# Classification
# Lecture 2: Methods

**Jing Gao**
**SUNY Buffalo**

# Outline

- **Basics**
  - Problem, goal, evaluation
- **Methods**
  - Nearest Neighbor
  - Decision Tree
  - Naïve Bayes
  - Rule-based Classification
  - Logistic Regression
  - Support Vector Machines
  - Ensemble methods
  - ………
- **Advanced topics**
  - Multi-view Learning
  - Semi-supervised Learning
  - Transfer Learning
  - ……

# Bayesian Classification

- Bayesian classifier vs. decision tree
  - Decision tree: predict the class label
  - Bayesian classifier: statistical classifier; predict class membership probabilities
- Based on Bayes theorem; estimate *posterior* probability
- Naïve Bayesian classifier:
  - Simple classifier that assumes attribute independence
  - Efficient when applied to large databases
  - Comparable in performance to decision trees

# Posterior Probability

- Let *X* be a data sample whose class label is unknown
- Let $H_i$ be the hypothesis that *X* belongs to a particular class $C_i$
- $P(H_i|X)$ is *posteriori* probability of *H* conditioned on *X*
  - Probability that data example *X* belongs to class $C_i$ given the attribute values of *X*
  - e.g., given X=(age:31…40, income: medium, student: yes, credit: fair), what is the probability X buys computer?

# Bayes Theorem

- To classify means to determine the highest $P(H_i|X)$ among all classes $C_1,\ldots C_m$
  - If $P(H_1|X)>P(H_0|X)$, then X buys computer
  - If $P(H_0|X)>P(H_1|X)$, then X does not buy computer
  - Calculate $P(H_i|X)$ using the Bayes theorem

Class Prior Probability

Descriptor Posterior Probability

$$P(H_i \mid X) = \frac{P(H_i)\,P(X \mid H_i)}{P(X)}$$

Class Posterior Probability

Descriptor Prior Probability

# Class Prior Probability

- $P(H_i)$ is *class prior* probability that $X$ belongs to a particular class $C_i$
  - Can be estimated by $n_i/n$ from training data samples
  - $n$ is the total number of training data samples
  - $n_i$ is the number of training data samples of class $C_i$

| | Age | Income | Student | Credit | Buys_computer |
|---|---|---|---|---|---|
| P1 | 31…40 | high | no | fair | no |
| P2 | <=30 | high | no | excellent | no |
| P3 | 31…40 | high | no | fair | yes |
| P4 | >40 | medium | no | fair | yes |
| P5 | >40 | low | yes | fair | yes |
| P6 | >40 | low | yes | excellent | no |
| P7 | 31…40 | low | yes | excellent | yes |
| P8 | <=30 | medium | no | fair | no |
| P9 | <=30 | low | yes | fair | yes |
| P10 | >40 | medium | yes | fair | yes |

H1: Buys_computer=yes
H0: Buys_computer=no
P(H1)=6/10 = 0.6
P(H0)=4/10 = 0.4

$$P(H_i | X) = \frac{P(X | H_i) P(H_i)}{P(X)}$$

# Descriptor Prior Probability

- P($X$) is *prior* probability of $X$
  - Probability that observe the attribute values of X
  - Suppose $X = (x_1, x_2, \ldots, x_d)$ and they are independent, then $P(X) = P(x_1)\, P(x_2) \ldots P(x_d)$
  - $P(x_j) = n_j/n$, where
  - $n_j$ is number of training examples having value $x_j$ for attribute $A_j$
  - $n$ is the total number of training examples
  - Constant for all classes

| | Age | Income | Student | Credit | Buys_computer |
|---|---|---|---|---|---|
| P1 | 31…40 | high | no | fair | no |
| P2 | <=30 | high | no | excellent | no |
| P3 | 31…40 | high | no | fair | yes |
| P4 | >40 | medium | no | fair | yes |
| P5 | >40 | low | yes | fair | yes |
| P6 | >40 | Low | yes | excellent | No |
| P7 | 31…40 | low | yes | excellent | yes |
| P8 | <=30 | medium | no | fair | no |
| P9 | <=30 | low | yes | fair | yes |
| P10 | >40 | medium | yes | fair | yes |

- X=(age:31…40, income: medium, student: yes, credit: fair)
- P(age=31…40)=3/10 P(income=medium)=3/10
  P(student=yes)=5/10 P(credit=fair)=7/10

$$P(H_i|X)=\frac{P(X|H_i)P(H_i)}{P(X)}$$

- P(X)=P(age=31…40) · P(income=medium) · P(student=yes) · P(credit=fair)
  =0.3 · 0.3 · 0.5 · 0.7 = 0.0315

# Descriptor Posterior Probability

- $P(X|H_i)$ is *posterior* probability of $X$ given $H_i$
  - Probability that observe X in class $C_i$
  - Assume $X=(x_1, x_2,\ldots, x_d)$ and they are independent, then $P(X|H_i) = P(x_1|H_i)\, P(x_2|H_i) \ldots P(x_d|H_i)$
  - $P(x_j|H_i)=n_{i,j}/n_i$, where
  - $n_{i,j}$ is number of training examples in class $C_i$ having value $x_j$ for attribute $A_j$
  - $n_i$ is number of training examples in $C_i$

| | Age | Income | Student | Credit | Buys_computer |
|---|---|---|---|---|---|
| P1 | 31…40 | high | no | fair | no |
| P2 | <=30 | high | no | excellent | no |
| P3 | 31…40 | high | no | fair | yes |
| P4 | >40 | medium | no | fair | yes |
| P5 | >40 | low | yes | fair | yes |
| P6 | >40 | low | yes | excellent | no |
| P7 | 31…40 | low | yes | excellent | yes |
| P8 | <=30 | medium | no | fair | no |
| P9 | <=30 | low | yes | fair | yes |
| P10 | >40 | medium | yes | fair | yes |

- X= (age:31…40, income: medium, student: yes, credit: fair)
- $H_1$ = X buys a computer
- $n_1 = 6$ , $n_{11}=2$, $n_{21}=2$, $n_{31}=4$, $n_{41}=5$,
- $P(X|H_1)= \frac{2}{6}\times\frac{2}{6}\times\frac{4}{6}\times\frac{5}{6}=\frac{5}{81}=0.062$

| | Age | Income | Student | Credit | Buys_computer |
|---|---|---|---|---|---|
| P1 | 31…40 | high | no | fair | no |
| P2 | <=30 | high | no | excellent | no |
| P3 | 31…40 | high | no | fair | yes |
| P4 | >40 | medium | no | fair | yes |
| P5 | >40 | low | yes | fair | yes |
| P6 | >40 | low | yes | excellent | no |
| P7 | 31…40 | low | yes | excellent | yes |
| P8 | <=30 | medium | no | fair | no |
| P9 | <=30 | low | yes | fair | yes |
| P10 | >40 | medium | yes | fair | yes |

- X= (age:31…40, income: medium, student: yes, credit: fair)
- $H_0$ = X does not buy a computer
- $n_0 = 4$ , $n_{10}=1$, $n_{20}=1$, $n_{31}=1$, $n_{41} = 2$,
- $P(X|H_0)= \frac{1}{4} \times \frac{1}{4} \times \frac{1}{4} \times \frac{2}{4} = \frac{1}{128} = 0.0078$

$$P(H_i|X) = \frac{P(X|H_i)P(H_i)}{P(X)}$$

# Bayesian Classifier – Basic Equation

Class Prior Probability

Descriptor Posterior Probability

$$P(H_i \mid X) = \frac{P(H_i)\,P(X \mid H_i)}{P(X)}$$

Class Posterior Probability

Descriptor Prior Probability

To classify means to determine the highest $P(H_i|X)$ among all classes $C_1,\ldots C_m$

$P(X)$ is constant to all classes

**Only need to compare $P(H_i)P(X|H_i)$**

# Weather Dataset Example

*X* =< rain, hot, high, false>

| Outlook | Temperature | Humidity | Windy | Class |
|---|---|---|---|---|
| sunny | hot | high | false | N |
| sunny | hot | high | true | N |
| overcast | hot | high | false | P |
| rain | mild | high | false | P |
| rain | cool | normal | false | P |
| rain | cool | normal | true | N |
| overcast | cool | normal | true | P |
| sunny | mild | high | false | N |
| sunny | cool | normal | false | P |
| rain | mild | normal | false | P |
| sunny | mild | normal | true | P |
| overcast | mild | high | true | P |
| overcast | hot | normal | false | P |
| rain | mild | high | true | N |

# Weather Dataset Example

- Given a training set, we can compute probabilities:

$P(H_i)$

| |
|---|
| P(p) = 9/14 |
| P(n) = 5/14 |

$P(x_j | H_i)$

| Outlook | P | N |
|---|---|---|
| sunny | 2/9 | 3/5 |
| overcast | 4/9 | 0 |
| rain | 3/9 | 2/5 |

| Temperature | P | N |
|---|---|---|
| hot | 2/9 | 2/5 |
| mild | 4/9 | 2/5 |
| cool | 3/9 | 1/5 |

| Humidity | P | N |
|---|---|---|
| high | 3/9 | 4/5 |
| normal | 6/9 | 1/5 |

| Windy | P | N |
|---|---|---|
| true | 3/9 | 3/5 |
| false | 6/9 | 2/5 |

# Weather Dataset Example: Classifying X

- An unseen sample $X$ = <rain, hot, high, false>
- $P(p)\ P(X|p)$
  $= P(p)\ \ P(\text{rain}|p)\ P(\text{hot}|p)\ P(\text{high}|p)\ P(\text{false}|p)$
  $= 9/14 \cdot 3/9 \cdot 2/9 \cdot 3/9 \cdot 6/9 \cdot = 0.010582$
- $P(n)\ P(X|n)$
  $= P(n)\ P(\text{rain}|n)\ P(\text{hot}|n)\ P(\text{high}|n)\ P(\text{false}|n)$
  $= 5/14 \cdot 2/5 \cdot 2/5 \cdot 4/5 \cdot 2/5 = 0.018286$
- Sample $X$ is classified in class $n$ (don't play)

# Avoiding the Zero-Probability Problem

- Descriptor posterior probability goes to 0 if any of probability is 0:

$$P(X \mid H_i) = \prod_{j=1}^{d} P(x_j \mid H_i)$$

- Ex. Suppose a dataset with 1000 tuples for a class C, income=low (0), income= medium (990), and income = high (10)
- Use **Laplacian correction** (or Laplacian estimator)
  - *Adding 1 to each case*

    Prob(income = low|H) = 1/1003

    Prob(income = medium|H) = 991/1003

    Prob(income = high|H) = 11/1003

# Independence Hypothesis

- makes computation possible
- yields optimal classifiers when satisfied
- but is seldom satisfied in practice, as attributes (variables) are often correlated
- Attempts to overcome this limitation:
  - Bayesian networks, that combine Bayesian reasoning with causal relationships between attributes

# Logistic Regression Classifier

- **Input distribution**
  - X is n-dimensional feature vector $< X_1 \ldots X_n >$
  - Y is 0 or 1
  - X|Y ~ Gaussian distribution
  - Y ~ Bernoulli distribution
- **Model P(Y|X)**
  - What does P(Y|X) look like?
  - What does P(Y=0|X)/P(Y=1|X) look like?

$$P(Y=1|X) = \frac{P(Y=1)P(X|Y=1)}{P(Y=1)P(X|Y=1) + P(Y=0)P(X|Y=0)}$$

$$= \frac{1}{1 + \frac{P(Y=0)P(X|Y=0)}{P(Y=1)P(X|Y=1)}}$$

$$= \frac{1}{1 + \exp(\ln \frac{P(Y=0)P(X|Y=0)}{P(Y=1)P(X|Y=1)})}$$

$$= \frac{1}{1 + \exp( \ (\ln \frac{1-\pi}{\pi}) + \sum_i \ln \frac{P(X_i|Y=0)}{P(X_i|Y=1)})}$$

$$P(x \mid y_k) = \frac{1}{\sigma_{ik}\sqrt{2\pi}} \ e^{\frac{-(x-\mu_{ik})^2}{2\sigma_{ik}^2}}$$

$$\sum_i \left( \frac{\mu_{i0} - \mu_{i1}}{\sigma_i^2} X_i + \frac{\mu_{i1}^2 - \mu_{i0}^2)}{2\sigma_i^2} \right)$$

$$P(Y=1|X) = \frac{1}{1 + \exp(w_0 + \sum_{i=1}^n w_i X_i)}$$

$$P(Y=1|X=<X_1,...X_n>) = \frac{1}{1+exp(w_0+\sum_i w_i X_i)}$$

implies

$$P(Y=0|X=<X_1,...X_n>) = \frac{exp(w_0+\sum_i w_i X_i)}{1+exp(w_0+\sum_i w_i X_i)}$$

implies

$$\frac{P(Y=0|X)}{P(Y=1|X)} = exp(w_0+\sum_i w_i X_i)$$

linear classification rule!

implies

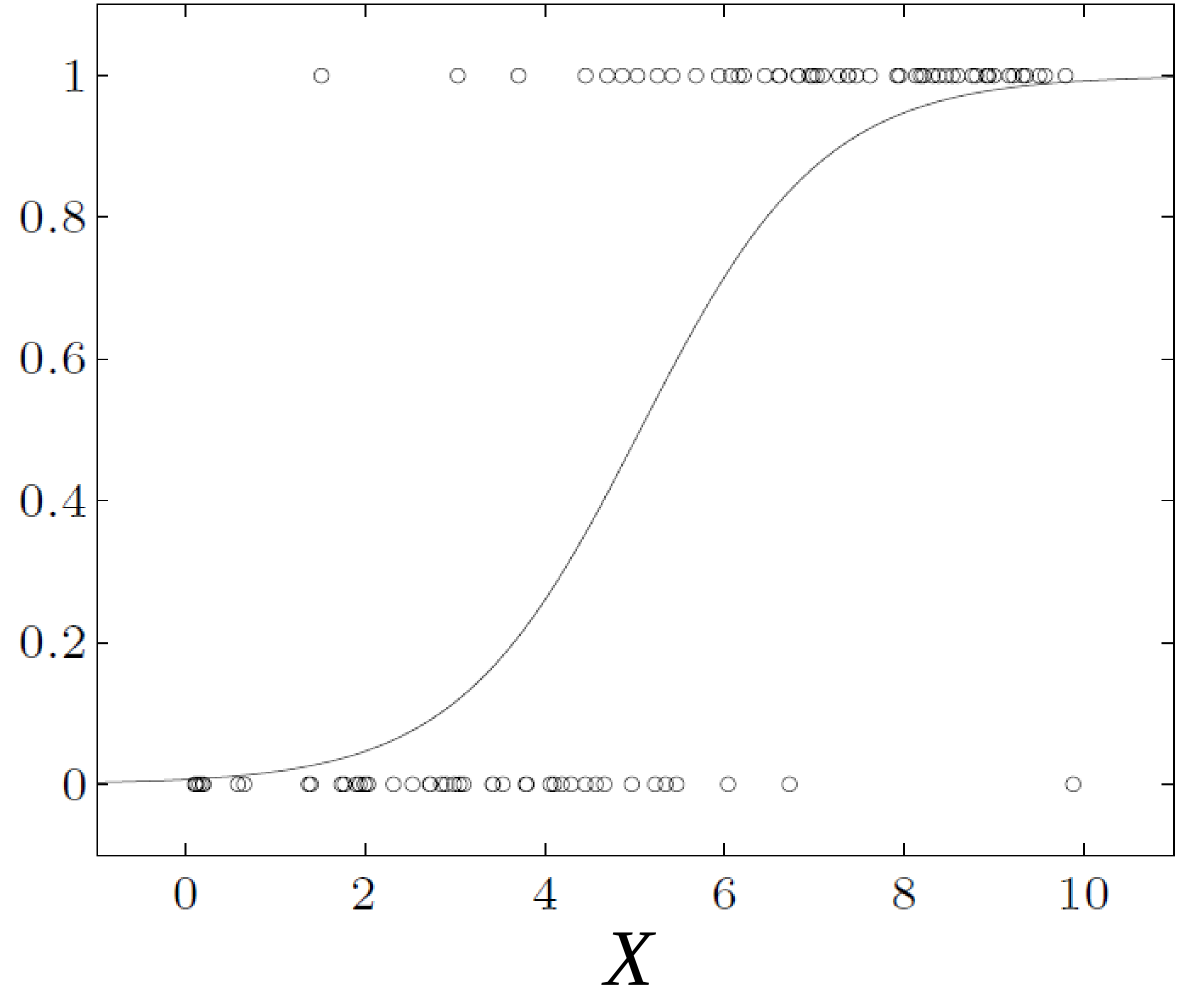
$$\ln \frac{P(Y=0|X)}{P(Y=1|X)} = w_0+\sum_i w_i X_i$$

**Log ratio**:

Positive—Class 0 Negative—Class 1

# Logistic Function

$$P(Y=1|X) = \frac{1}{1+\exp(wX+b)}$$




**Training set**:

Y=1—P(Y=1|X)=1 Y=0—P(Y=1|X)=0

# Maximizing Conditional Likelihood

- Training Set: $\{\langle X^1, Y^1\rangle, \ldots \langle X^L, Y^L\rangle\}$
- Find W that maximizes conditional likelihood:

$$\arg\max_W \prod_l P(Y^l|W, X^l)$$

$$P(Y=1|X=<X_1,...X_n>) = \frac{1}{1+exp(w_0+\sum_i w_i X_i)}$$

$$P(Y=0|X=<X_1,...X_n>) = \frac{exp(w_0+\sum_i w_i X_i)}{1+exp(w_0+\sum_i w_i X_i)}$$

- A concave function in W
- Gradient descent approach to solve it

# Rule-Based Classifier

- Classify records by using a collection of "if…then…" rules
- Rule: $(Condition) \rightarrow y$
  - where
    - *Condition* is a conjunctions of attributes
    - *y* is the class label
  - *LHS*: rule condition
  - *RHS*: rule consequent
  - Examples of classification rules:
    - (Blood Type=Warm) $\wedge$ (Lay Eggs=Yes) $\rightarrow$ Birds
    - (Taxable Income < 50K) $\wedge$ (Refund=Yes) $\rightarrow$ Evade=No

# Rule-based Classifier (Example)

| Name | Blood Type | Give Birth | Can Fly | Live in Water | Class |
|---|---|---|---|---|---|
| human | warm | yes | no | no | mammals |
| python | cold | no | no | no | reptiles |
| salmon | cold | no | no | yes | fishes |
| whale | warm | yes | no | yes | mammals |
| frog | cold | no | no | sometimes | amphibians |
| komodo | cold | no | no | no | reptiles |
| bat | warm | yes | yes | no | mammals |
| pigeon | warm | no | yes | no | birds |
| cat | warm | yes | no | no | mammals |
| leopard shark | cold | yes | no | yes | fishes |
| turtle | cold | no | no | sometimes | reptiles |
| penguin | warm | no | no | sometimes | birds |
| porcupine | warm | yes | no | no | mammals |
| eel | cold | no | no | yes | fishes |
| salamander | cold | no | no | sometimes | amphibians |
| gila monster | cold | no | no | no | reptiles |
| platypus | warm | no | no | no | mammals |
| owl | warm | no | yes | no | birds |
| dolphin | warm | yes | no | yes | mammals |
| eagle | warm | no | yes | no | birds |

R1: (Give Birth = no) $\wedge$ (Can Fly = yes) $\rightarrow$ Birds

R2: (Give Birth = no) $\wedge$ (Live in Water = yes) $\rightarrow$ Fishes

R3: (Give Birth = yes) $\wedge$ (Blood Type = warm) $\rightarrow$ Mammals

R4: (Give Birth = no) $\wedge$ (Can Fly = no) $\rightarrow$ Reptiles

R5: (Live in Water = sometimes) $\rightarrow$ Amphibians

# Application of Rule-Based Classifier

- A rule *r* covers an instance **x** if the attributes of the instance satisfy the condition of the rule

R1: (Give Birth = no) $\wedge$ (Can Fly = yes) $\rightarrow$ Birds

R2: (Give Birth = no) $\wedge$ (Live in Water = yes) $\rightarrow$ Fishes

R3: (Give Birth = yes) $\wedge$ (Blood Type = warm) $\rightarrow$ Mammals

R4: (Give Birth = no) $\wedge$ (Can Fly = no) $\rightarrow$ Reptiles

R5: (Live in Water = sometimes) $\rightarrow$ Amphibians

| Name | Blood Type | Give Birth | Can Fly | Live in Water | Class |
|---|---|---|---|---|---|
| hawk | warm | no | yes | no | ? |
| grizzly bear | warm | yes | no | no | ? |

The rule R1 covers a hawk => Bird

The rule R3 covers the grizzly bear => Mammal

# Rule Coverage and Accuracy

- **Coverage of a rule:**
  - Fraction of records that satisfy the condition of a rule
- **Accuracy of a rule:**
  - Fraction of records that satisfy both the LHS and RHS of a rule

| *Tid* | Refund | Marital Status | Taxable Income | Class |
|---|---|---|---|---|
| 1 | Yes | Single | 125K | No |
| 2 | No | Married | 100K | No |
| 3 | No | Single | 70K | No |
| 4 | Yes | Married | 120K | No |
| 5 | No | Divorced | 95K | Yes |
| 6 | No | Married | 60K | No |
| 7 | Yes | Divorced | 220K | No |
| 8 | No | Single | 85K | Yes |
| 9 | No | Married | 75K | No |
| 10 | No | Single | 90K | Yes |

(Status=Single) $\rightarrow$ No

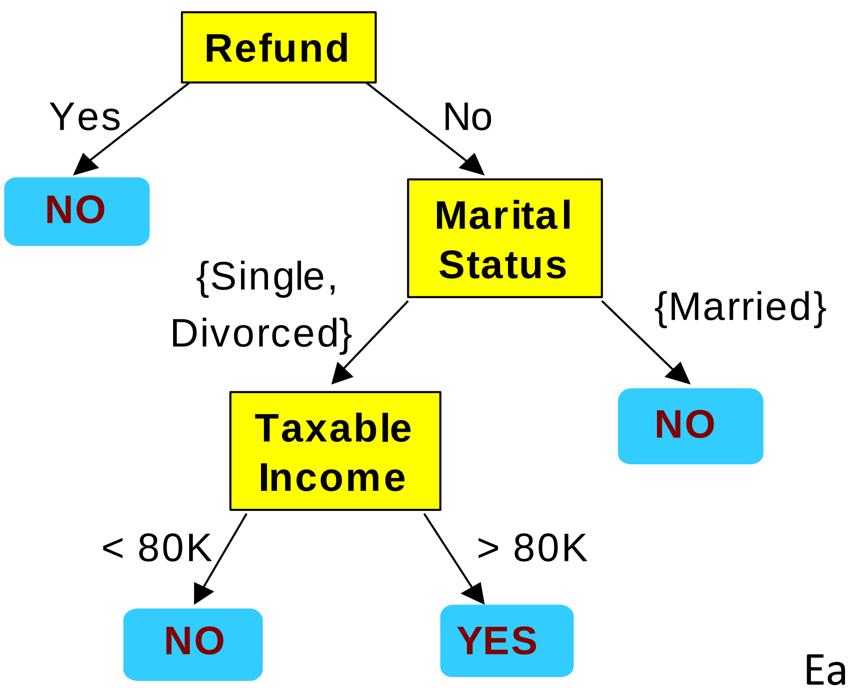
Coverage = 40%,  Accuracy = 50%

# Characteristics of Rule-Based Classifier

- **Mutually exclusive rules**
  - Classifier contains mutually exclusive rules if the rules are independent of each other
  - Every record is covered by at most one rule

- **Exhaustive rules**
  - Classifier has exhaustive coverage if it accounts for every possible combination of attribute values
  - Each record is covered by at least one rule

# From Decision Trees To Rules




**Classification Rules**

(Refund=Yes) ==> No

(Refund=No, Marital Status={Single,Divorced}, Taxable Income<80K) ==> No

(Refund=No, Marital Status={Single,Divorced}, Taxable Income>80K) ==> Yes

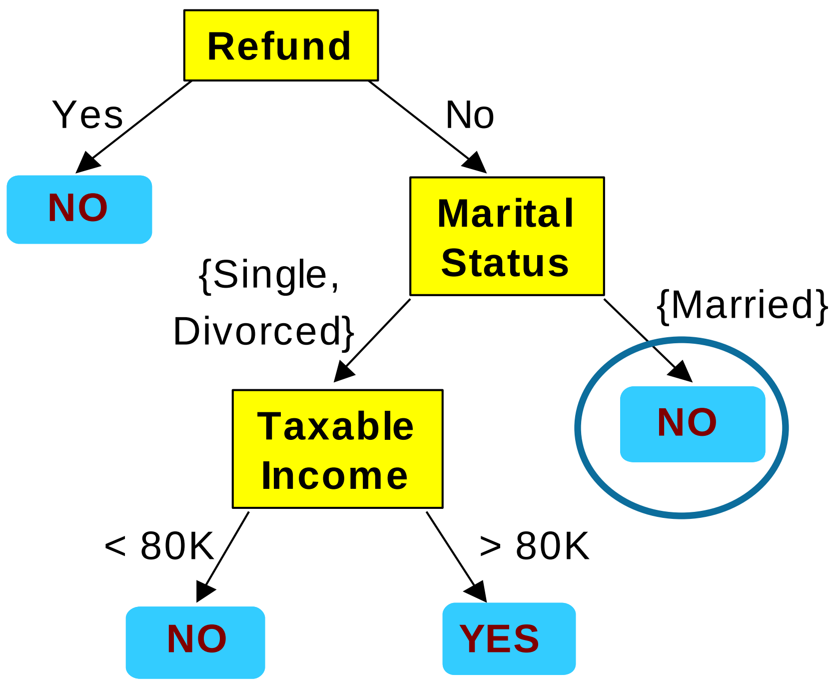
(Refund=No, Marital Status={Married}) ==> No

Each path in the tree forms a rule

Rules are mutually exclusive and exhaustive

Rule set contains as much information as the tree

# Rules Can Be Simplified




| *Tid* | Refund | Marital Status | Taxable Income | Cheat |
|---|---|---|---|---|
| 1 | Yes | Single | 125K | No |
| 2 | No | Married | 100K | No |
| 3 | No | Single | 70K | No |
| 4 | Yes | Married | 120K | No |
| 5 | No | Divorced | 95K | Yes |
| 6 | No | Married | 60K | No |
| 7 | Yes | Divorced | 220K | No |
| 8 | No | Single | 85K | Yes |
| 9 | No | Married | 75K | No |
| 10 | No | Single | 90K | Yes |

Initial Rule: (Refund=No) $\wedge$ (Status=Married) $\rightarrow$ No

Simplified Rule: (Status=Married) $\rightarrow$ No

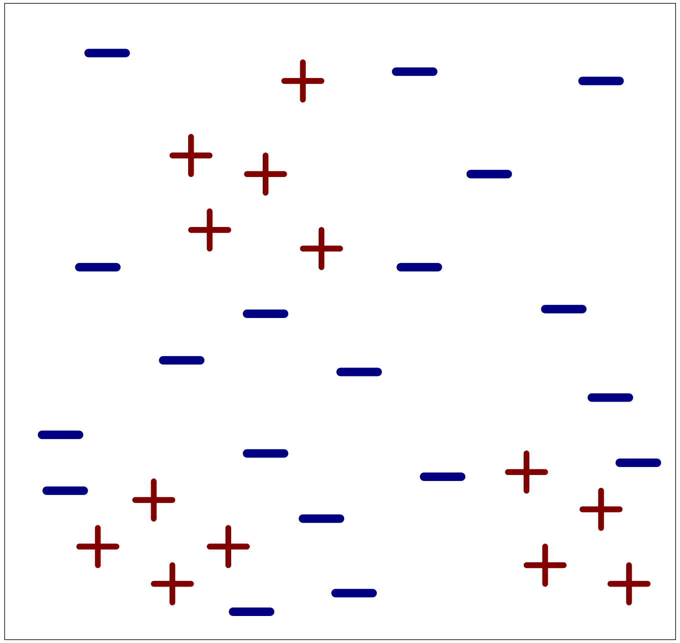
# Effect of Rule Simplification

- **Rules are no longer mutually exclusive**
  - A record may trigger more than one rule
  - Solution?
    - Ordered rule set
    - Unordered rule set – use voting schemes
- **Rules are no longer exhaustive**
  - A record may not trigger any rules
  - Solution?
    - Use a default class

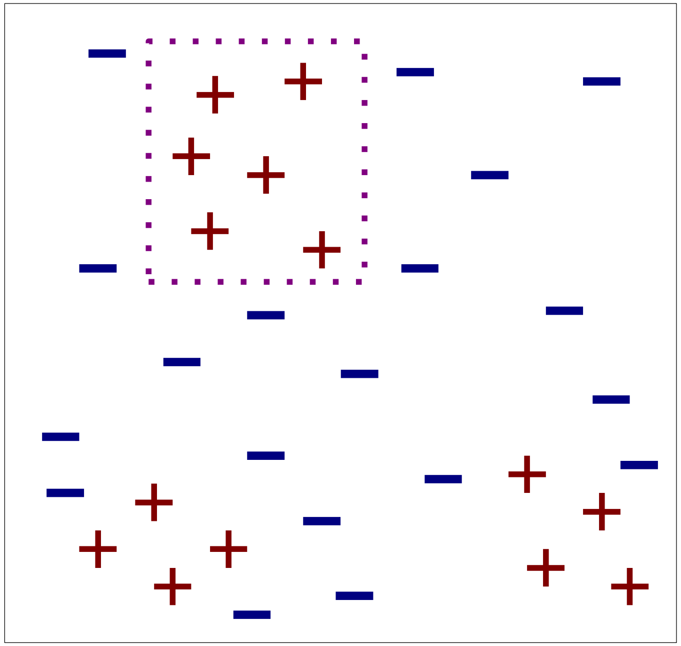
# Learn Rules from Data: Sequential Covering

1. Start from an empty rule
2. Grow a rule using the Learn-One-Rule function
3. Remove training records covered by the rule
4. Repeat Step (2) and (3) until stopping criterion is met

# Example of Sequential Covering

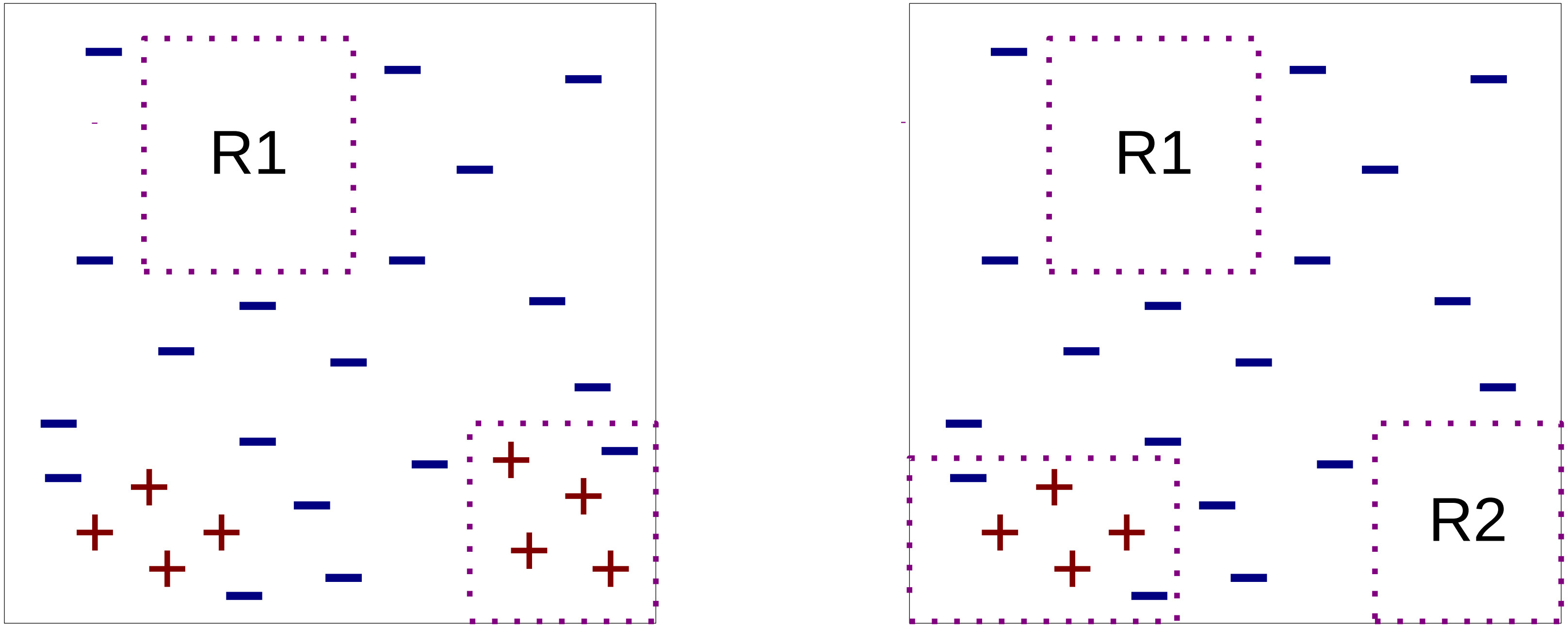


(i) Original Data



(ii) Step 1

# Example of Sequential Covering…




(iii) Step 2 (iv) Step 3

# How to Learn-One-Rule?

- Start with the *most general rule* possible: condition = empty
- *Adding new attributes* by adopting a greedy depth-first strategy
  - Picks the one that most improves the rule quality
- Rule-Quality measures: consider both coverage and accuracy
  - Foil-gain: assesses info_gain by extending condition

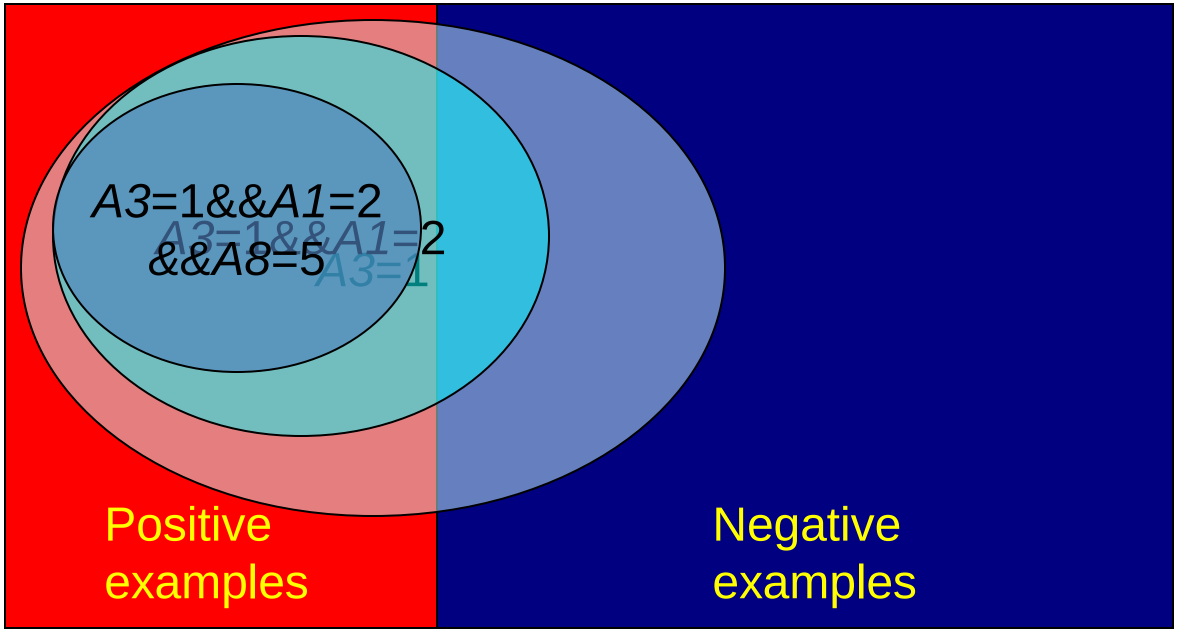
$$FOIL\_Gain = pos' \times (\log_2 \frac{pos'}{pos'+neg'} - \log_2 \frac{pos}{pos+neg})$$

    - favors rules that have high accuracy and cover many positive tuples

# Rule Generation

- **To generate a rule**

**while**(true)
find the best predicate *p*
**if** foil-gain(*p*) > threshold **then** add *p* to current rule
**else** break



# Associative Classification

- **Associative classification: Major steps**
  - Mine data to find strong associations between frequent patterns (conjunctions of attribute-value pairs) and class labels
  - Association rules are generated in the form of

    $$P_1 \wedge p_2 \ldots \wedge p_l \rightarrow \text{“}A_{class} = C\text{”} \text{ (conf, sup)}$$

  - Organize the rules to form a rule-based classifier

# Associative Classification

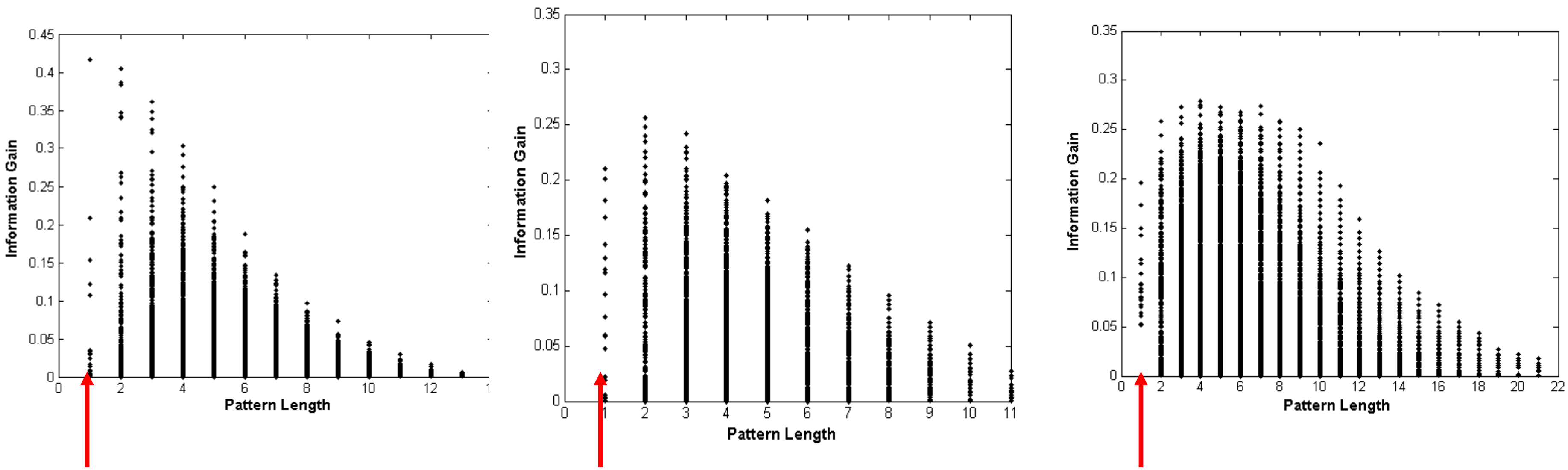
- **Why effective?**
  - It explores highly confident associations among multiple attributes and may overcome some constraints introduced by decision-tree induction, which considers only one attribute at a time
  - Associative classification has been found to be often more accurate than some traditional classification methods, such as C4.5

# Associative Classification

- **Basic idea**
  - Mine possible association rules in the form of
    - Cond-set (a set of attribute-value pairs) → class label
  - Pattern-based approach
    - Mine frequent patterns as candidate condition sets
    - Choose a subset of frequent patterns based on discriminativeness and redundancy

# Frequent Pattern vs. Single Feature

The discriminative power of some frequent patterns is higher than that of single features.




(a) Austral (b) Cleve (c) Sonar

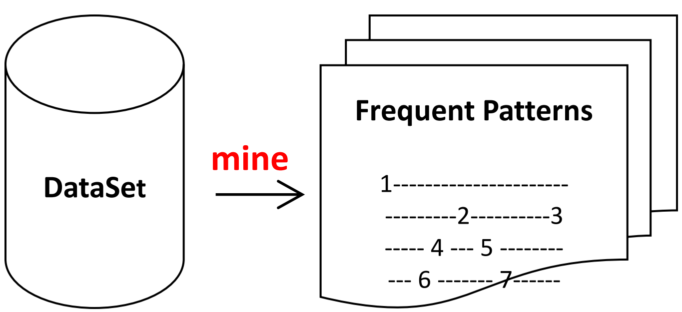
Information Gain vs. Pattern Length

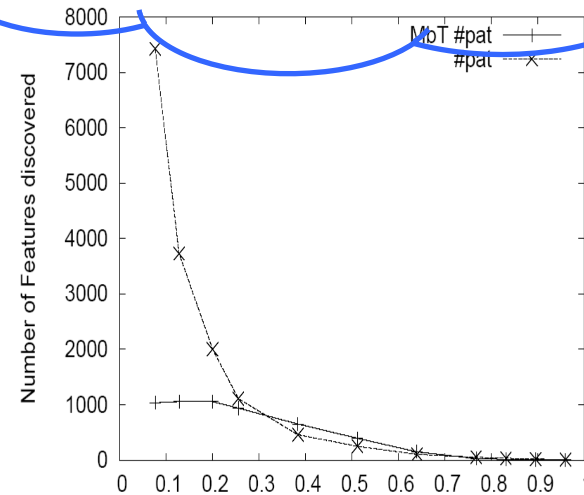
# Two Problems

- **Mine step**
  - **combinatorial explosion**

1. exponential explosion

2. patterns not considered if *minsupport* isn't small enough




(a) Number of itemsets mined with varying supports

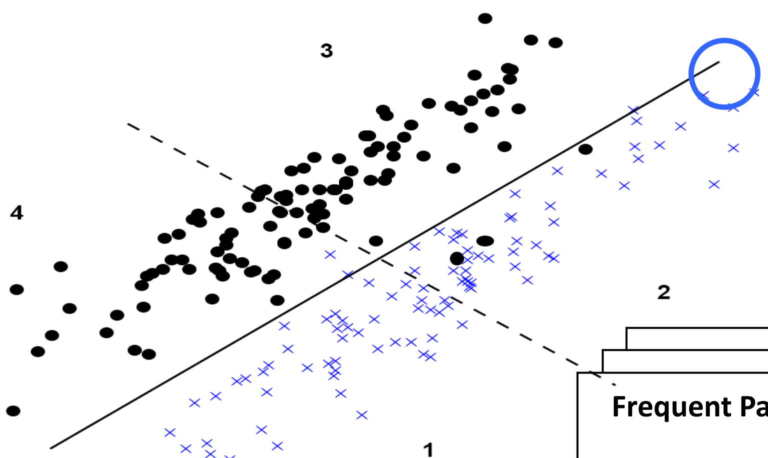
# Two Problems

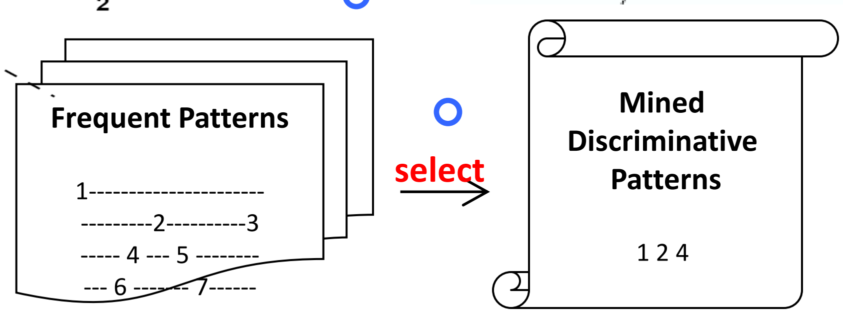
- **Select step**
  - **Issue of discriminative power**

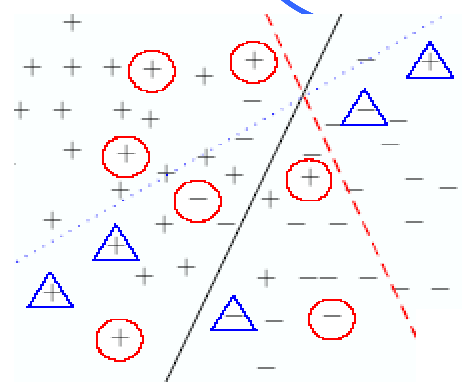
3. InfoGain against the complete dataset, NOT on subset of examples

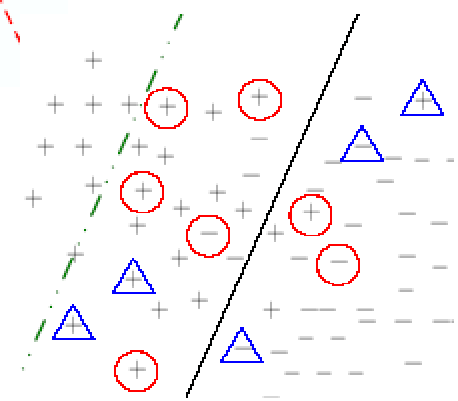
4. Correlation not directly evaluated on their joint predictability

Uncorrelated Patterns $\neq$ higher accuracy



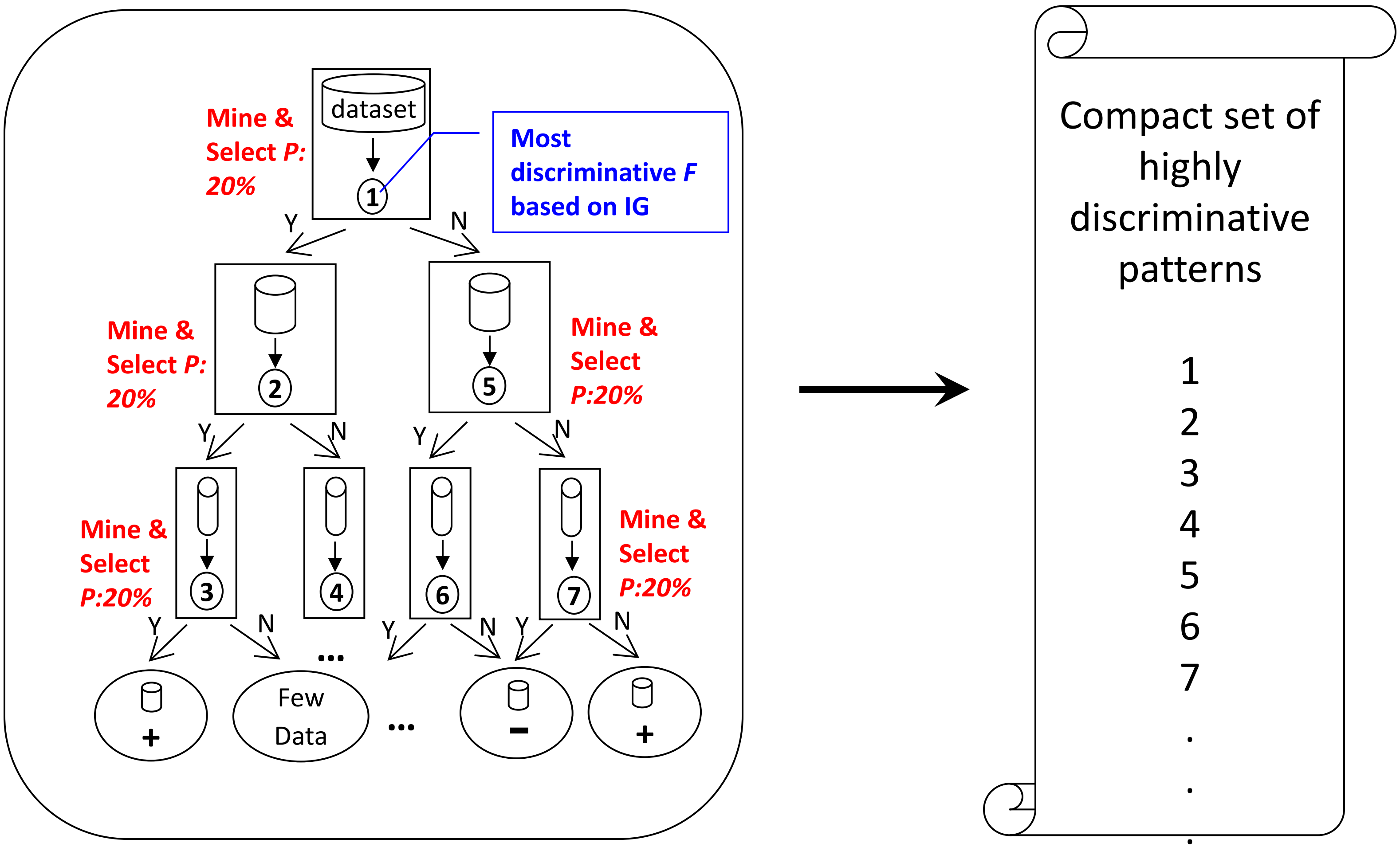





# Direct Mining & Selection via Model-based Search Tree

- **Basic Flow**



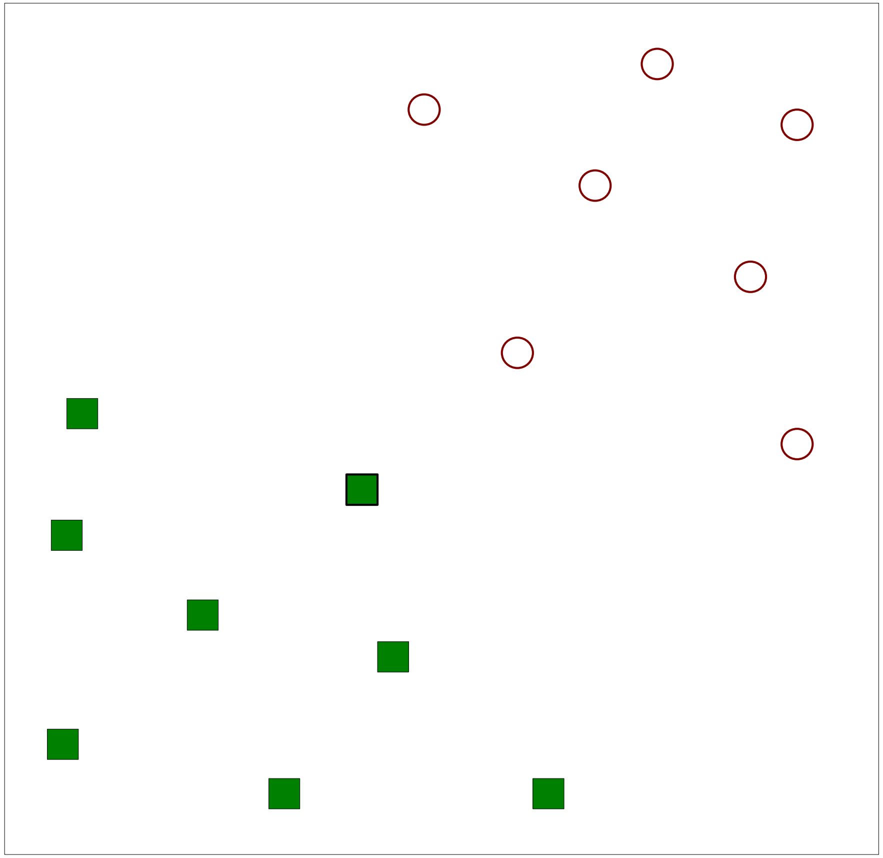

**Divide-and-Conquer Based Frequent Pattern Mining**

**Mined Discriminative Patterns**

# Advantages of Rule-Based Classifiers

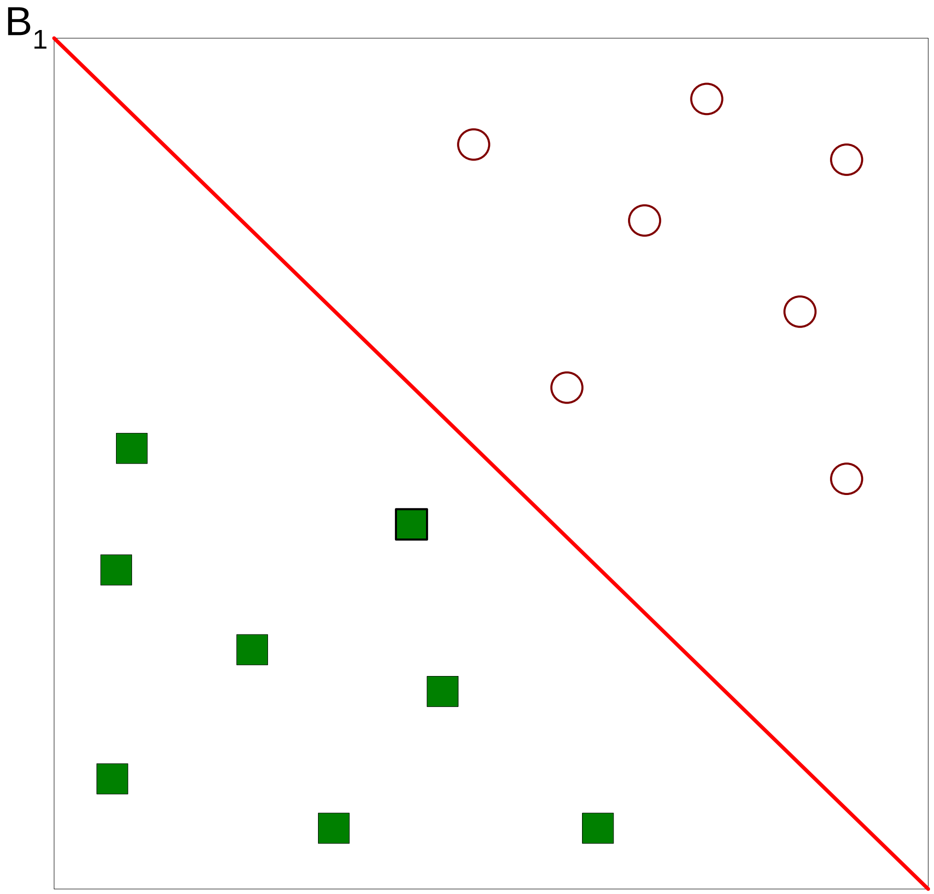
- As highly expressive as decision trees
- Easy to interpret
- Easy to generate
- Can classify new instances rapidly
- Performance comparable to decision trees

# Support Vector Machines—An Example



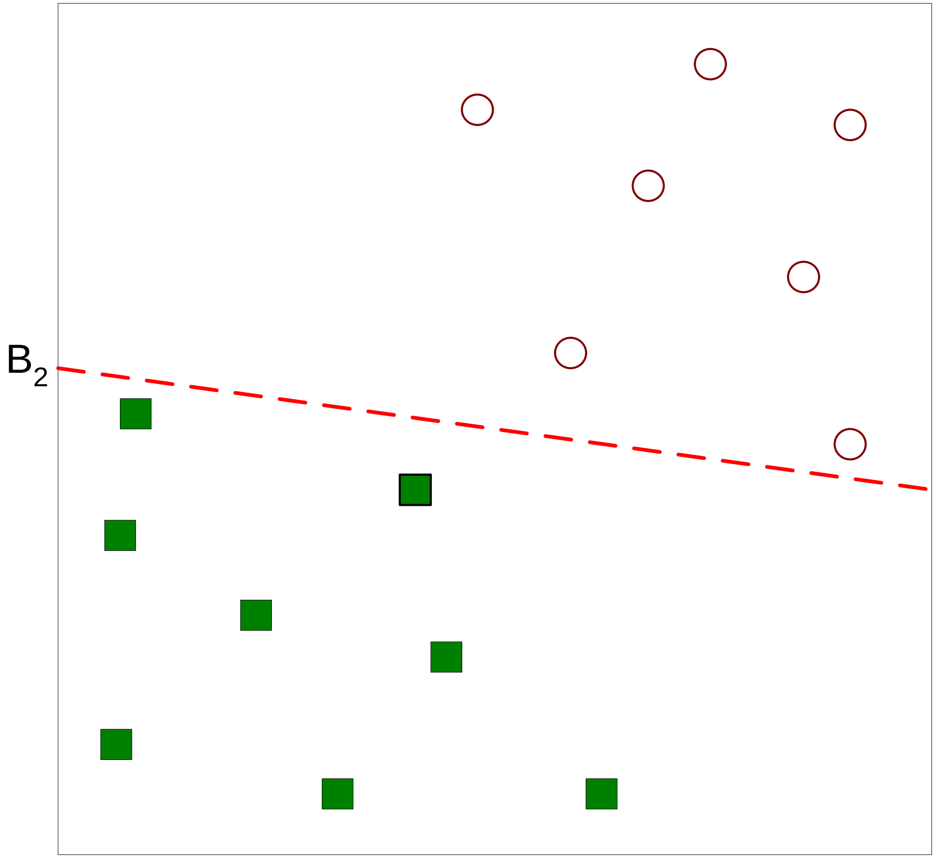
- Find a linear hyperplane (decision boundary) that will separate the data

# Example



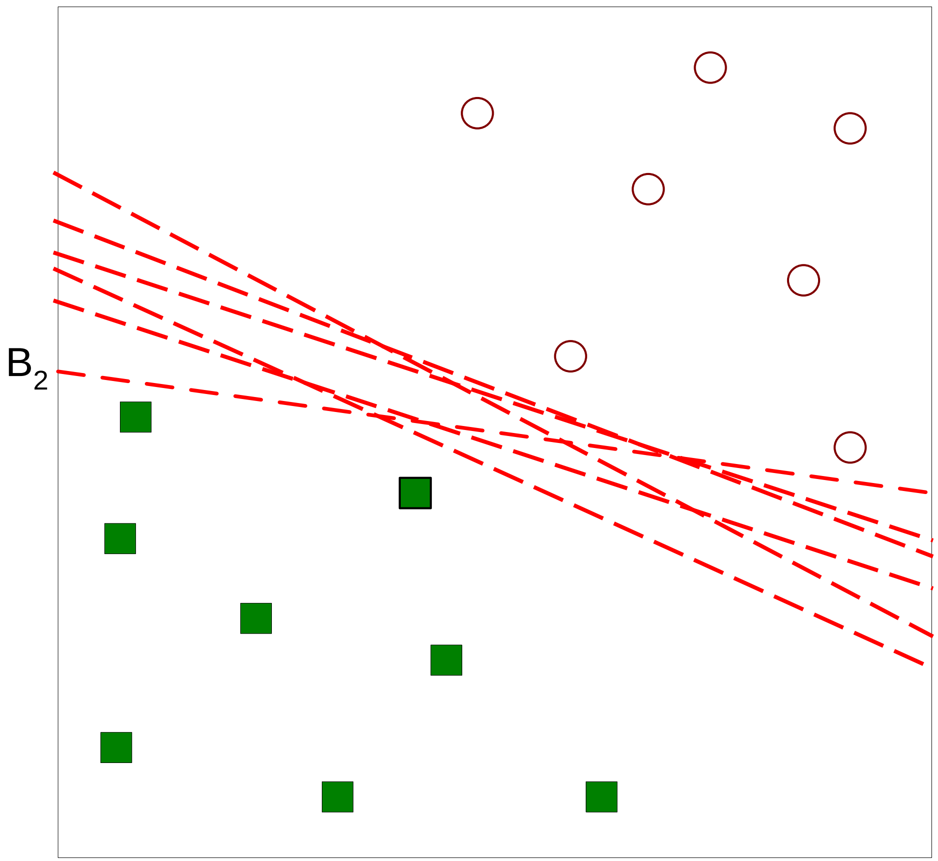

- One Possible Solution

# Example



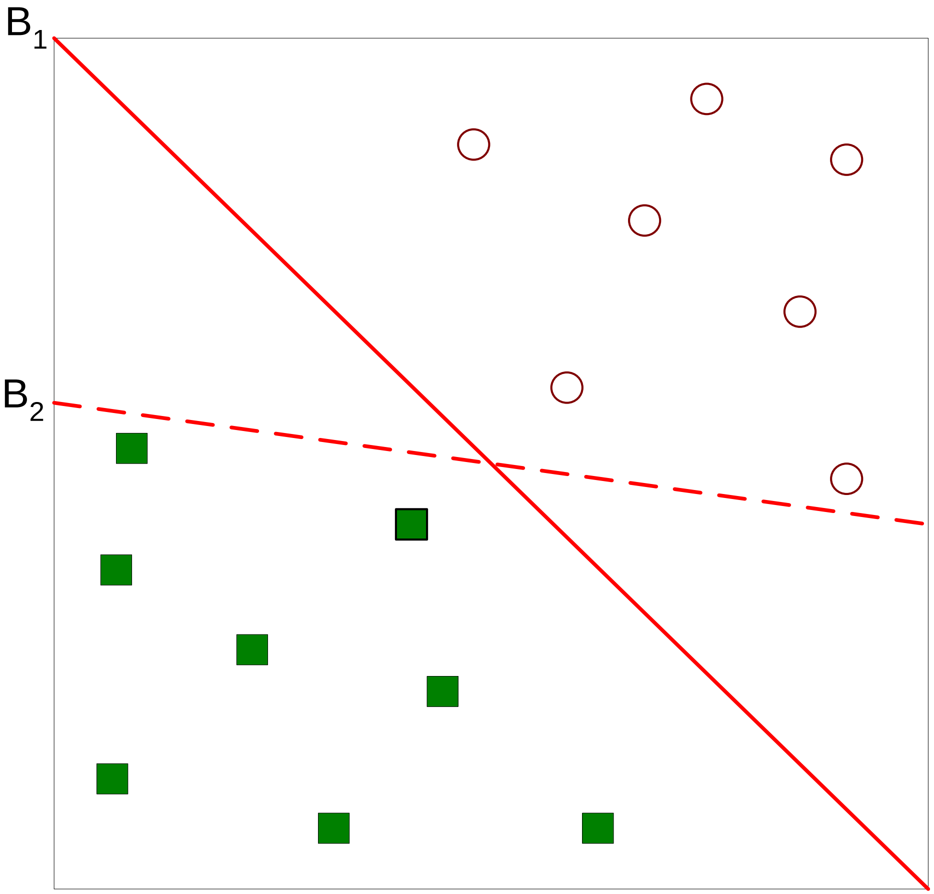

- Another possible solution

# Example



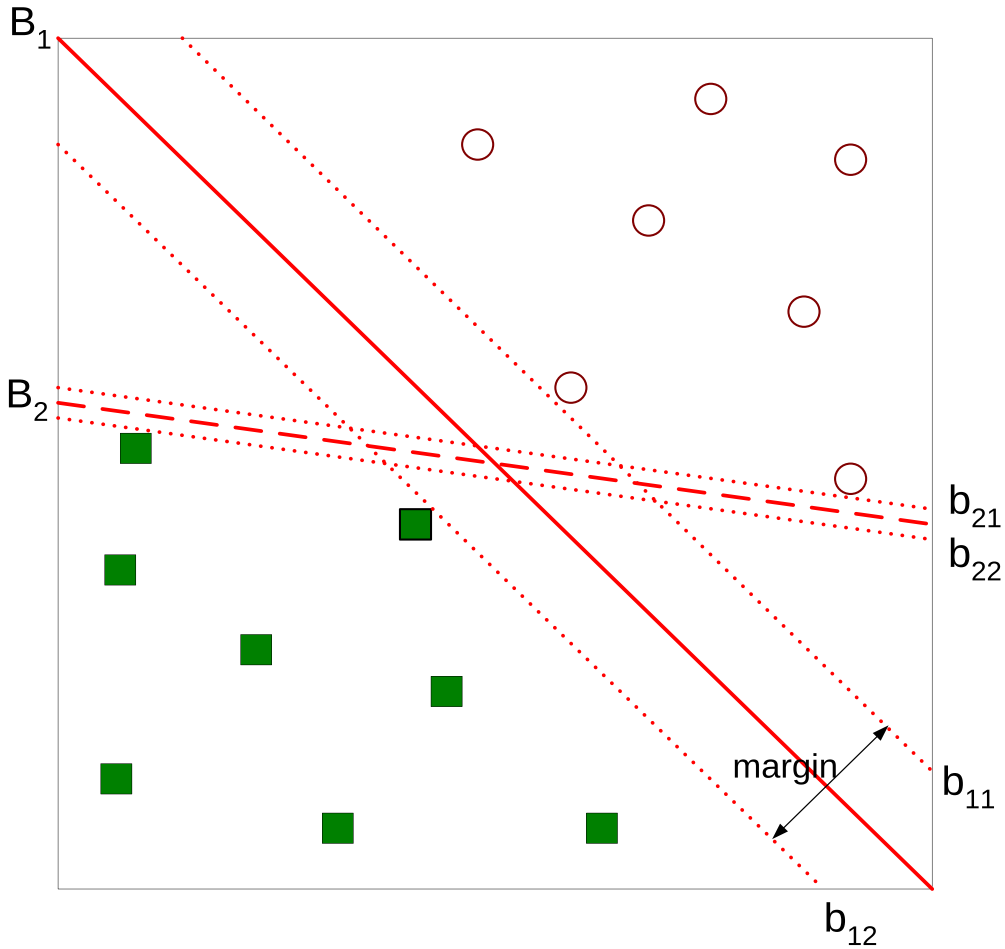

- Other possible solutions

# Choosing Decision Boundary



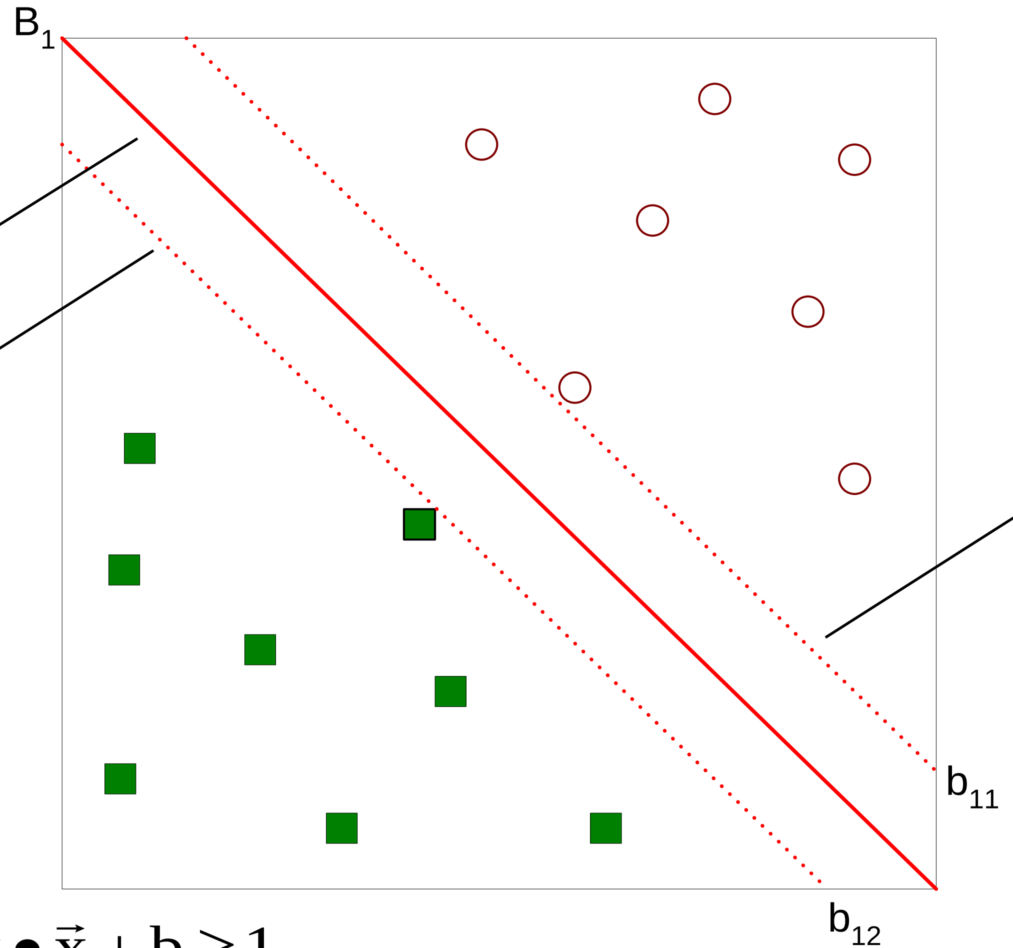

- Which one is better? B1 or B2?
- How do you define better?

# Maximize Margin between Classes




- Find hyperplane maximizes the margin => B1 is better than B2

# Formal Definition




$$\vec{w} \bullet \vec{x} + b = 0$$

$$\vec{w} \bullet \vec{x} + b = -1$$

$$\vec{w} \bullet \vec{x} + b = +1$$

$$y = \begin{cases} 1 & \text{if } \vec{\mathrm{w}} \bullet \vec{\mathrm{x}} + \mathrm{b} \geq 1 \\ -1 & \text{if } \vec{\mathrm{w}} \bullet \vec{\mathrm{x}} + \mathrm{b} \leq -1 \end{cases}$$

$$\text{Margin} = \frac{2}{\|\vec{w}\|^2}$$

# Support Vector Machines

- We want to maximize: $\text{Margin} = \dfrac{2}{\|\vec{w}\|^2}$
  - Which is equivalent to minimizing: $L(w) = \dfrac{\|\vec{w}\|^2}{2}$
  - But subjected to the following constraints:

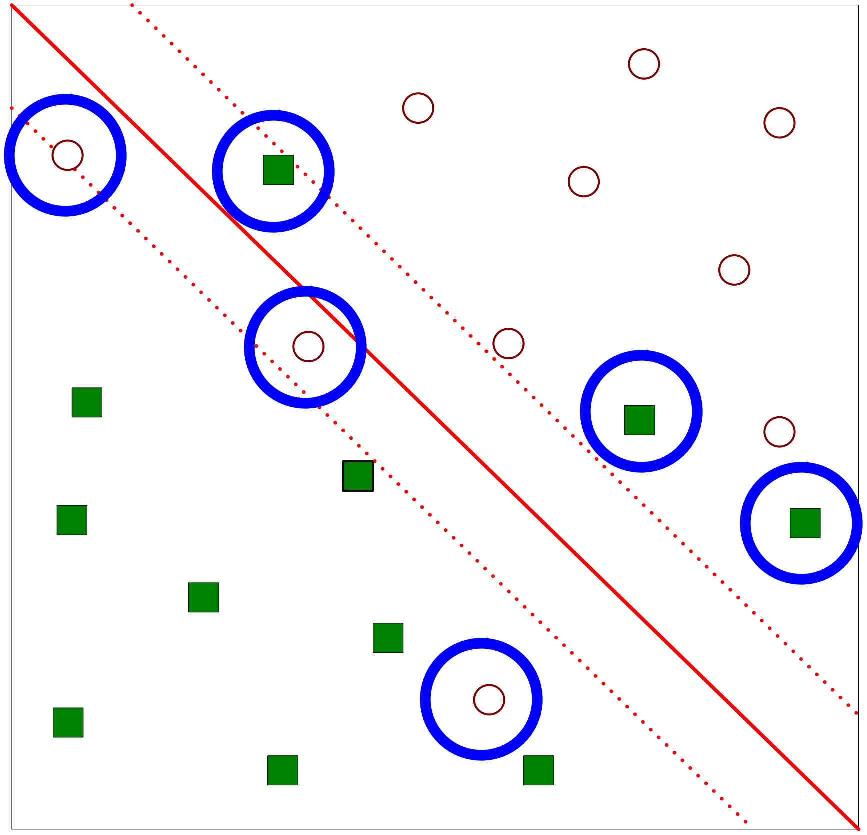
$$\vec{w} \bullet \vec{x}_i + b \geq 1 \text{ if } y_i = 1$$

$$\vec{w} \bullet \vec{x}_i + b \leq -1 \text{ if } y_i = -1$$

  - This is a constrained optimization problem
    - Numerical approaches to solve it (e.g., quadratic programming)

# Noisy Data

- What if the problem is not linearly separable?



# Slack Variables

- What if the problem is not linearly separable?
  - Introduce slack variables
    - Need to minimize:

$$L(w) = \frac{\|\vec{w}\|^2}{2} + C\left(\sum_{i=1}^{N} \xi_i^k\right)$$

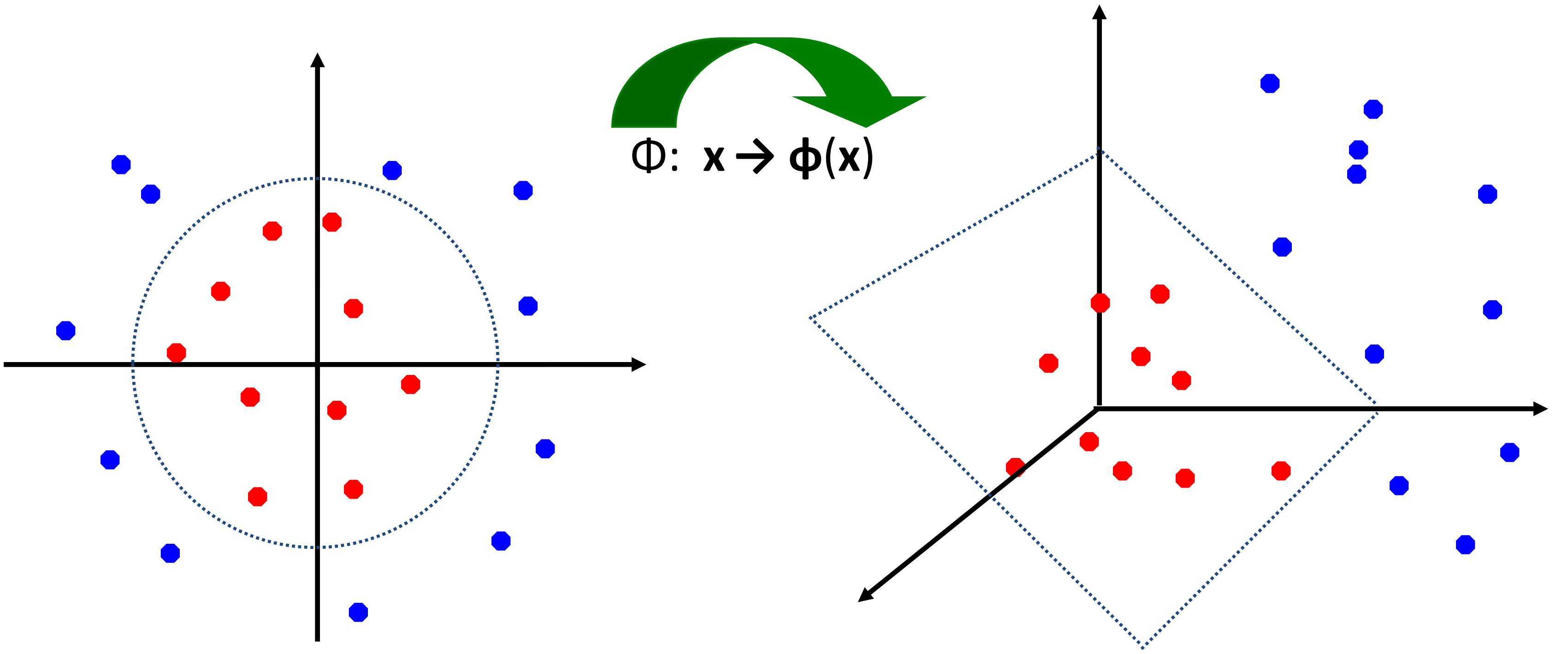
    - Subject to:

$$\vec{\mathrm{w}} \bullet \vec{\mathrm{x}}_{\mathrm{i}} + \mathrm{b} \geq 1 - \xi_{\mathrm{i}} \quad \text{if} \quad \mathrm{y}_{\mathrm{i}} = 1$$

$$\vec{\mathrm{w}} \bullet \vec{\mathrm{x}}_{\mathrm{i}} + \mathrm{b} \leq -1 + \xi_i \quad \text{if} \quad \mathrm{y}_{\mathrm{i}} = -1$$

# 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 linearly separable:



# Ensemble Learning

- **Problem**
  - Given a data set $D=\{x_1,x_2,\ldots,x_n\}$ and their corresponding labels $L=\{l_1,l_2,\ldots,l_n\}$
  - An ensemble approach computes:
    - A set of classifiers $\{f_1,f_2,\ldots,f_k\}$, each of which maps data to a class label: $f_j(x)=l$
    - A combination of classifiers $f^*$ which minimizes generalization error: $f^*(x)= w_1 f_1(x)+ w_2 f_2(x)+\ldots+ w_k f_k(x)$

# Generating Base Classifiers

- **Sampling training examples**
  - Train k classifiers on k subsets drawn from the training set
- **Using different learning models**
  - Use all the training examples, but apply different learning algorithms
- **Sampling features**
  - Train k classifiers on k subsets of features drawn from the feature space
- **Learning "randomly"**
  - Introduce randomness into learning procedures

# Bagging (1)

- **Bootstrap**
  - Sampling with replacement
  - Contains around 63.2% original records in each sample
- **Bootstrap Aggregation**
  - Train a classifier on each bootstrap sample
  - Use majority voting to determine the class label of ensemble classifier

# Bagging (2)

**Original Data:**

| x | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 | 0.8 | 0.9 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|
| y | 1 | 1 | 1 | -1 | -1 | -1 | -1 | 1 | 1 | 1 |

**Bootstrap samples and classifiers:**

| x | 0.1 | 0.2 | 0.2 | 0.3 | 0.4 | 0.4 | 0.5 | 0.6 | 0.9 | 0.9 |
|---|---|---|---|---|---|---|---|---|---|---|
| y | 1 | 1 | 1 | 1 | -1 | -1 | -1 | -1 | 1 | 1 |

| x | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 | 0.5 | 0.9 | 1 | 1 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|
| y | 1 | 1 | 1 | -1 | -1 | -1 | 1 | 1 | 1 | 1 |

| x | 0.1 | 0.2 | 0.3 | 0.4 | 0.4 | 0.5 | 0.7 | 0.7 | 0.8 | 0.9 |
|---|---|---|---|---|---|---|---|---|---|---|
| y | 1 | 1 | 1 | -1 | -1 | -1 | -1 | -1 | 1 | 1 |

| x | 0.1 | 0.2 | 0.5 | 0.5 | 0.5 | 0.7 | 0.7 | 0.8 | 0.9 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|
| y | 1 | 1 | -1 | -1 | -1 | -1 | -1 | 1 | 1 | 1 |

**Combine predictions by majority voting**

# Boosting (1)

- **Principles**
  - Boost a set of weak learners to a strong learner
  - Make records currently misclassified more important

- **Example**
  - Record 4 is hard to classify
  - Its weight is increased, therefore it is more likely to be chosen again in subsequent rounds

| Original Data | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| Boosting (Round 1) | 7 | 3 | 2 | 8 | 7 | 9 | 4 | 10 | 6 | 3 |
| Boosting (Round 2) | 5 | 4 | 9 | 4 | 2 | 5 | 1 | 7 | 4 | 2 |
| Boosting (Round 3) | 4 | 4 | 8 | 10 | 4 | 5 | 4 | 6 | 3 | 4 |

# Boosting (2)

- **AdaBoost**
  - Initially, set uniform weights on all the records
  - At each round
    - Create a bootstrap sample based on the weights
    - Train a classifier on the sample and apply it on the original training set
    - Records that are wrongly classified will have their weights increased
    - Records that are classified correctly will have their weights decreased
    - If the error rate is higher than 50%, start over
  - Final prediction is weighted average of all the classifiers with weight representing the training accuracy

# Boosting (3)

- **Determine the weight**
  - For classifier *i*, its error is

$$\varepsilon_i = \frac{\sum_{j=1}^{N} w_j \delta(C_i(x_j) \neq y_j)}{\sum_{j=1}^{N} w_j}$$

  - The classifier's importance is represented as:

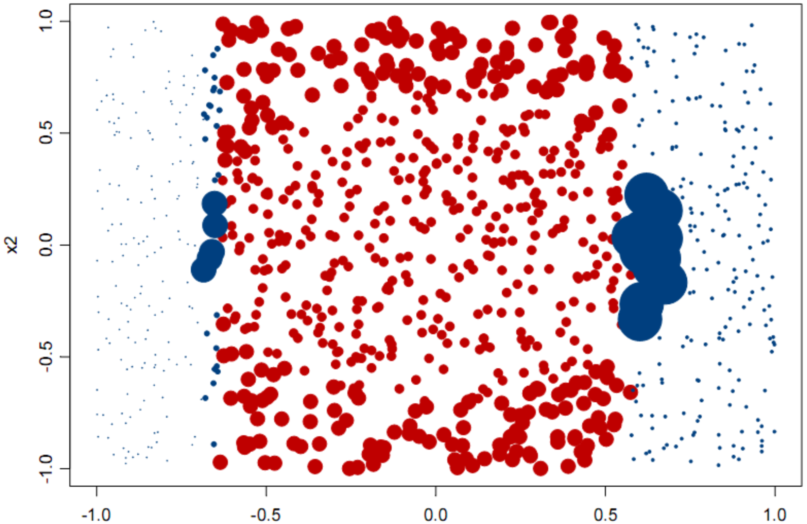
$$\alpha_i = \frac{1}{2} \ln\left(\frac{1-\varepsilon_i}{\varepsilon_i}\right)$$

  - The weight of each record is updated as:

$$w_j^{(i+1)} = \frac{w_j^{(i)} \exp\left(-\alpha_i y_j C_i(x_j)\right)}{Z^{(i)}}$$

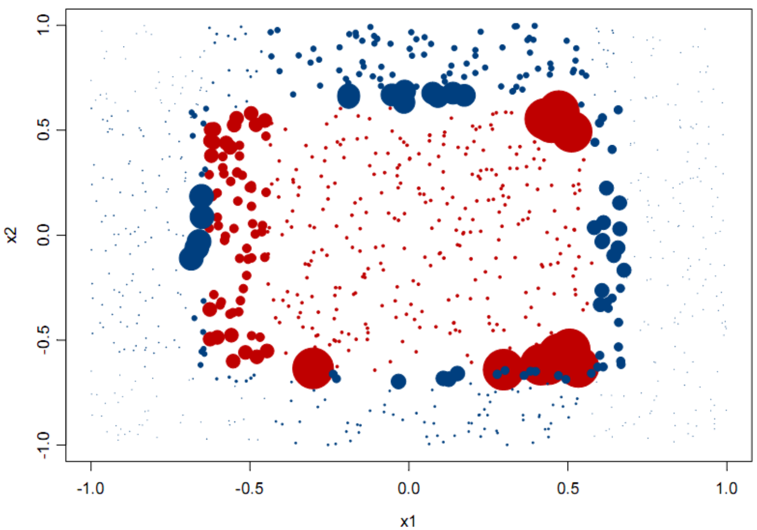
  - Final combination:

$$C^*(x) = \arg\max_y \sum_{i=1}^{K} \alpha_i \delta\left(C_i(x) = y\right)$$

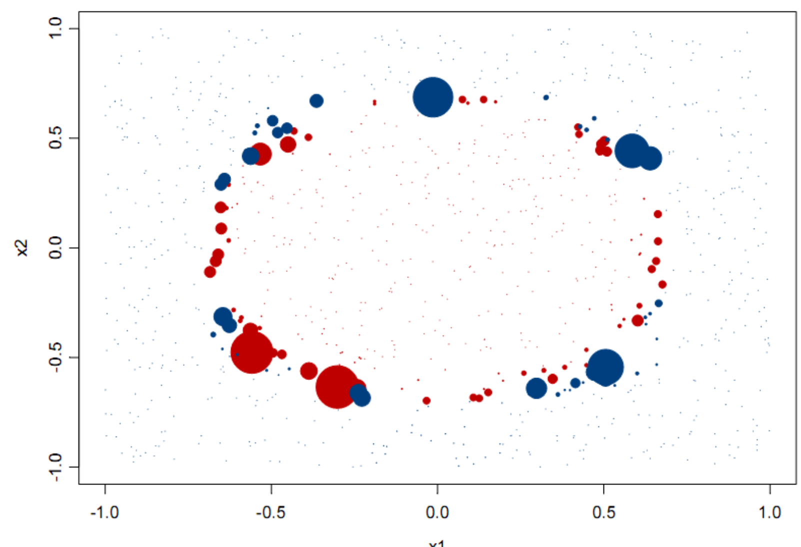



**Classifications (colors) and Weights (size) after *1 iteration* Of AdaBoost**

***3 iterations***




***20 iterations***



# Boosting (4)

- **Explanation**
  - Among the classifiers of the form:

$$f(x) = \sum_{i=1}^{K} \alpha_i C_i(x)$$

  - We seek to minimize the exponential loss function:

$$\sum_{j=1}^{N} \exp\left(-y_j f(x_j)\right)$$

  - Not robust in noisy settings

# Random Forests (1)

- **Algorithm**
  - Choose *T*—number of trees to grow
  - Choose *m<M* (M is the number of total features) —number of features used to calculate the best split at each node (typically 20%)
  - For each tree
    - Choose a training set by choosing *N* times (*N* is the number of training examples) with replacement from the training set
    - For each node, randomly choose *m* features and calculate the best split
    - Fully grown and not pruned
  - Use majority voting among all the trees

# Random Forests (2)

- **Discussions**
  - Bagging+random features
  - Improve accuracy
    - Incorporate more diversity and reduce variances
  - Improve efficiency
    - Searching among subsets of features is much faster than searching among the complete set

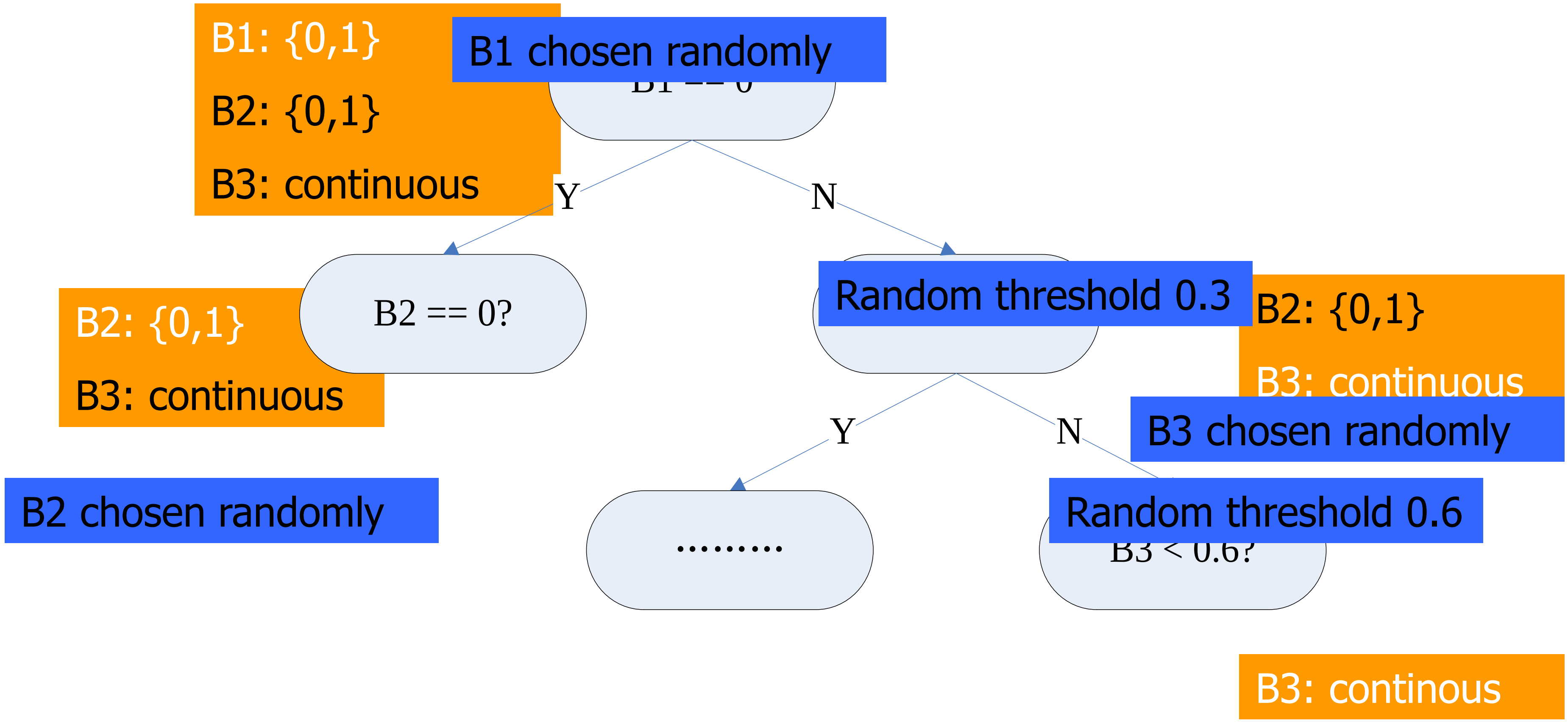
# Random Decision Tree (1)

- **Single-model learning algorithms**
  - Fix structure of the model, minimize some form of errors, or maximize data likelihood (eg., Logistic regression, Naive Bayes, etc.)
  - Use some “free-form” functions to match the data given some “preference criteria” such as information gain, gini index and MDL. (eg., Decision Tree, Rule-based Classifiers, etc.)
- **Such methods will make mistakes if**
  - Data is insufficient
  - Structure of the model or the preference criteria is inappropriate for the problem
- **Learning as Encoding**
  - Make no assumption about the true model, neither parametric form nor free form
  - Do not prefer one base model over the other, just average them

# Random Decision Tree (2)

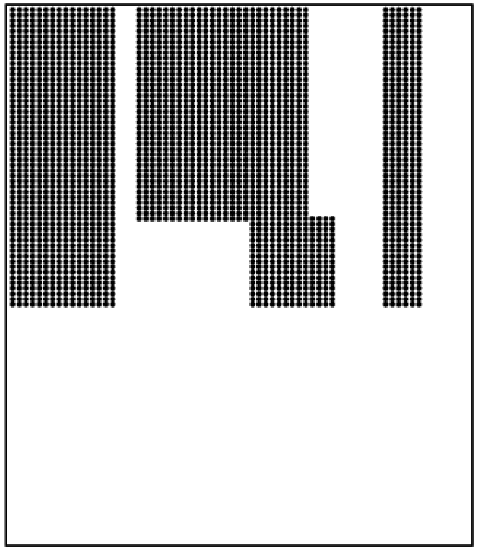
- **Algorithm**
  - At each node,  an un-used feature is chosen randomly
    - A discrete feature is un-used if it has never been chosen previously on a given decision path starting from the root to the current node.
    - A continuous feature can be chosen multiple times on the same decision path, but each time a different threshold value is chosen
  - We stop when one of the following happens:
    - A node becomes too small (<= 3 examples).
    - Or the total height of the tree exceeds some limits, such as the total number of features.
  - Prediction
    - Simple averaging over multiple trees

# Random Decision Tree (3)

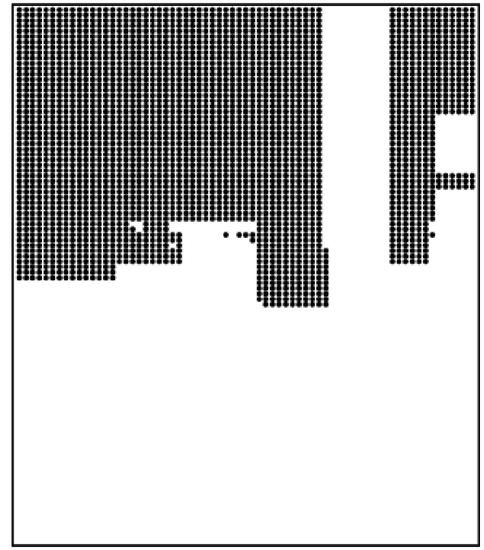


# Random Decision Tree (4)

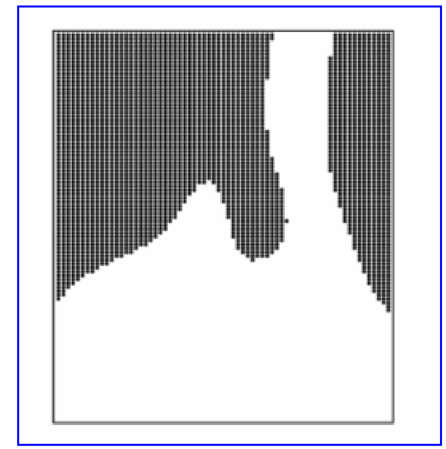
- **Advantages**
  - Training can be very efficient. Particularly true for very large datasets.
    - No cross-validation based estimation of parameters for some parametric methods.
  - Natural multi-class probability.
  - Imposes very little about the structures of the model.

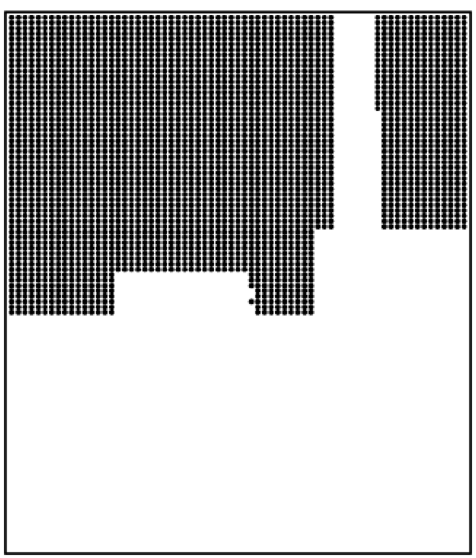


(a) unpruned C4.5

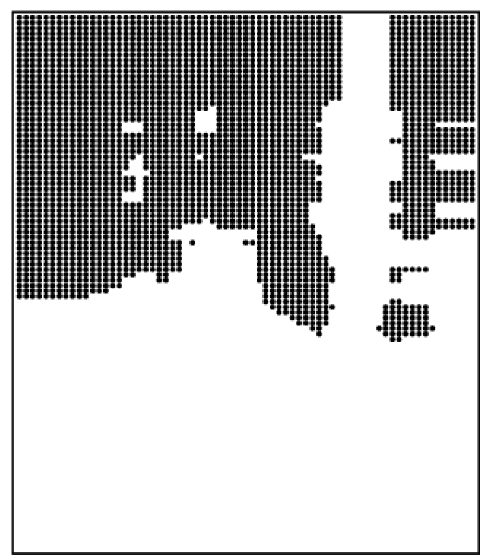


(b) Bagging

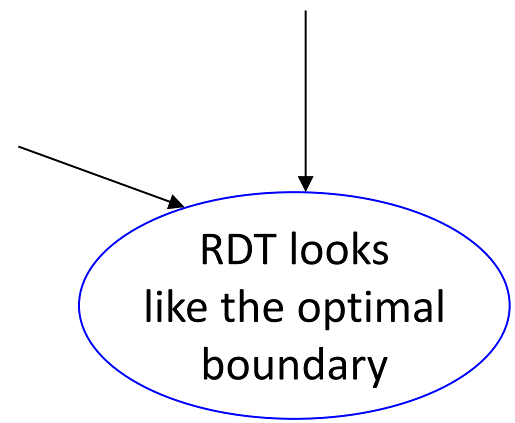




(c) Random Forests

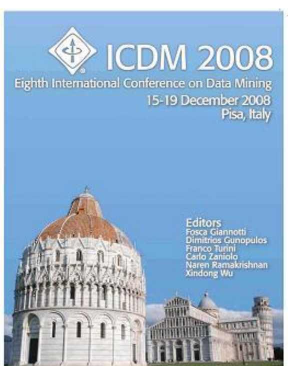


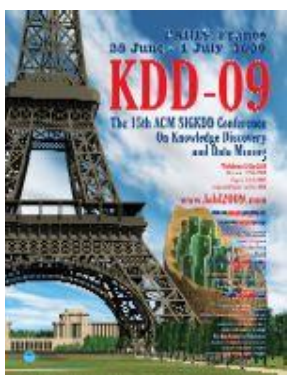
(d) Complete-random tree ensemble



# Ensemble Learning--Stories of Success


- **Million-dollar prize**
  - Improve the baseline movie recommendation approach of Netflix by 10% in accuracy
  - The top submissions all combine several teams and algorithms as an ensemble




- **Data mining competitions**
  - Classification problems
  - Winning teams employ an ensemble of classifiers

# Netflix Prize

- Supervised learning task
  - Training data is a set of users and ratings (1,2,3,4,5 stars) those users have given to movies.
  - Construct a classifier that given a user and an unrated movie, correctly classifies that movie as either 1, 2, 3, 4, or 5 stars
  - $1 million prize for a 10% improvement over Netflix's current movie recommender
- Competition
  - At first, single-model methods are developed, and performance is improved
  - However, improvement slowed down
  - Later, individuals and teams merged their results, and significant improvement is observed

# Leaderboard

| Rank | Team Name | Best Test Score | % Improvement | Best Submit Time |
|---|---|---|---|---|
| **Grand Prize - RMSE = 0.8567 - Winning Team: BellKor's Pragmatic Chaos** | | | | |
| 1 | BellKor's Pragmatic Chaos | 0.8567 | 10.06 | 2009-07-26 18:18:28 |
| 2 | The Ensemble | 0.8567 | 10.06 | 2009-07-26 18:38:22 |
| 3 | Grand Prize Team | 0.8582 | 9.90 | 2009-07-10 21:24:40 |
| 4 | Opera Solutions and Vandelay United | 0.8588 | 9.84 | 2009-07-10 01:12:31 |
| 5 | Vandelay Industries ! | 0.8591 | 9.81 | 2009-07-10 00:32:20 |
| 6 | PragmaticTheory | 0.8594 | 9.77 | 2009-06-24 12:06:56 |
| 7 | BellKor in BigChaos | 0.8601 | 9.70 | 2009-05-13 08:14:09 |
| 8 | Dace_ | 0.8612 | 9.59 | 2009-07-24 17:18:43 |
| 9 | Feeds2 | 0.8622 | 9.48 | 2009-07-12 13:11:51 |
| 10 | BigChaos | 0.8623 | 9.47 | 2009-04-07 12:33:59 |
| 12 | BellKor | 0.8624 | 9.46 | 2009-07-26 17:19:11 |
| **Progress Prize 2008 - RMSE = 0.8627 - Winning Team: BellKor in BigChaos** | | | | |
| 13 | xiangliang | 0.8642 | 9.27 | 2009-07-15 14:53:22 |
| 14 | Gravity | 0.8643 | 9.26 | 2009-04-22 18:31:32 |
| 15 | Ces | 0.8651 | 9.18 | 2009-06-21 19:24:53 |
| **Progress Prize 2007 - RMSE = 0.8723 - Winning Team: KorBell** | | | | |
| **Cinematch score - RMSE = 0.9525** | | | | |

**"Our final solution (RMSE=0.8712) consists of blending 107 individual results. "**

**"Predictive accuracy is substantially improved when blending multiple predictors. Our experience is that most efforts should be concentrated in deriving substantially different approaches, rather than refining a single technique. "**

# Take-away Message

- Various classification approaches
  - how they work
  - their strengths and weakness
- Algorithms
  - Decision tree
  - K nearest neighbors
  - Naive Bayes
  - Logistic regression
  - Rule-based classifier
  - SVM
  - Ensemble method