose some η between 0 and 1. Vapnik (1995) showed that with probability 1- η :

$$R(\alpha) \leq R_{emp}(\alpha) + \sqrt{\frac{h(\log(2l/h)+1) - \log(\eta/4)}{l}}$$

•*h* is the Vapnik Chervonenkis (VC) dimension and is a measure of the capacity or complexity of the machine.

•Note the bound is independent of $P(\mathbf{x},y)$!!!

•If we know *h*, can readily compute the RHS. This provides a principled way to choose a learning machine.

VC Dimension (cont.)

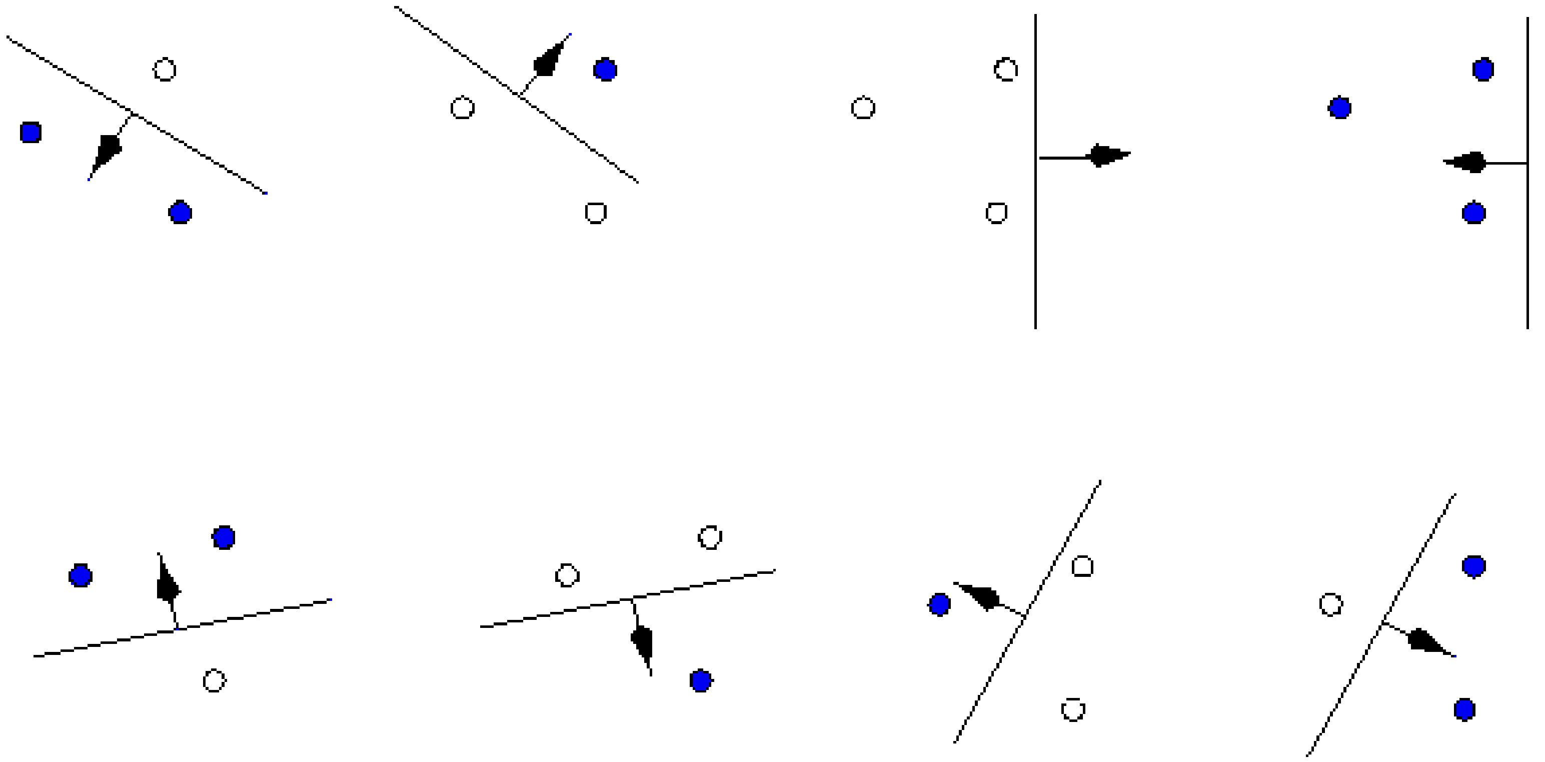
Consider a set of function $f(\mathbf{x},\alpha) \in \{-1,1\}$. A given set of l points can be labeled in 2^l ways. If a member of the set $\{f(\alpha)\}$ can be found which correctly assigns the labels for all labelings, then the set of points is *shattered* by that set of functions

The *VC dimension* of $\{f(\alpha)\}$ is the maximum number of training points that can be shattered by $\{f(\alpha)\}$

For example, the VC dimension of a set of oriented lines in R^2 is three.

In general, the VC dimension of a set of oriented hyperplanes in R^n is $n+1$.

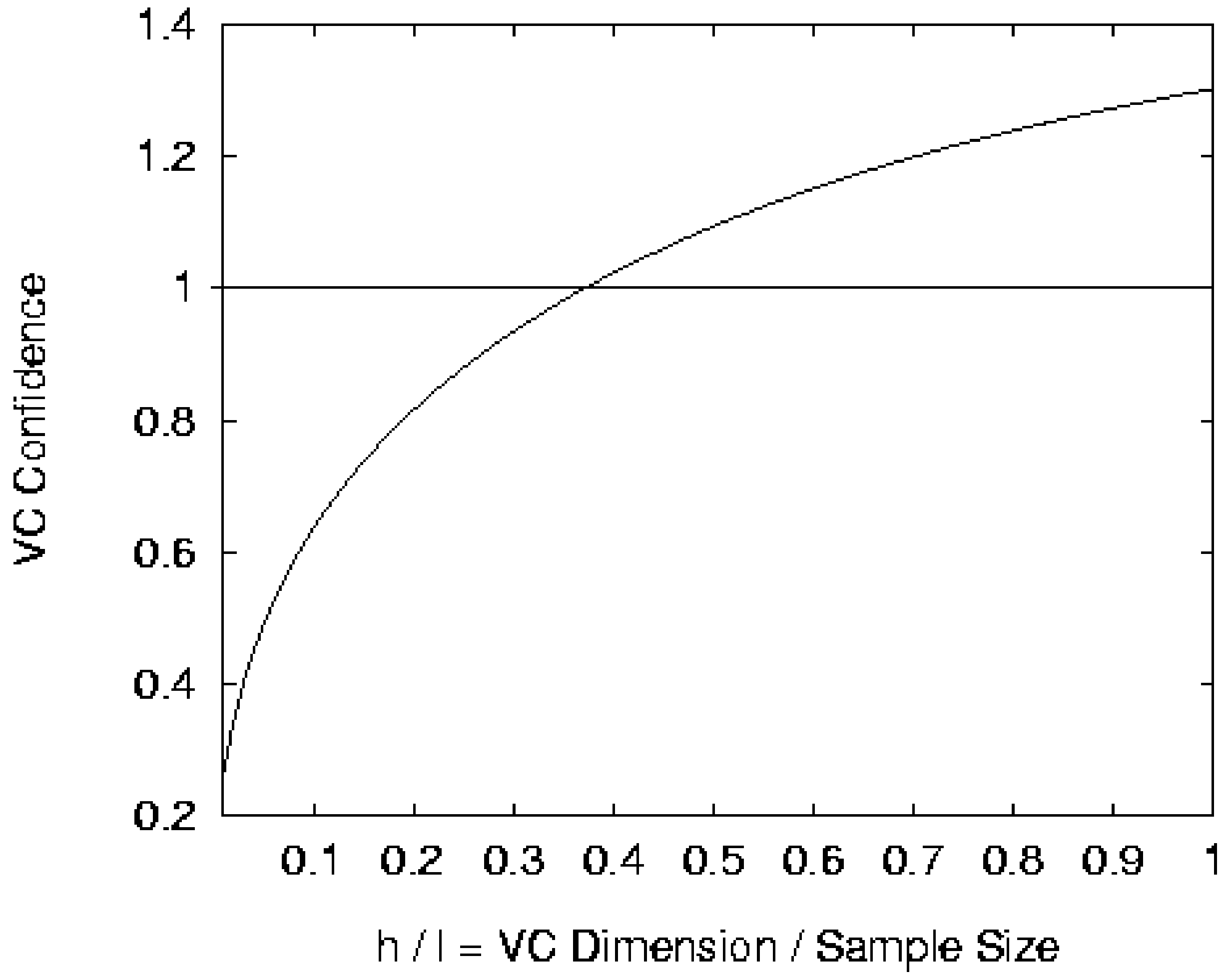
Note: need to find just one set of points.



VC Dimension (cont.)

Note: VC dimension is *not* directly related to number of parameters. Vapnik (1995) has an example with 1 parameter and infinite VC dimension.

$$R(\alpha) \le R_{emp}(\alpha) + \underbrace{\sqrt{\frac{h(\log(2l/h)+1) - \log(\eta/4)}{l}}}_{\text{VC Confidence}}$$



η =0.05 and l=10,000

Amongst machines with zero empirical risk, choose the one with smallest VC dimension

Linear SVM - Separable Case

l observations consisting of a pair: $\boldsymbol{x}_i \in R^d$, i=1,…,l and the associated "label" $y_i \in \{-1,1\}$

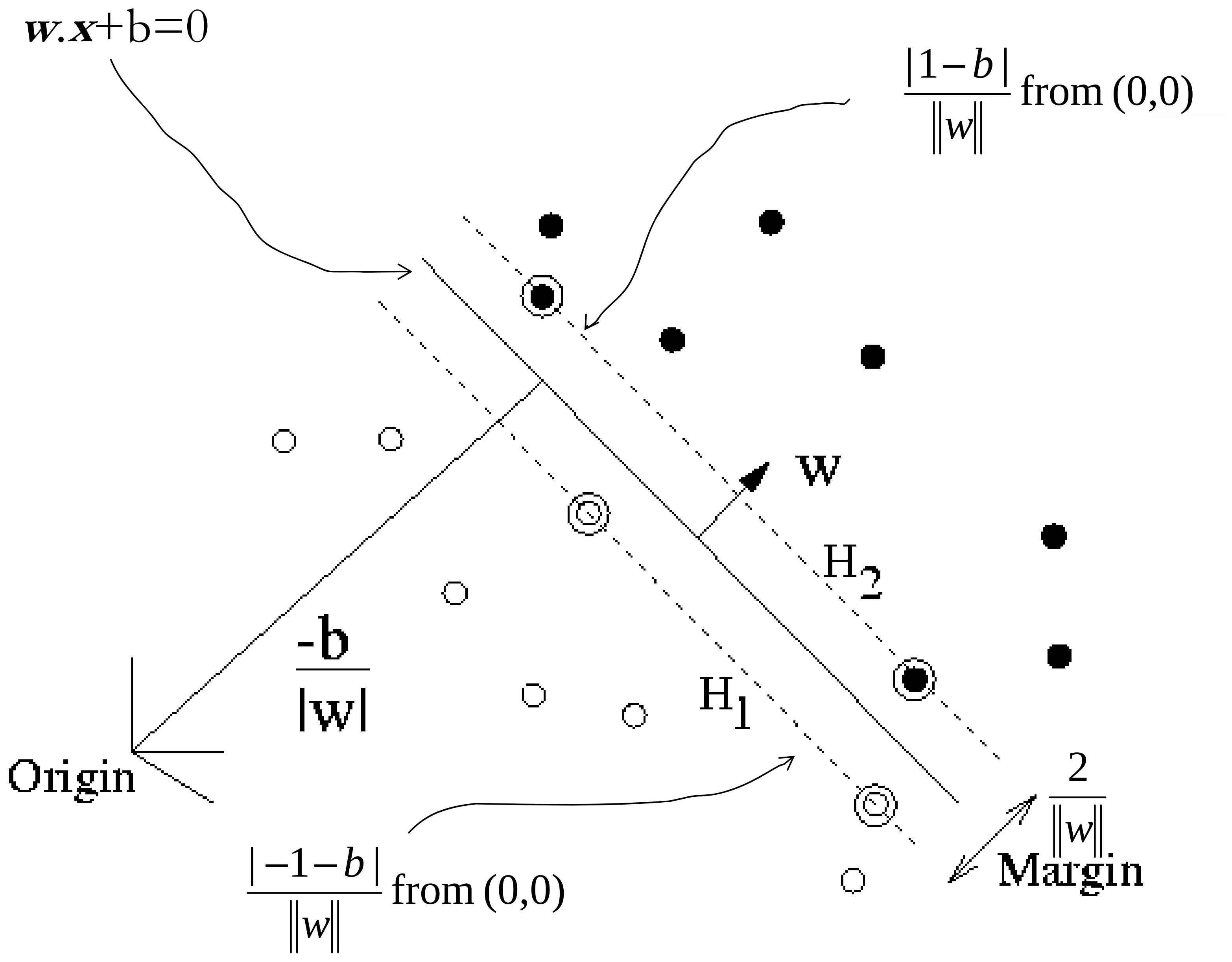
Suppose ∃ a (separating) hyperplane $\boldsymbol{w}.\boldsymbol{x}$+b=0 that separates the positive from the negative examples. That is, all the training examples satisfy:

$$x_i \cdot w + b \geq +1 \text{ when } y_i = +1$$
$$x_i \cdot w + b \leq -1 \text{ when } y_i = -1$$

equivalently:

$$y_i(x_i \cdot w + b) - 1 \geq 0 \forall i$$

Let d_+ (d_-) be the shortest distance from the sep. hyperplane to the closest positive (negative) example. The *margin* of the sep. hyperplane is defined to be $d_+ + d_-$



$\boldsymbol{w}.\boldsymbol{x}+b=0$
$\frac{|1-b|}{\|w\|}$ from (0,0)
w
H_2
$\frac{-b}{|w|}$
H_1
Origin
$\frac{|-1-b|}{\|w\|}$ from (0,0)
$\frac{2}{\|w\|}$
Margin

Linear SVM - Separable Case(II)

SVM finds the hyperplane that minimizes $|\boldsymbol{w}|$ (equiv $|\boldsymbol{w}|^2$) subject to $y_i(w^T x_i + b) - 1 \geq 0$ for $i = 1, \ldots, N$ i.e.

$$\text{minimise } \Phi(w) = 1/2\, w^T w$$

The characteristics of the above QP are: convex quadratic objective function and linear constraints in w.

- The Lagrangian of the QP is

$$L_p = 1/2\, w^T w - \sum_{i=1}^{l} \alpha_i [y_i(w^T x_i + b) - 1]$$

- α_i , $i = 1, \ldots, n$ is the Lagrange multiplier for constraint i.
- Optimality conditions:

$$\delta L_p / \delta w = 0 \text{ and } \delta L_p / \delta b = 0$$

SVM (cont.)

•The two optimality conditions yield the following:

$$w = \sum \alpha_i y_i x_i \quad \textbf{and} \quad \sum \alpha_i y_i = 0$$

Equivalently maximize:

$$L_D = \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j x_i \cdot x_j$$

with respect to the α_i's, subject to $\alpha_i \geq 0$ and this is a convex quadratic programming problem

Note: only depends on dot-products of feature vectors

(Support vectors are points for which equality holds)

Linear SVM - Non-Separable Case

l observations consisting of a pair: $\boldsymbol{x}_i \in \mathrm{R}^d$, i=1,…,*l* and the associated "label" $y_i \in \{-1,1\}$

Introduce positive slack variables ξ_i:

$$x_i \cdot w + b \geq +1 - \xi_i \quad \text{when } y_i = +1$$
$$x_i \cdot w + b \leq -1 + \xi_i \quad \text{when } y_i = -1$$

and modify the objective function to be:

$$\|w\|^2 / 2 + C(\sum \xi_i)^k$$

Non-Linear SVM

$$\boxed{\text{Replace } x_i \cdot x_j \text{ by } k(x_i, x_j)}$$

$$k(x_i, x_j) = (x_i \cdot x_j)^d$$

$$k(x_i, x_j) = \exp(-\|x_i - x_j\|^2 / (2\sigma^2)) \quad \text{radial basis functions}$$

$$k(x_i, x_j) = \tanh(\kappa(x_i \cdot x_j) + \theta) \text{ sigmoid kernels}$$

•Finding VC dimension of machines with different kernels is non-trivial.

•Some (e.g. RBF) have infinite VC dimension but still work well in practice.

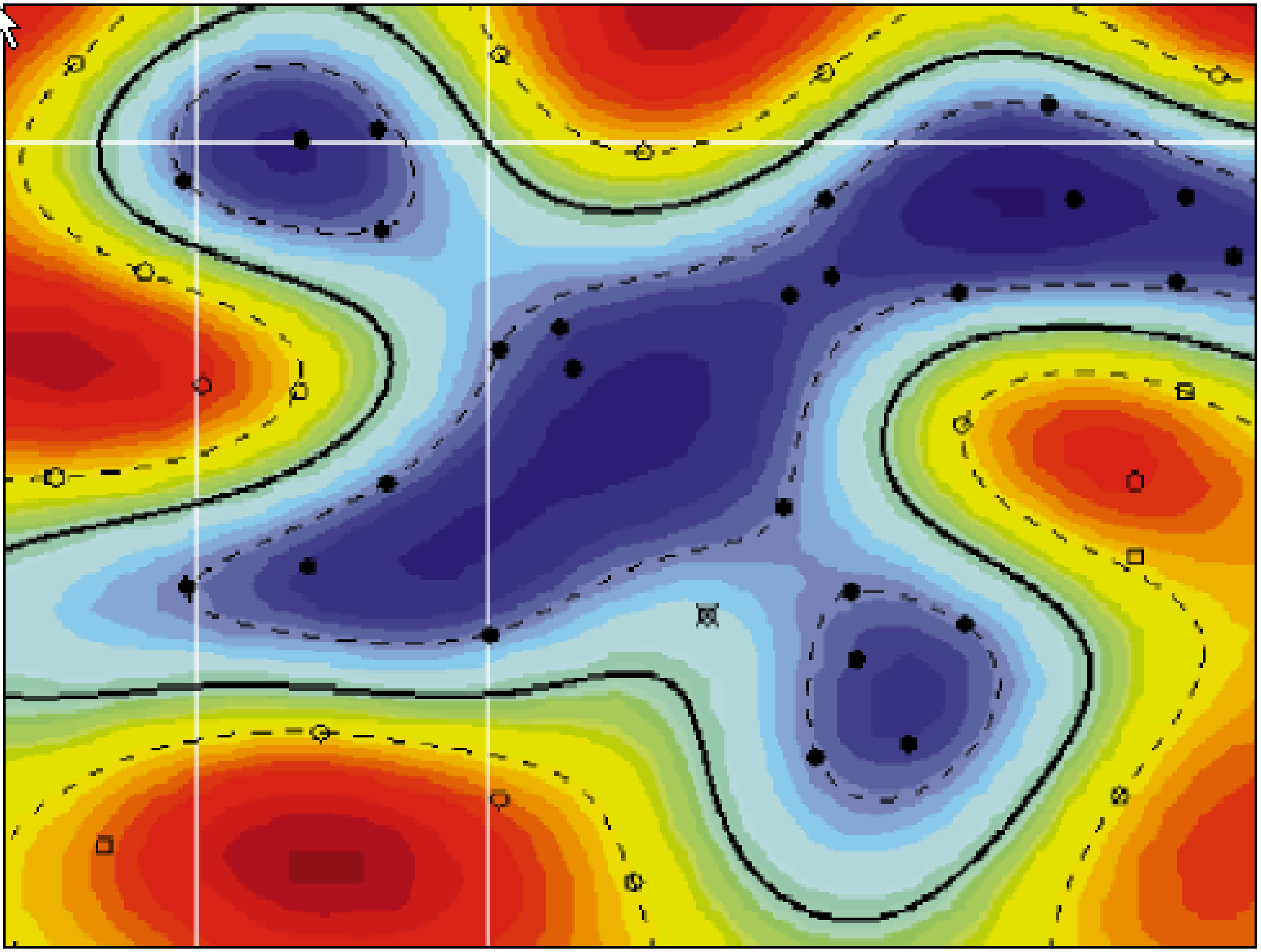


Figure 3. Example of an SV classifier found by using a radial basis function kernel (Equation 8). Circles and disks are two classes of training examples; the solid line is the decision surface; the support vectors found by the algorithm lie on, or between, the dashed lines. Colors code the modulus of the argument $\sum_i v_i \cdot k(\mathbf{x}, \mathbf{x}_i) + b$ of the decision function in Equation 10.

SVM: Issues

•Lots of work on speeding up the quadratic program

•Choice of kernel: doesn’t seem to matter much in practice

•Many open theoretical problems

Outline

- Introduction
- Data Preparation
- Linear Discriminant
- Decision Tree Building
- Support Vector Machine(SVM)
- Bayesian Learning
- Other Classification Methods
- Combining Classifiers
- Validation Methods
- Regression

Bayes Theorem

$$P(h \mid D) = \frac{P(D \mid h)P(h)}{P(D)}$$

- $P(h)$ = prior probability of hypothesis h
- $P(D)$ = prior probability of training data D
- $P(h|D)$ = posterior probability of h given D
- $P(D|h)$ = posterior probability of D given h

Choosing Hypotheses

$$P(h \mid D) = \frac{P(D \mid h)P(h)}{P(D)}$$

- Generally want the most probable hypothesis given the training data *Maximum a posteriori* hypothesis h_{MAP}:

$$\begin{aligned} h_{MAP} &= \arg\max_{h \in H} P(h|D) \\ &= \arg\max_{h \in H} \frac{P(D|h)P(h)}{P(D)} \\ &= \arg\max_{h \in H} P(D|h)P(h) \end{aligned}$$

- If assume $P(h_i) = P(h_j)$ then can further simplify, and choose the *Maximum likelihood* (ML) hypothesis

$$h_{ML} = \arg\max_{h_i \in H} P(D|h_i)$$

Basic Formulas for Probabilities

- *Product Rule*: probability $P(A \wedge B)$ of a conjunction of two events A and B:

$$P(A \wedge B) = P(A \mid B)\, P(B) = P(B \mid A)\, P(A)$$

- *Sum Rule*: probability of a disjunction of two events A and B:

$$P(A \vee B) = P(A) + P(B) - P(A \wedge B)$$

- *Theorem of total probability*: if events $A_1, \dots, A_n$ are mutually exclusive with $\Sigma_{i=1}^{n} P(A_i) = 1$ then

$$P(B) = \sum_{i=1}^{n} P(B|A_i) P(A_i)$$

Most Probable Classification of New Instances

- So far we've sought the most probable *hypothesis* given the data D (i.e., h_{MAP})
- Given new instance x, what is its most probable *classification*?
  - $h_{MAP}(x)$ is **not** the most probable classification!
- Consider:
  - Three possible hypotheses:

    $$P(h_1|D) = .4,\ P(h_2|D) = .3,\ \ P(h_3|D) = .3$$
  - Given new instance x,

    $$h_1(x) = +,\ \ h_2(x) = -,\ \ h_3(x) = -$$
  - What's most probable classification of x?

Bayes Optimal Classifier

- **Bayes optimal classification:**

$$\arg\max_{v_j \in V} \sum_{h_i \in H} P(v_j|h_i)P(h_i|D)$$

- Example:

$$P(h_1|D) = .4,\ P(-|h_1) = 0,\ P(+|h_1) = 1$$
$$P(h_2|D) = .3,\ P(-|h_2) = 1,\ P(+|h_2) = 0$$
$$P(h_3|D) = .3,\ P(-|h_3) = 1,\ P(+|h_3) = 0$$

therefore

$$\sum_{h_i \in H} P(+|h_i)P(h_i|D) = .4$$
$$\sum_{h_i \in H} P(-|h_i)P(h_i|D) = .6$$

and

$$\arg\max_{v_j \in V} \sum_{h_i \in H} P(v_j|h_i)P(h_i|D) = -$$

Gibbs Classifier

- Bayes optimal classifier provides best result, but can be expensive if many hypotheses.
- Gibbs algorithm:
  1. Choose one hypothesis at random, according to $P(h|D)$
  2. Use this to classify new instance
- Surprising fact: Assume target concepts are drawn at random from *H* according to priors on *H*. Then:

$$E[error_{Gibbs}] \leq 2E[error_{BayesOptional}]$$

- Suppose correct, uniform prior distribution over *H*, then
  - Pick any hypothesis from *VS*, with uniform probability
  - Its expected error no worse than twice Bayes optimal

Naive Bayes Classifier (I)

- Along with decision trees, neural networks, nearest nbr, one of the most practical learning methods.
- When to use
  - Moderate or large training set available
  - Attributes that describe instances are conditionally independent given classification
- Successful applications:
  - Diagnosis
  - Classifying text documents

Naive Bayes Classifier (II)

- Assume target function $f: X \rightarrow V$, where each instance x described by attributes $<a_1, a_2 \ldots a_n>$.
- Most probable value of $f(x)$ is:

$$v_{MAP} = \operatorname*{argmax}_{v_j \in V} P(v_j | a_1, a_2 \ldots a_n)$$

$$\begin{aligned} v_{MAP} &= \operatorname*{argmax}_{v_j \in V} \frac{P(a_1, a_2 \ldots a_n | v_j) P(v_j)}{P(a_1, a_2 \ldots a_n)} \\ &= \operatorname*{argmax}_{v_j \in V} P(a_1, a_2 \ldots a_n | v_j) P(v_j) \end{aligned}$$

Naive Bayes assumption: $P(a_1, a_2 \ldots a_n | v_j) = \prod_i P(a_i | v_j)$

which gives

Naive Bayes classifier: $v_{NB} = \operatorname*{argmax}_{v_j \in V} P(v_j) \prod_i P(a_i | v_j)$

Naive Bayes Algorithm

- Naive Bayes Learn(*examples*)

  For each target value v_j

  $$P(v_j) \leftarrow \text{estimate } \hat{P}(v_j)$$

  For each attribute value a_i of each attribute a

  $$P(\hat{a}_i \mid v_j) \leftarrow \text{estimate } \hat{P}(a_i \mid v_j)$$

- Classify New Instance(x)

$$v_{NB} = \operatorname*{argmax}_{v_j \in V} \hat{P}(v_j) \prod_{a_i \in x} \hat{P}(a_i|v_j)$$

Naive Bayes: Example

- Consider *PlayTennis* again, and new instance

  <*Outlk* = *sun*, *Temp* = *cool*, *Humid* = *high*, *Wind* = *strong*>

- Want to compute: $v_{NB} = \operatorname*{argmax}_{v_j \in V} P(v_j) \prod_i P(a_i|v_j)$

- $P(y)\ P(sun|y)\ P(cool|y)\ P(high|y)\ P(strong|y) = .005$

  $P(n)\ P(sun|n)\ P(cool|n)\ P(high|n)\ P(strong|n) = .021$

$$\rightarrow v_{NB} = n$$

Example

Outlook	***Temp***	***Humid***	***Wind***	***PlayTennis***
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Overcast	Hot	High	Weak	Yes
Rain	Mild	High	Weak	Yes
Rain	Cool	Normal	Weak	Yes
Rain	Cool	Normal	Strong	No
Overcast	Cool	Normal	Strong	Yes
Sunny	Mild	High	Weak	No
Sunny	Cool	Normal	Weak	Yes
Rain	Mild	Normal	Weak	Yes
Sunny	Mild	Normal	Strong	Yes
Overcast	Mild	High	Strong	Yes
Overcast	Hot	Normal	Weak	Yes
Rain	Mild	High	Strong	No

Naive Bayes: Subtleties (I)

1. Conditional independence assumption is often violated $P(a_1, a_2 \ldots a_n | v_j) = \prod_i P(a_i | v_j)$

- ...but it works surprisingly well anyway. Note don't need estimated posteriors $\hat{P}(v_j | x)$ to be correct; need only that

$$\operatorname*{argmax}_{v_j \in V} \hat{P}(v_j) \prod_i \hat{P}(a_i | v_j) = \operatorname*{argmax}_{v_j \in V} P(v_j) P(a_1 \ldots, a_n | v_j)$$

- see [Domingos & Pazzani, 1996] for analysis
- Naive Bayes posteriors often unrealistically close to 1 or 0

Pedro Domingos, Michael Pazzan. **Beyond Independence: Conditions for the Optimality of the Simple Bayesian Classifier (1996).**

Naive Bayes: Subtleties (II)

2. What if none of the training instances with target value v_j have attribute value a_i? Then

$$\hat{P}(a_i|v_j) = 0, \text{ and...}$$
$$\hat{P}(v_j)\prod_i \hat{P}(a_i|v_j) = 0$$

Typical solution is Bayesian estimate for $\hat{P}(a_i|v_j)$

$$\hat{P}(a_i|v_j) \leftarrow \frac{n_c + mp}{n + m}$$

where

- n is number of training examples for which $v = v_i$,
- n_c number of examples for which $v = v_j$ and $a = a_i$
- p is prior estimate for $\hat{P}(a_i|v_j)$
- m is weight given to prior(i.e. number of "virtual" examples)

Bayesian Belief Networks

Interesting because:

- Naive Bayes assumption of conditional independence too restrictive
- But it's intractable without some such assumptions...
- Bayesian Belief networks describe conditional independence among *subsets* of variables

$\rightarrow$ allows combining prior knowledge about (in)dependencies among variables with observed training data (also called Bayes Nets)

Conditional Independence

- **Definition:** X is *conditionally independent* of Y given Z if the probability distribution governing X is independent of the value of Y given the value of Z; that is, if

$$(\forall x_i, y_j, z_k)\ P(X = x_i | Y = y_j, Z = z_k) = P(X = x_i | Z = z_k)$$

  more compactly, we write

$$P(X | Y, Z) = P(X | Z)$$

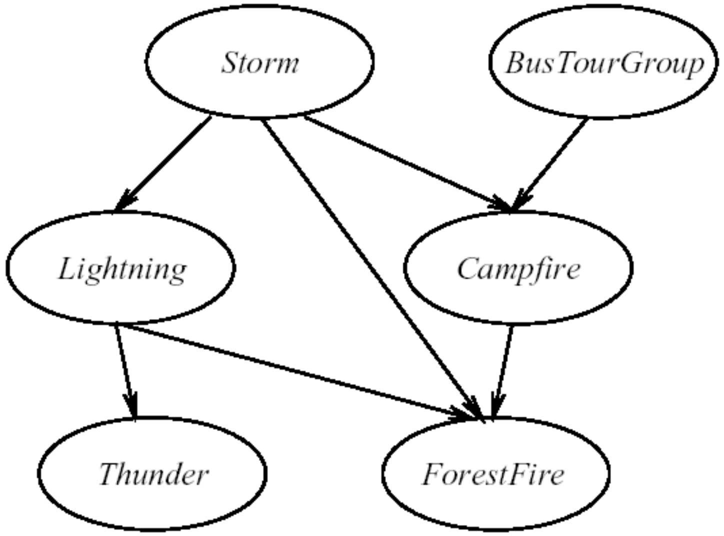
- Example: *Thunder* is conditionally independent of *Rain*, given *Lightning*

$$P(Thunder | Rain, Lightning) = P(Thunder | Lightning)$$

- Naive Bayes uses cond. indep. to justify

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

Bayesian Belief Network (I)



	S,B	$S,\neg B$	$\neg S,B$	$\neg S,\neg B$
C	0.4	0.1	0.8	0.2
$\neg C$	0.6	0.9	0.2	0.8

- Network represents a set of conditional independence assertions:
    - Each node is asserted to be conditionally independent of its nondescendants, given its immediate predecessors.
    - Directed acyclic graph

Bayesian Belief Network (II)

- Represents joint probability distribution over all variables
  - e.g., *P*(*Storm*, *BusTourGroup*, . . . , *ForestFire*)
  - in general,

$$P(y_1, \ldots, y_n) = \prod_{i=1}^{n} P(y_i | Parents(Y_i))$$

    where *Parents*(Y_i) denotes immediate predecessors of Y_i in graph
  - so, joint distribution is fully defined by graph, plus the P(y_i|*Parents*(Y_i))

Inference in Bayesian Networks

- How can one infer the (probabilities of) values of one or more network variables, given observed values of others?
  - Bayes net contains all information needed for this inference
  - If only one variable with unknown value, easy to infer it
- In practice, can succeed in many cases
  - Exact inference methods work well for some network structures
  - Monte Carlo methods "simulate" the network randomly to calculate approximate solutions

Learning of Bayesian Networks

- Several variants of this mining task
  - Network structure might be *known* or *unknown*
  - Training examples might provide values of *all* network variables, or just *some*

- If structure known and observe all variables
  - Then it's easy as training a Naive Bayes classifier

Learning Bayes Nets

- Suppose structure known, variables partially observable
- e.g., observe *ForestFire*, *Storm*, *BusTourGroup*, *Thunder*, but not *Lightning*, *Campfire*...
  - In fact, can learn network conditional probability tables using gradient ascent!
  - Converge to network h that (locally) maximizes $P(D|h)$

Gradient Ascent for Bayes Nets

- Let w_{ijk} denote one entry in the conditional probability table for variable Y_i in the network

$$w_{ijk} = P(Y_i = y_{ij} | Parents(Y_i) = \text{the list } u_{ik} \text{ of values})$$

- e.g., if $Y_i = Campfire$, then u_{ik} might be

$$< Storm = \mathrm{T},\ BusTourGroup = F >$$

- Perform gradient ascent by repeatedly
  1. update all w_{ijk} using training data D

$$w_{ijk} \leftarrow w_{ijk} + \eta \sum_{d \in D} \frac{P_h(y_{ij}, u_{ik} | d)}{w_{ijk}}$$

  2. then, renormalize the weight w_{ijk} to assure
     - $\Sigma_j\, w_{ijk} = 1$ – $0 \le w_{ijk} \le 1$

More on Learning Bayes Nets

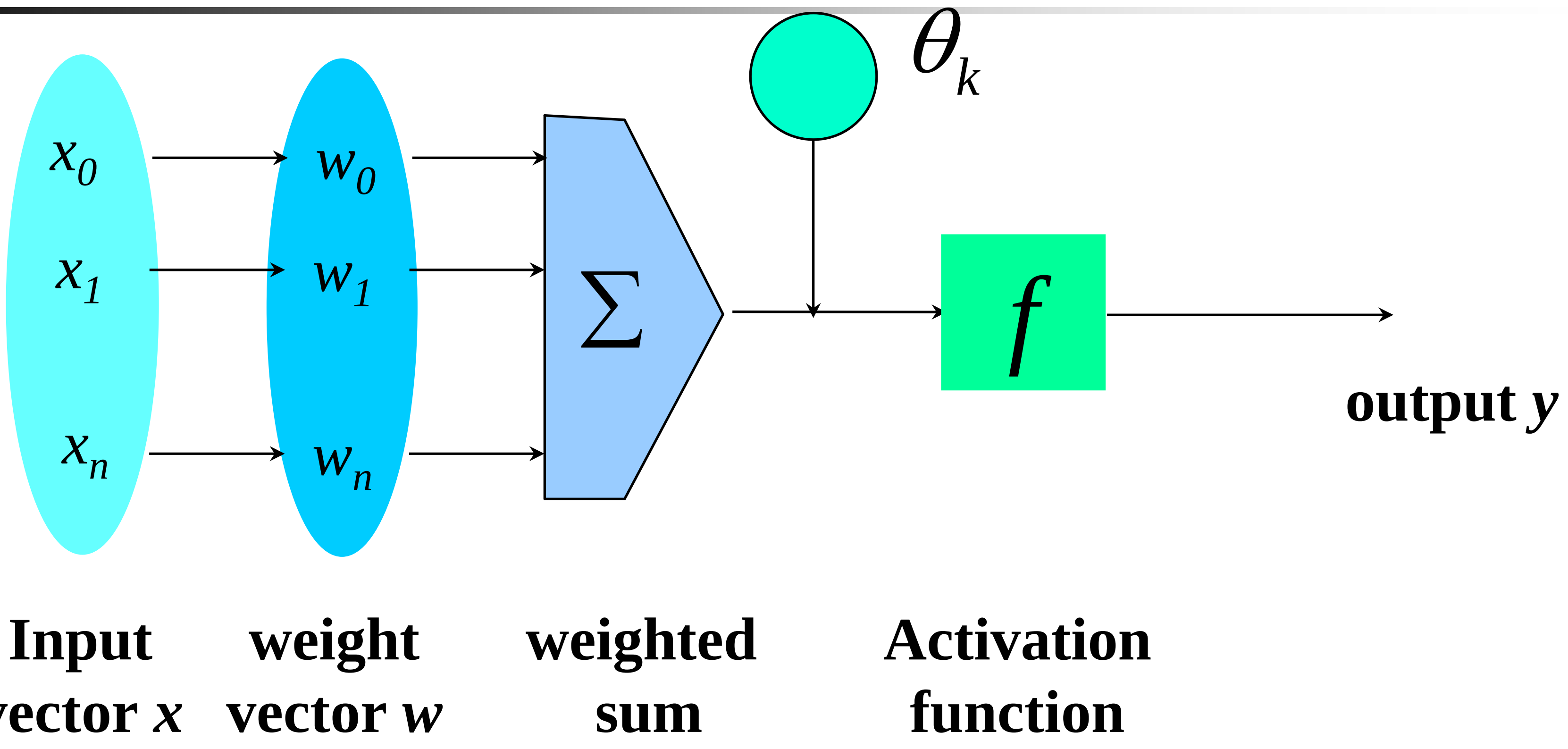
- EM algorithm can also be used. Repeatedly:
  1. Calculate probabilities of unobserved variables, assuming *h*
  2. Calculate new w_{ijk} to maximize E [ln $P(D|h)$] where D now includes both observed and (calculated probabilities of) unobserved variables

- When structure unknown...
  - Algorithms use greedy search to add/substract edges and nodes
  - Active research topic

Outline

- Introduction
- Data Preparation
- Linear Discriminant
- Decision Tree Building
- Support Vector Machine(SVM)
- Bayesian Learning
- Other Classification Methods
- Combining Classifiers
- Validation Methods
- Regression

A Neuron

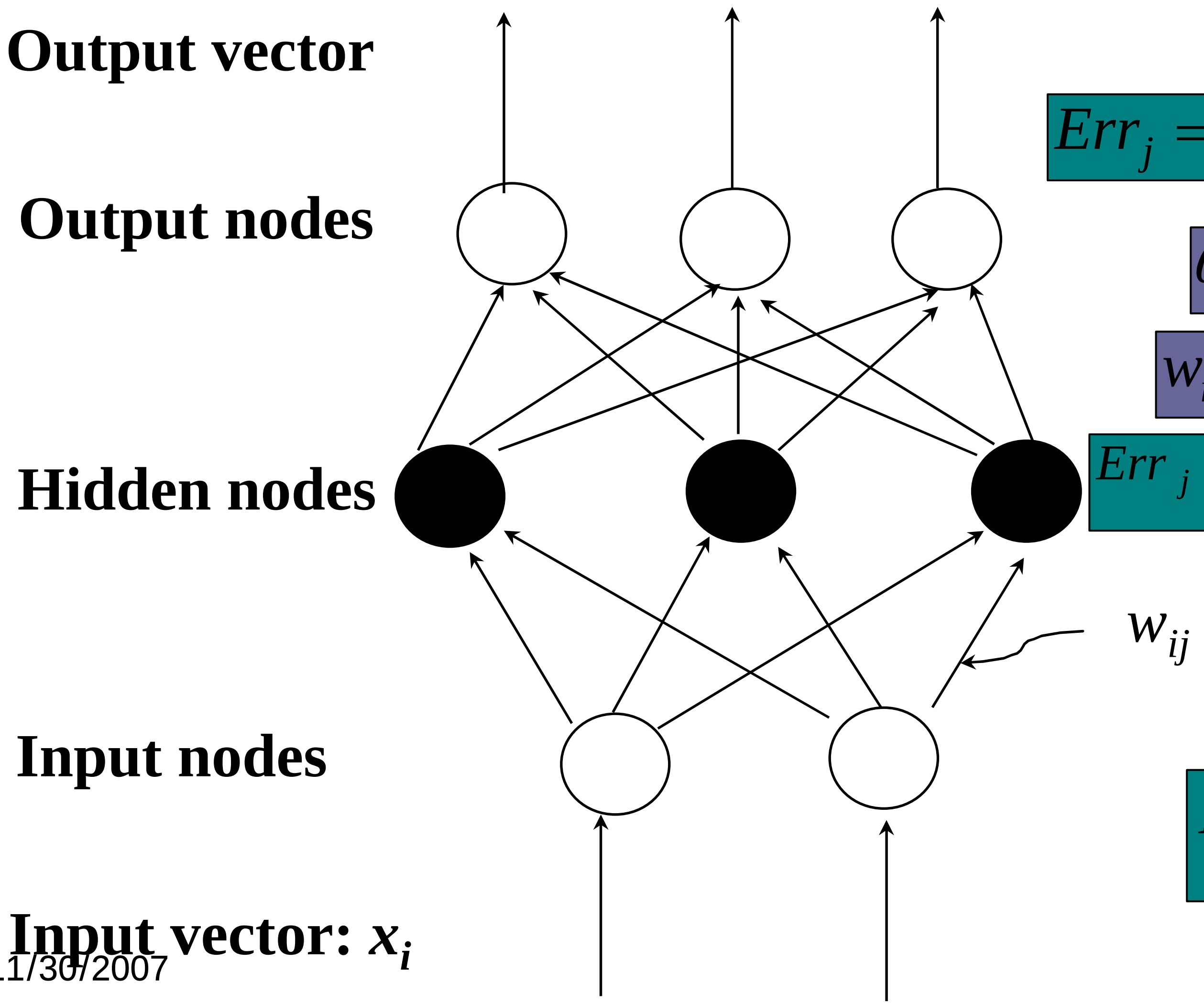


- The *n*-dimensional input vector *x* is mapped into variable *y* by means of the scalar product and a nonlinear function mapping

Network Training

- The ultimate objective of training
  - obtain a set of weights that makes almost all the tuples in the training data classified correctly
- Steps
  - Initialize weights with random values
  - Feed the input tuples into the network one by one
  - For each unit
    - Compute the net input to the unit as a linear combination of all the inputs to the unit
    - Compute the output value using the activation function
    - Compute the error
    - Update the weights and the bias

Multi-Layer Perceptron



$$Err_j = O_j(1-O_j)(T_j - O_j)$$

$$\theta_j = \theta_j + (l)Err_j$$

$$w_{ij} = w_{ij} + (l)Err_j O_i$$

$$Err_j = O_j(1-O_j)\sum_k Err_k w_{jk}$$

$$O_j = \frac{1}{1+e^{-I_j}}$$

$$I_j = \sum_i w_{ij} O_i + \theta_j$$

Association-Based Classification

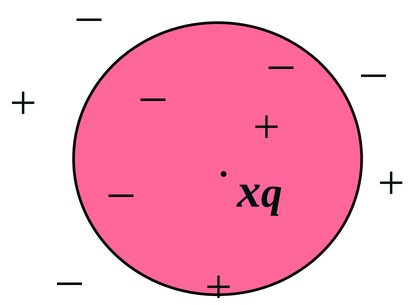
- Several methods for association-based classification
  - Associative classification(CBA): (Liu et al'98)
    - It mines high support and high confidence rules in the form of "cond_set => y", where y is a class label
  - CAEP (Classification by aggregating emerging patterns) (Dong et al'99)
    - Emerging patterns (EPs): the itemsets whose support increases significantly from one class to another
    - Mine EPss based on minimum support and growth rate

Instance-based Methods

- Instance-based learning:
  - Store training examples and delay the processing ("lazy evaluation") until a new instance must be classified
- Typical approaches
  - K-nearest neighbor approach
    - Instances represented as points in an Euclidean space
  - Case-based reasoning
    - Uses symbolic representations and knowledge-based inference

The K-nearest Neighbor Algorithm (KNN)

- Instances are points in an n-D space
- The nearest neighbor in the Euclidean distance
- Discrete-/real-valued target functions
- Return the most common value among the k training examples nearest to the query point



Case-based Reasoning

- Lazy evaluation + analysis of similar instances
- Methodology
  - Instances represented by rich symbolic descriptions (e.g., function graphs)
  - Combine multiple retrieved cases
  - Tight coupling between case retrieval, knowledge-based reasoning, and problem solving

Lazy vs. Eager Learning

- Efficiency: lazy learning uses less training time but more predicting time
- Accuracy
  - Lazy method effectively uses a richer hypothesis space
  - Eager: must commit to a single hypothesis that covers the entire instance space

Outline

- Introduction
- Data Preparation
- Linear Discriminant
- Decision Tree Building
- Support Vector Machine(SVM)
- Bayesian Learning
- Other Classification Methods
- Combining Classifiers
- Validation Methods
- Regression

Bagging and Boosting

- General idea
- Training data $\xrightarrow{\text{Classification method (CM)}}$ **Classifier C**
- Altered Training data $\xrightarrow{\text{CM}}$ **Classifier C1**
- Altered Training data $\xrightarrow{\text{CM}}$ **Classifier C2**
- ....
- Aggregation .... $\xrightarrow{}$ **Classifier C***

Bagging

- Given a set S of s samples, generate a sequence of k independent bootstrap training sets
- Construct a sequence of classifiers C1,C2,…,Ck by using the same classification algorithm
- To classify an unknown sample X, let each classifier predict or vote
- The bagged classifier C* counts the votes and assigns X to the class with the “most” votes

Boosting Technique

- Assign every example an equal weight 1/N
- For t = 1, 2, ..., T Do
  - Obtain a classifier C(t) under w(t)
  - Calculate the error of C(t) and re-weight the examples based on the errors. Samples incorrectly predicted have bigger weight
- Output a weighted sum of all the classifiers, with each classifier weighted according to its accuracy on the training set

Outline

- Introduction
- Data Preparation
- Linear Discriminant
- Decision Tree Building
- Support Vector Machine(SVM)
- Bayesian Learning
- Other Classification Methods
- Combining Classifiers
- Validation Methods
- Regression

Holdout

- Randomly partition the given data into a training set and a test set
  - Typically, 2/3 are in the training set and 1/3 in the test set
  - Random subsampling is also feasible

K-fold Cross-validation

- Randomly partition the given data into k folds: mutually exclusive subsets with approximately equal size
- In iteration j, use S_j as the test set and the remaining folds as training set
  - Stratified cross-validation: each fold has approximately equal class distribution
  - Stratified 10-fold cross-validation
- Bootstrapping: sample the test set with replacement

Sensitivity and Specificity

- Sensitivity = t_pos / pos
  - The percentage of positive samples correctly classified
- Specificity = t_neg / neg
  - The percentage of negative samples correctly classified

Precision

- Precision = t_pos / (t_pos + f_pos)
  - The percentage of samples classified positive are actually positive

$$accuracy = \frac{sensitivity \times pos + specificity \times neg}{pos + neg}$$

Outline

- Introduction
- Data Preparation
- Linear Discriminant
- Decision Tree Building
- Support Vector Machine(SVM)
- Bayesian Learning
- Other Classification Methods
- Combining Classifiers
- Validation Methods
- Regression

What Is Regression

- Regression is similar to classification
  - Construct a model
  - Use the model to predict unknown value
    - Linear and multiple regression, non-linear regression
- Regression models continuous-valued functions

Regression Analysis and Log-Linear Models

- Linear regression: $Y = \beta_0 + \beta_1 X$
  - Two parameters , β_0 and β_1 specify the line and are to be estimated by using the data at hand.
  - using the least squares criterion
- Multiple regression: $Y = b_0 + b_1 X_1 + b_2 X_2$.
  - Many nonlinear functions can be transformed into the above.
- Log-linear models:
  - The multi-way table of joint probabilities is approximated by a product of lower-order tables.
  - Probability: $p(a, b, c, d) = \alpha ab\ \beta ac \chi ad\ \delta bcd$

Linear Regression Model

- **Relationship Between Variables Is a <u>Linear Function</u>**

Y-Intercept

Slope

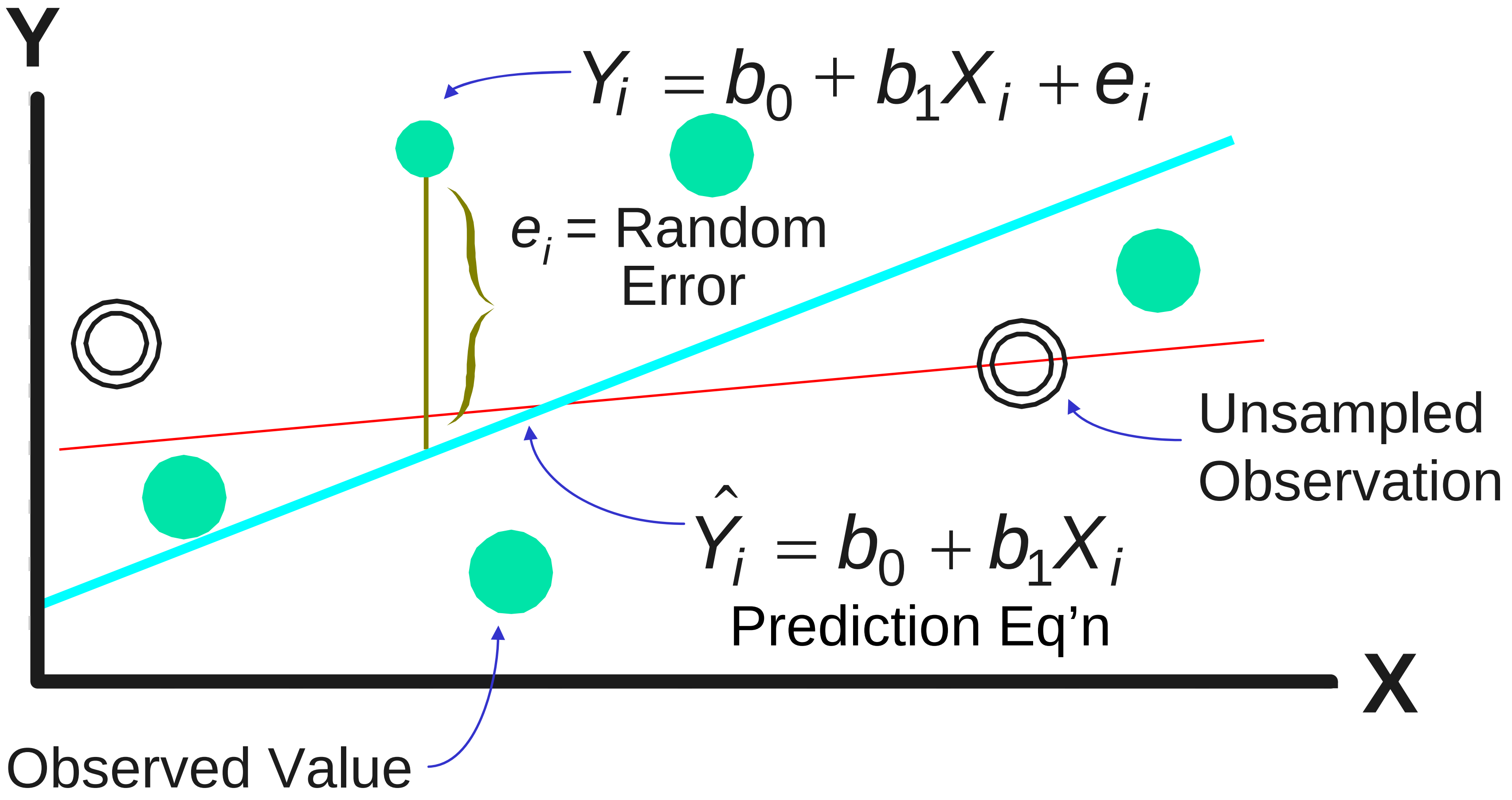
Residue
s

$$Y_i = \beta_0 + \beta_1 X_i + \varepsilon_i$$

Dependent
Variable

Independent
Variable

Linear Regression Model



Model Estimates

- b_0 estimates β_0 - “Y - Intercept”
  - Expected value of Y when X = 0
- b_1 estimates β_1 - “Slope”
  - Expected change in Y per unit change in X
- Valid only over - “Relevant Range”
  - Interpolate - Do Not Extrapolate!!

Formula for b_0 and b_1

$$b_0 = \frac{(\Sigma y_i)\,(\Sigma x_i^2) - (\Sigma x_i)\ \ (\Sigma x_i y_i)}{N^*(\Sigma x_i^2) - (\Sigma x_i)^2}$$ **(y-intercept)**

$$b_1 = \frac{N^*(\Sigma x_i y_i) - (\Sigma x_i)\,(\Sigma y_i)}{N^*(\Sigma x_i^2) - (\Sigma x_i)^2}$$ **(slope)**

N=number of points

How to get those formulae ?

- Score Function
  - Minimize $SSE = \sum [y_i - (b_0 + b_1 x_i)]^2$
- Partial Derivation

$$\frac{\partial SSE}{\partial b_0} = -\sum 2[y_i - (b_0 + b_1 x_i)] = 0$$

$$\frac{\partial SSE}{\partial b_1} = \sum 2[y_i - (b_0 + b_1 x_i)](-x_i) = 0$$

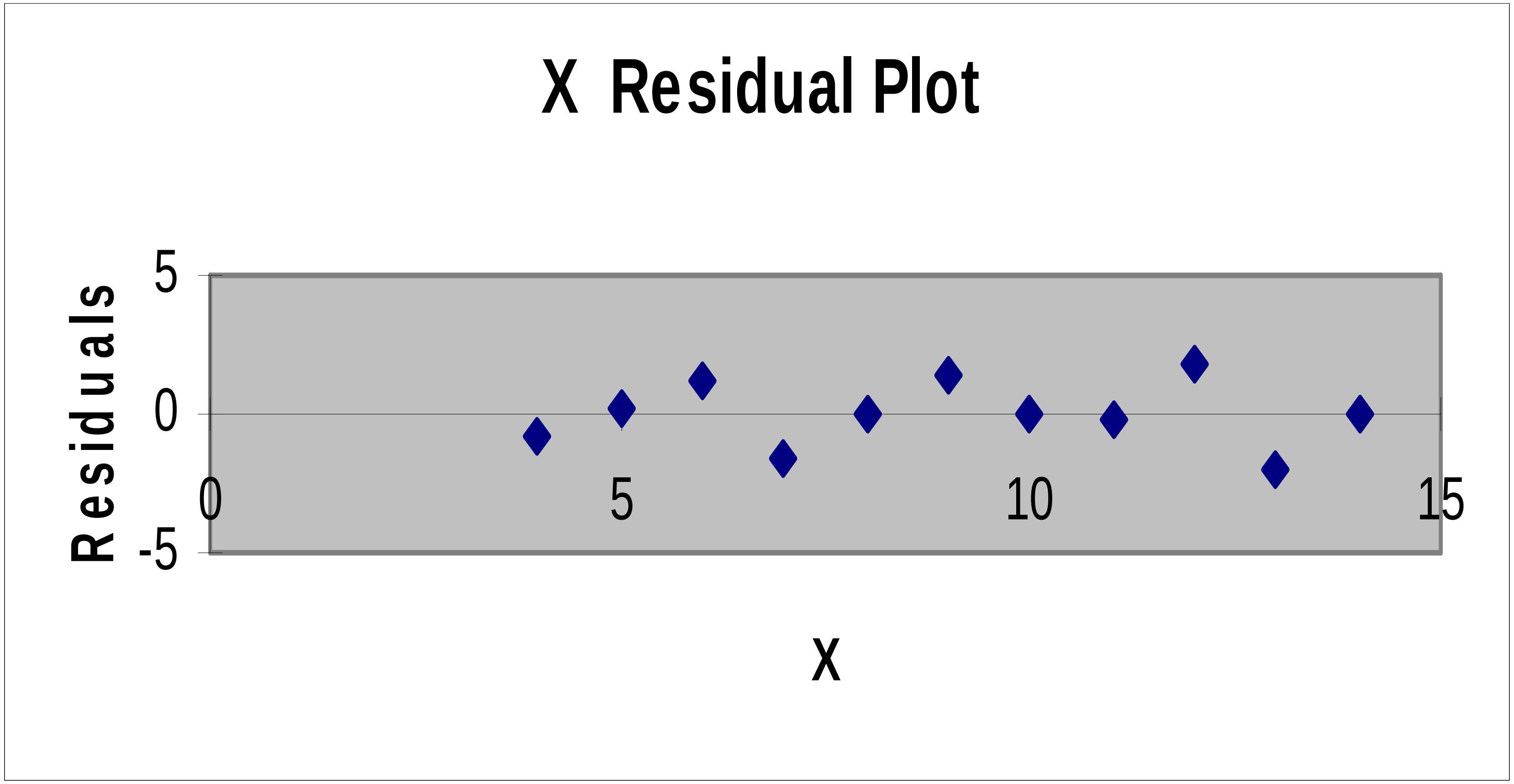
Locally Weighted Regression

- Construct an explicit approximation to *f* over a local region surrounding query instance x_q.
- Locally weighted linear regression:
  - The target function *f* is approximated near x_q using the linear function:
$$\hat{f}(x) = w_0 + w_1 a_1(x) + \cdots + w_n a_n(x)$$
  - minimize the squared error: distance-decreasing weight *K*
$$E(x_q) \equiv \frac{1}{2} \sum_{x \in knn(x_q)} (f(x) - \hat{f}(x))^2 K(d(x_q, x))$$
  - the gradient descent training rule:
$$\Delta w_j \equiv \eta \sum_{x \in knn(x_q)} K(d(x_q, x))((f(x) - \hat{f}(x))a_j(x)$$
- In most cases, the target function is approximated by a constant, linear, or quadratic function.

Residual Analysis

- 1.Graphical Analysis of Residuals ("errors")
  - Residuals = Difference between actual Y_i & predicted Y_i
  - Plot residuals vs. X_i values
- 2.Purpose
  - Examine functional form (linear vs. non-linear)
  - Test independence of errors

Residual Analysis



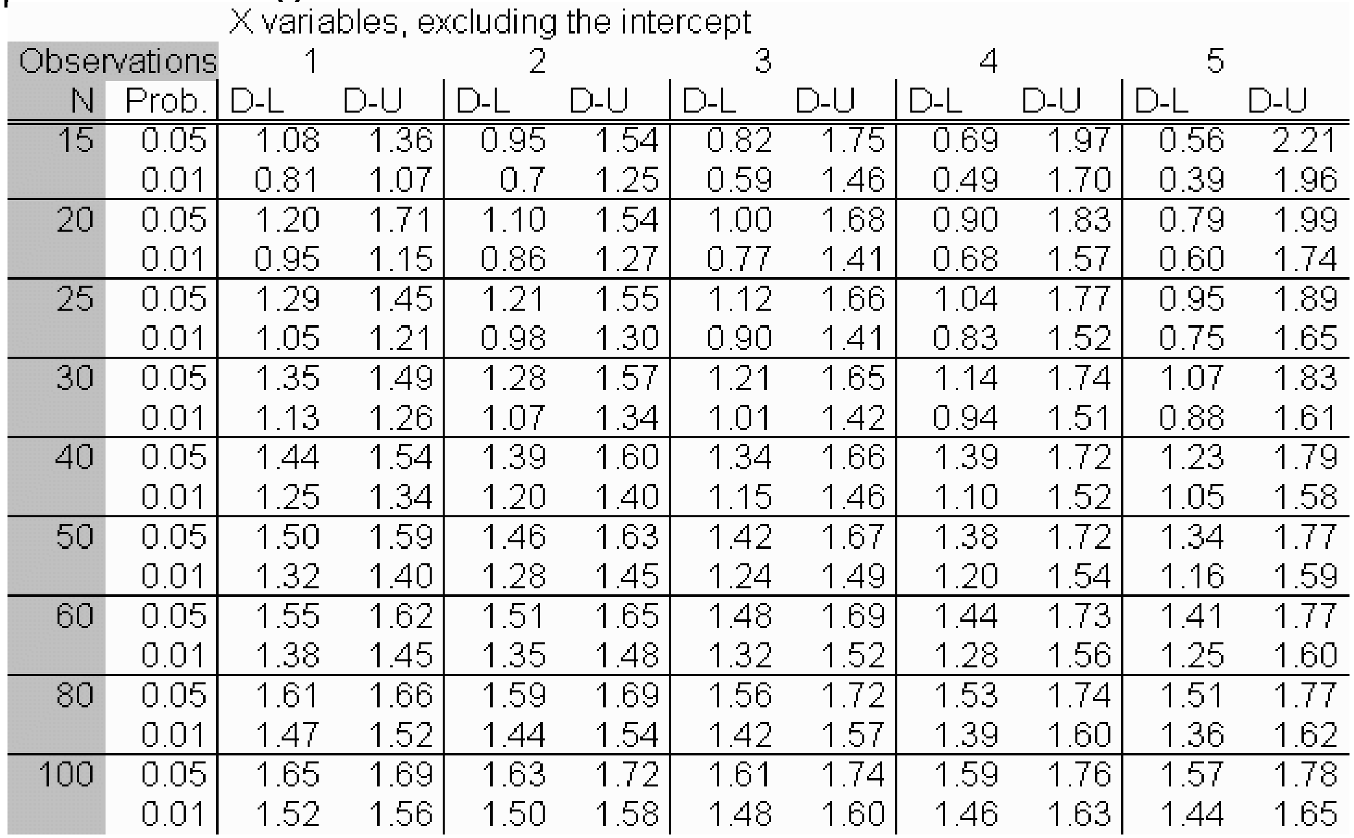
Durbin-Watson Procedure

- 1. Used to Detect Autocorrelation
  - Residuals in one time period are related to residuals in another period
  - Violation of independence assumption
- 2. Durbin-Watson Test Statistic

$$D = \frac{\sum_{i=2}^{n} (e_i - e_{i-1})^2}{\sum_{i=1}^{n} e_i^2}$$

Durbin-Watson Rules

- For given α, n, & p:
- If $D < d_L$, then auto-correlation exists
- If $D > d_U$, then no auto-correlation exists
- If $d_L < D < d_U$, then no definite conclusion



		X variables, excluding the intercept									
Observations		1		2		3		4		5	
N	Prob.	D-L	D-U	D-L	D-U	D-L	D-U	D-L	D-U	D-L	D-U
15	0.05	1.08	1.36	0.95	1.54	0.82	1.75	0.69	1.97	0.56	2.21
	0.01	0.81	1.07	0.7	1.25	0.59	1.46	0.49	1.70	0.39	1.96
20	0.05	1.20	1.71	1.10	1.54	1.00	1.68	0.90	1.83	0.79	1.99
	0.01	0.95	1.15	0.86	1.27	0.77	1.41	0.68	1.57	0.60	1.74
25	0.05	1.29	1.45	1.21	1.55	1.12	1.66	1.04	1.77	0.95	1.89
	0.01	1.05	1.21	0.98	1.30	0.90	1.41	0.83	1.52	0.75	1.65
30	0.05	1.35	1.49	1.28	1.57	1.21	1.65	1.14	1.74	1.07	1.83
	0.01	1.13	1.26	1.07	1.34	1.01	1.42	0.94	1.51	0.88	1.61
40	0.05	1.44	1.54	1.39	1.60	1.34	1.66	1.39	1.72	1.23	1.79
	0.01	1.25	1.34	1.20	1.40	1.15	1.46	1.10	1.52	1.05	1.58
50	0.05	1.50	1.59	1.46	1.63	1.42	1.67	1.38	1.72	1.34	1.77
	0.01	1.32	1.40	1.28	1.45	1.24	1.49	1.20	1.54	1.16	1.59
60	0.05	1.55	1.62	1.51	1.65	1.48	1.69	1.44	1.73	1.41	1.77
	0.01	1.38	1.45	1.35	1.48	1.32	1.52	1.28	1.56	1.25	1.60
80	0.05	1.61	1.66	1.59	1.69	1.56	1.72	1.53	1.74	1.51	1.77
	0.01	1.47	1.52	1.44	1.54	1.42	1.57	1.39	1.60	1.36	1.62
100	0.05	1.65	1.69	1.63	1.72	1.61	1.74	1.59	1.76	1.57	1.78
	0.01	1.52	1.56	1.50	1.58	1.48	1.60	1.46	1.63	1.44	1.65

Essential Readings

- "Data Mining: Concepts and Techniques" Jiawei Han and Micheline Kamber. Chapter 6
- "Principles of Data Mining", David Hand, Heikki Mannila and Padhraic Smyth. Chapter 10.1 - 10.4.
- Johannes Gehrke, "Scalable Classification Tree Construction.".
- "Langrange Multipliers without Permanent Scarring", Dan Klein.
- "A Tutorial on Support Vector Machines for Pattern Recognition", Christopher J.C. Burges.
- "Machine Learning", Tom M. Mitchell Chapter 6.1, 6.2, 6.7-6.11

Optional Readings

- Pedro Domingos and Michael Pazzani. **Beyond independence: Conditions for the optimality of the simple bayesian classifier**. In Proceedings of the 13th International Conference on Machine Learning, pages 105-112, 1996.

- Russell, S., Binder, J., Koller, D., and Kanazawa, K. (1995). **Local learning in probabilistic networks with hidden variables**. In Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence. San Mateo, CA: Morgan Kaufmann. 1146--1152.

Reference

- http://www.cs.otago.ac.nz/cosc453/student_tutorials/principal_components.pdf
- T. M. Mitchell. Machine Learning. McGraw Hill, 1997. Chapter 3
- J. R. Quinlan. Induction of decision trees. Machine Learning, 1:81-106, 1986
- Dash, M. and Liu, H. (1997). *Feature Selection for Classification*. Intelligent Data Analysis, 1 (3): 131-156. Elsevier Science Inc.
- Shafer, J. C.; Agrawal, R.; and Metha, M. SPRINT: A scalable parallel classifier for data mining. In VLDB'96
- J. Gehrke, R. Ramakrishnan and V. Ganti, Rainforest - a framework for fast decision tree construction of large datasets, Proc. VLDB'98.
- M. Mehta, R. Agrawal, and J. Rissanen. SLIQ : A fast scalable classifier for data mining. (EDBT'96), Avignon, France, March 1996.
- R. Rastogi and K. Shim. Public: A decision tree classifer that integrates building and pruning. In Proc. 1998 Int. Conf. Very Large Data Bases, 404-415, New York, NY, August 1998.