POSTECH
POHANG UNIVERSITY OF SCIENCE AND TECHNOLOGY

Week 8-2

Machine Learning 2: Naïve Bayes, Rule Learning, Lazy Learning

Big Data

Prof. Hwanjo Yu
POSTECH

POSTECH
POHANG UNIVERSITY OF SCIENCE AND TECHNOLOGY

Bayesian Classification

Big Data

Bayesian theorem: Basics

- Let X be a data sample ("*evidence*"): class label is unknown
- Let H be a *hypothesis* that X belongs to class C
- Classification is to determine P(H|X), (*posteriori probability*)
  - the probability that the hypothesis holds given the observed data sample X
- P(H) (*prior probability*), the initial probability
  - E.g. X will buy computer, regardless of age, income, …
- P(X): probability that sample data is observed
- P(X|H) (*likelihood*), the probability of observing the sample X, given that the hypothesis holds
  - E.g. Given that X will buy computer, the prob. that X is 31..40, medium income

Bayesian theorem: Basics

- Given training data $\mathbf{X}$, *posteriori probability* of a hypothesis H, $\mathrm{P}(\mathrm{H}|\mathbf{X})$, follows the Bayes theorem

$$P(H|\mathbf{X}) = \frac{P(\mathbf{X}|H)P(H)}{P(\mathbf{X})}$$

- Informally, this can be written as

$$\text{posteriori} = \text{likelihood x prior/evidence}$$

- Predicts $\mathbf{X}$ belongs to C_2 iff the probability $\mathrm{P}(\mathrm{C_i}|\mathbf{X})$ is the highest among all the $\mathrm{P}(\mathrm{C_k}|\mathbf{X})$ for all the k classes

- Practical difficulty: require initial knowledge of many probabilities, significant computational cost

Naïve Bayesian classifier

- A simplified assumption:

  attributes are conditionally independent (i.e. no dependence relation between attributes):

$$P(\mathbf{X}|C_i) = P(x_1 x_2 \ldots x_k|C_i) = \prod_{k=1}^{n} p(x_k|C_i) = P(x_1|C_i) \times P(x_2|C_i) \times \cdots \times P(x_n|C_i)$$

- This greatly reduces the computation cost: Only counts the class distribution
- If A_k is categorical, $P(x_k|C_i)$ is the # of tuples in C_i having value x_k for A_k divided by $|C_{i,D}|$ (# of tuples of C_i in D)
- If A_k is continuous-valued, $P(x_k|C_i)$ is usually computed based on Gaussian distribution with a mean μ and standard deviation σ

$$g(x, \mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

  and $P(x_k|C_i)$ is $P(\mathbf{X}|C_i) = g(x_k, \mu_{C_i}, \sigma_{C_i})$

Naïve Bayesian classifier: Training dataset

Class:
C1:buys_computer = ‘yes’
C2:buys_computer = ‘no’

Data sample:
X = (age <=30, Income = medium,
Student = yes, Credit_rating = Fair)

age	income	student	credit_ratin	buys_
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

Naïve Bayesian classifier: An example

$P(C_i)$: P(buys_computer = "yes") = 9/14 = 0.643
P(buys_computer = "no") = 5/14= 0.357

Compute $P(\mathbf{X}|C_i)$ for each class
P(age = "<=30" | buys_computer = "yes") = 2/9 = 0.222
P(age = "<= 30" | buys_computer = "no") = 3/5 = 0.6
P(income = "medium" | buys_computer = "yes") = 4/9 = 0.444
P(income = "medium" | buys_computer = "no") = 2/5 = 0.4
P(student = "yes" | buys_computer = "yes) = 6/9 = 0.667
P(student = "yes" | buys_computer = "no") = 1/5 = 0.2
P(credit_rating = "fair" | buys_computer = "yes") = 6/9 = 0.667
P(credit_rating = "fair" | buys_computer = "no") = 2/5 = 0.4

$\mathbf{X}$ = (age <= 30 , income = medium, student = yes, credit_rating = fair)

$P(\mathbf{X}|C_i)$: P($\mathbf{X}$|buys_computer = "yes") = 0.222 x 0.444 x 0.667 x 0.667 = 0.044
P(X|buys_computer = "no") = 0.6 x 0.4 x 0.2 x 0.4 = 0.019
$P(\mathbf{X}|C_i)*P(C_i)$: P($\mathbf{X}$|buys_computer = "yes") * P(buys_computer = "yes") = 0.028
P($\mathbf{X}$|buys_computer = "no") * P(buys_computer = "no") = 0.007

Therefore, $\mathbf{X}$ belongs to class ("buys_computer = yes")

age	income	student	credit_ratin	buys_
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

Avoiding the 0-probability problem

- Naïve Bayesian prediction requires each conditional prob. be non-zero. Otherwise, the predicted prob. will be zero

$$P(\mathbf{X}|C_i) = \prod_{k=1}^{n} P(x_k|C_i)$$

- E.g. Suppose a dataset with 1000 tuples, income=low (0), income= medium (990), and income = high (10).
- Smoothing: e.g. use Laplacian correction (or Laplacian estimator)
  - Adding 1 to each case
    - Prob(income = low) = 1/1003
    - Prob(income = medium) = 991/1003
    - Prob(income = high) = 11/1003

Naïve Bayesian classifier: Comments

- Advantages
  - Easy to implement
  - Good results obtained in most of the cases
- Disadvantages
  - Assumption: class conditional independence, therefore loss of accuracy
  - Practically, dependencies exist among variables
    - E.g. Symptoms: fever, cough etc., Disease: lung cancer, diabetes, etc.
    - Dependencies among these cannot be modeled by Naïve Bayesian Classifier

Rule-based Classification

Big Data

Using IF-THEN rules for classification

- Represent the knowledge in the form of IF-THEN rules

  R: IF *age* = youth AND *student* = yes THEN *buys_computer* = yes

  - Rule antecedent/precondition vs. rule consequent

- Assessment of a rule: *coverage and accuracy*

  - n_{covers} = # of tuples covered by R
  - $n_{correct}$ = # of tuples correctly classified by R

  coverage(R) = n_{covers} /|D| /* D: training data set */

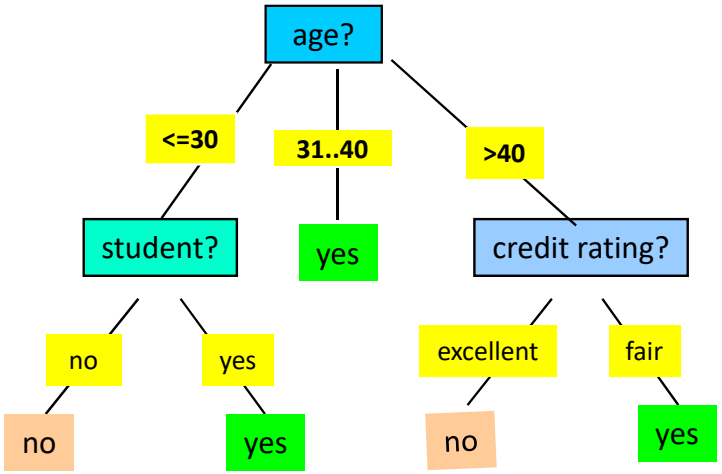
  accuracy(R) = $n_{correct}$ / n_{covers}

- If more than one rule are triggered, need conflict resolution

  - Size ordering: assign the highest priority to the triggering rules that has the "toughest" requirement (i.e. with the *most attribute test*)
  - Class-based ordering: decreasing order of *prevalence or misclassification cost per class*
  - Rule-based ordering (decision list): rules are organized into one long priority list, according to some measure of rule quality or by experts

Rule extraction from a decision tree

- Rules are easier to understand than large trees
- One rule is created for each path from the root to a leaf
- Each attribute-value pair along a path forms a conjunction: the leaf holds the class prediction
- Rules are mutually exclusive and exhaustive
- Example: Rule extraction from our *buys_computer* decision-tree



IF *age* = young AND *student* = no THEN *buys_computer* = no

IF *age* = young AND *student* = yes THEN *buys_computer* = yes

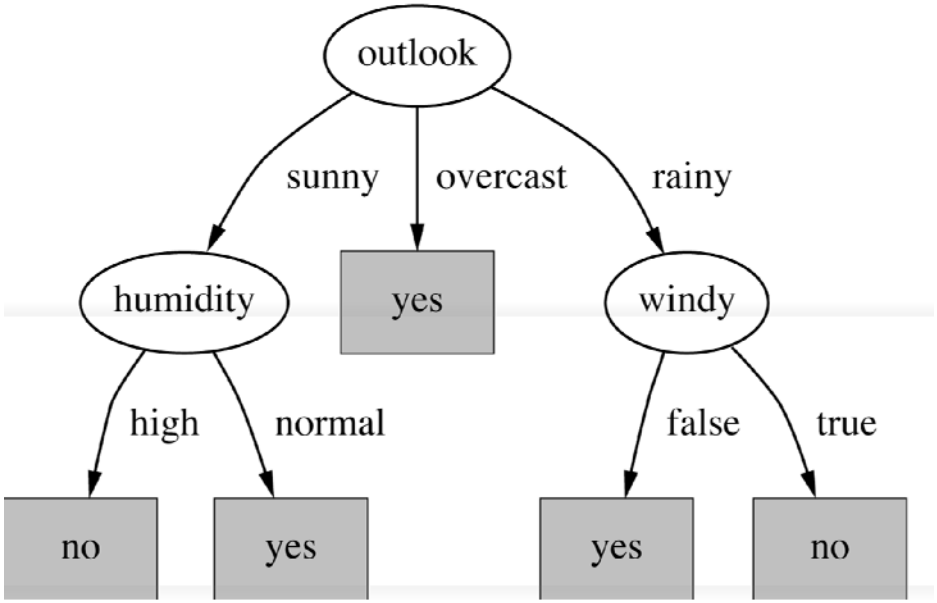
IF *age* = mid-age THEN *buys_computer* = yes

IF *age* = old AND *credit_rating* = *excellent* THEN *buys_computer* = yes

IF *age* = young AND *credit_rating* = *fair* THEN *buys_computer* = no

Decision trees and rules

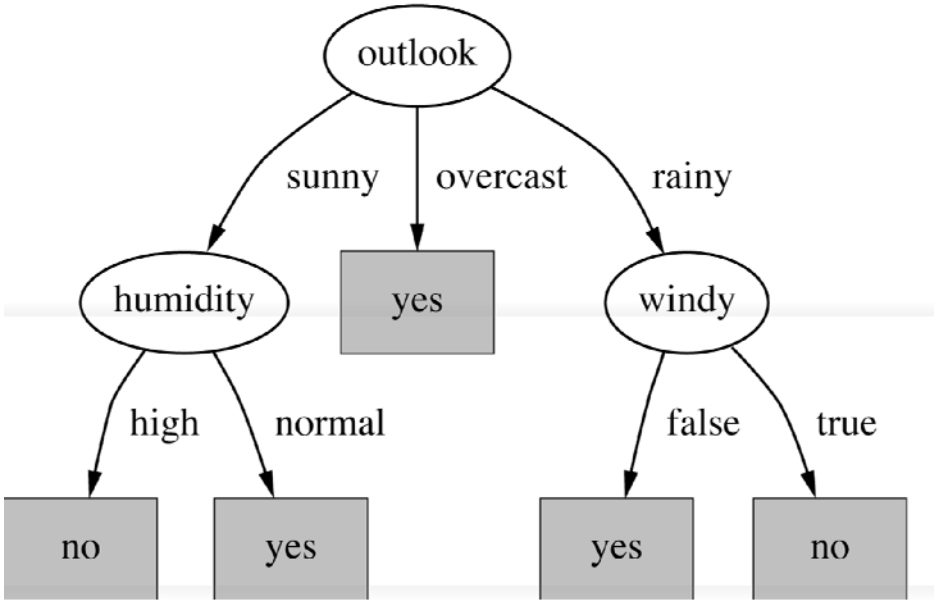
For any decision tree you can read off an equivalent set of *ordered* rules



- If outlook = sunny and humidity = high then no
- If outlook = sunny and humidity = normal then yes
- If outlook = overcast then yes
- if outlook = rainy and windy = false then yes
- if outlook = rainy and windy = true then no

Decision trees and rules

For any decision tree you can read off an equivalent set of *ordered* rules



- If outlook = sunny and humidity = high then no
- ~~If outlook = sunny and humidity = normal then yes~~
- ~~If outlook = overcast then yes~~
- ~~if outlook = rainy and windy = false then yes~~
- if outlook = rainy and windy = true then no

=

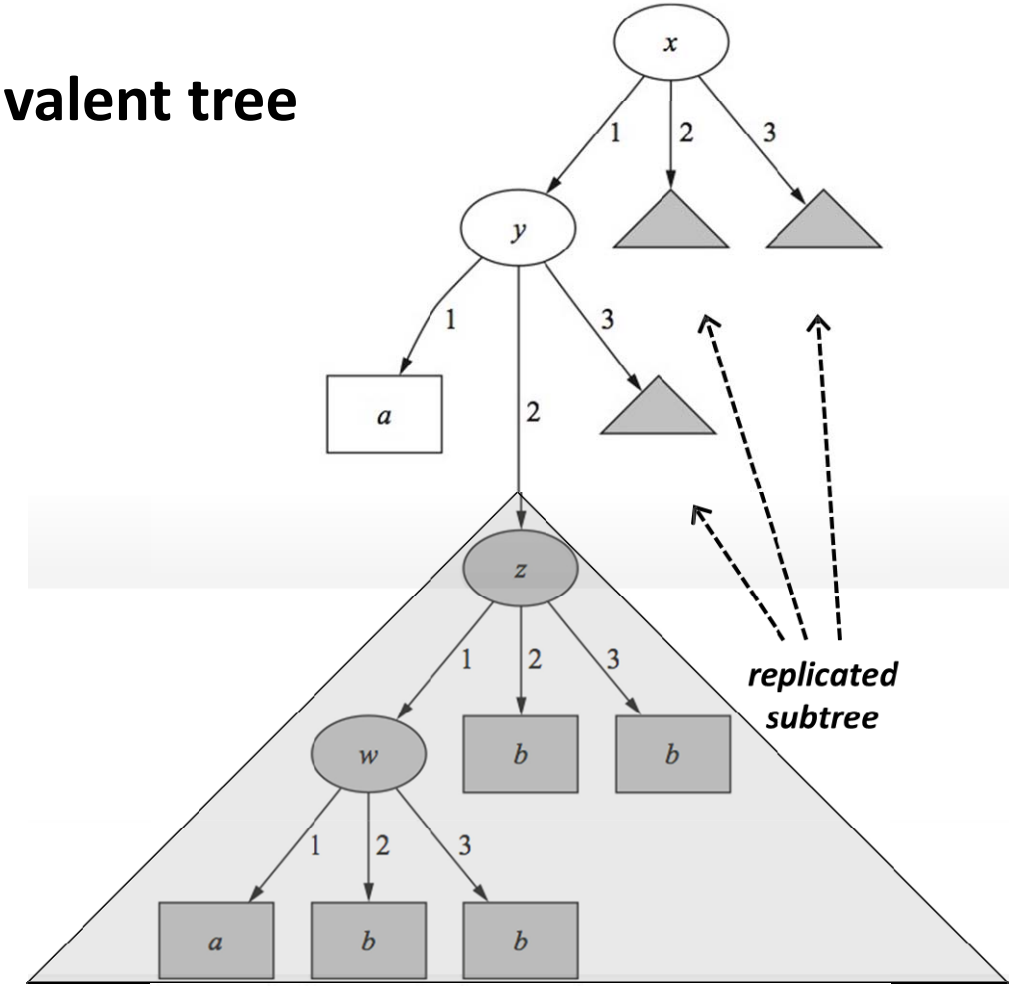
- If outlook = sunny and humidity = high then no
- If outlook = rainy and windy = true then no
- Otherwise yes

Decision trees and rules

For any set of rules there is an equivalent tree

but it might be very complex

if x = 1 and y = 1 then a
if z = 1 and w = 1 then a
otherwise b



Decision trees and rules

- Theoretically, rules and trees have equivalent “descriptive power”
- But practically they are very different

  … because rules are usually expressed as a decision list, to be executed sequentially, in order, until one “fires”

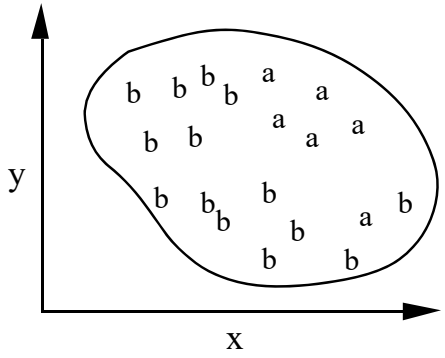
- People like rules: they’re easy to read and understand
- It’s tempting to view them as independent “nuggets of knowledge”
- … but that’s misleading

  when rules are executed sequentially, each one must be interpreted in the context of its predecessors

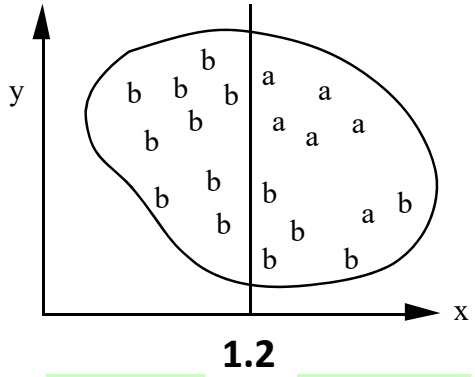
Decision trees and rules

Generating a rule

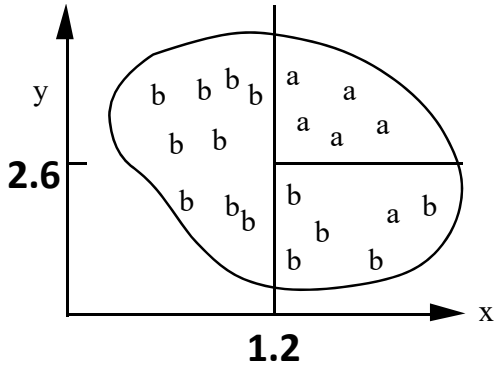
- Generating a rule for class *a*



if *true*
then class = *a*



if x > 1.2
then class = *a*



if x > 1.2 and y > 2.6
then class = *a*

- Possible rule set for class *b*:

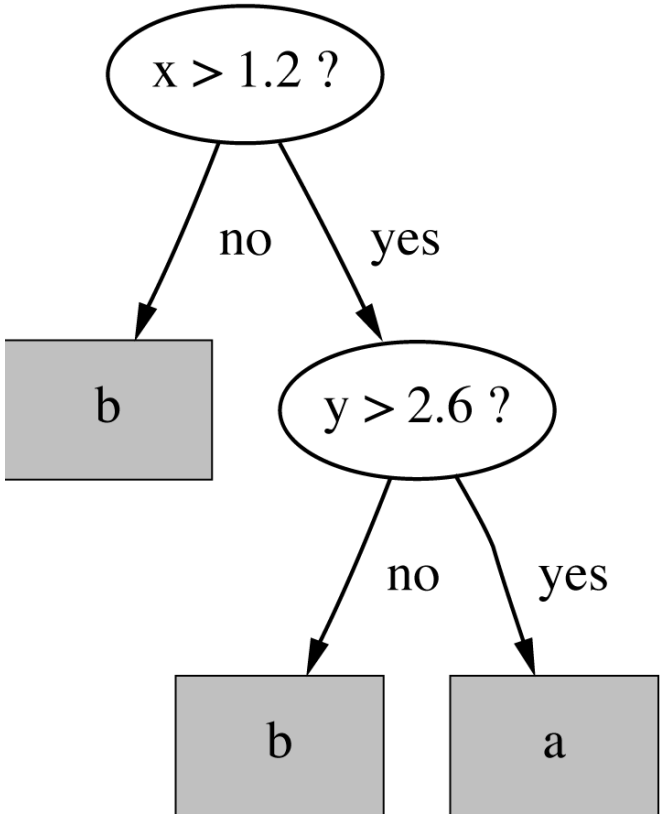
if x ≤ 1.2 then class = *b*
if ~~x > 1.2 and~~ y ≤ 2.6 then class = *b*

- Could add more rules, get "perfect" rule set

Decision trees and rules

Rules vs. trees

- Corresponding decision tree
  - produces exactly the same predictions
- Rule sets can be more perspicuous
  - E.g. when decision trees contain replicated subtrees
- Also: in multiclass situations,
  - covering algorithm concentrates on one class at a time
  - decision tree learner takes all classes into account



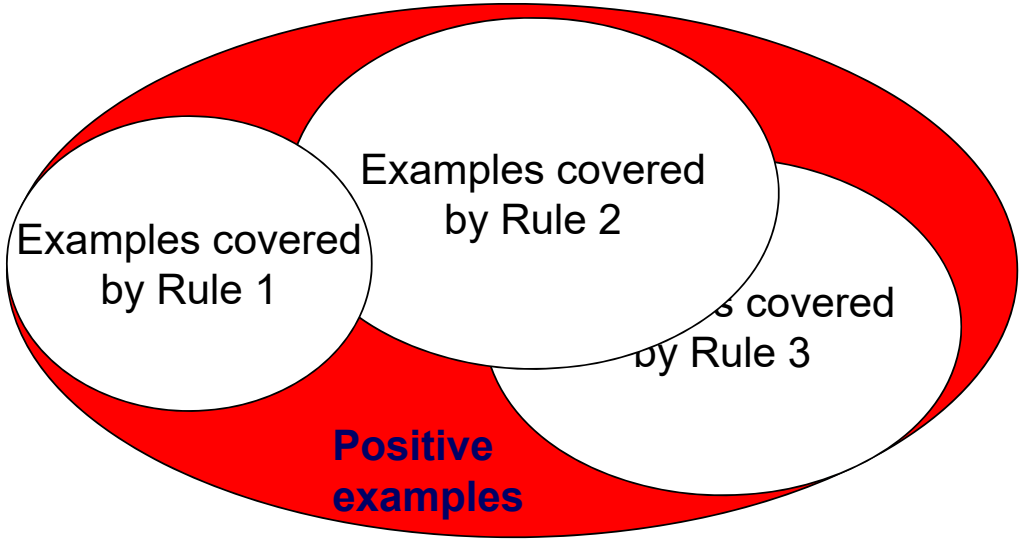
Rule induction: Sequential covering method

- Sequential covering algorithm: Extracts rules directly from training data
- Typical sequential covering algorithms: FOIL, RIPPER
- Rules are learned sequentially, each for a given class C_i will cover many tuples of C_i but none (or few) of the tuples of other classes
- Steps:
  - Rules are learned one at a time
  - Each time a rule is learned, the tuples covered by the rules are removed
  - The process repeats on the remaining tuples unless termination condition, e.g. when no more training examples or when the quality of a rule returned is below a user-specified threshold
- Comp. w. decision-tree induction: learning a set of rules *simultaneously*

Rule induction: Sequential covering method

while (enough target tuples left)

- **generate a rule**
- **remove positive target tuples satisfying this rule**



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 (in FOIL & RIPPER): assesses info_gain by extending condition

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

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

- Rule pruning based on an independent set of test tuples

$$FOIL\_Prune(R) = \frac{pos - neg}{pos + neg}$$

Pos/neg are # of positive/negative tuples covered by R.

If FOIL_Prune is higher for the pruned version of R, prune R

Rule generation

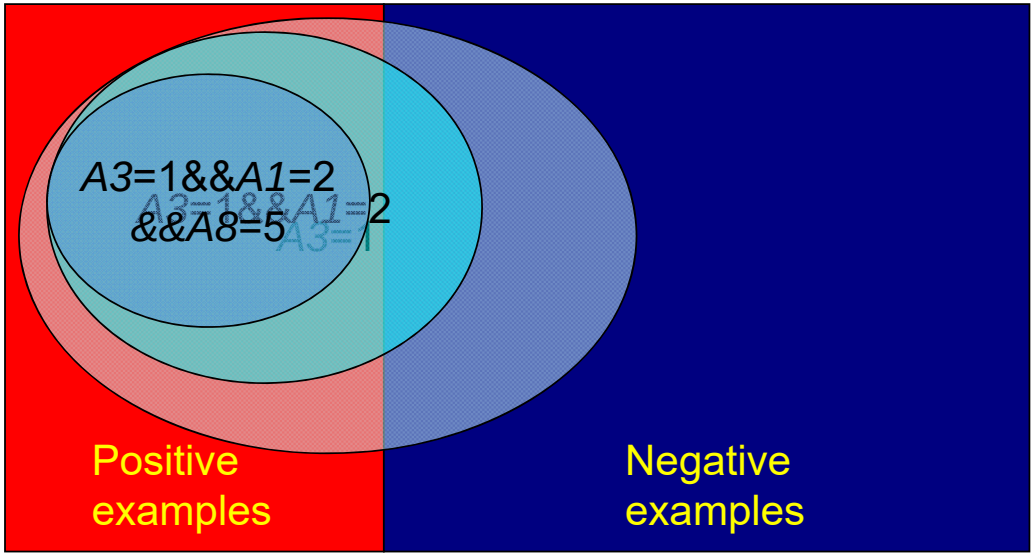
- **To generate a rule**

  while(true)

  find the best predicate p

  if foil-gain(p) > threshold then add p to current rule

  else break



Lazy Learner (e.g. kNN)

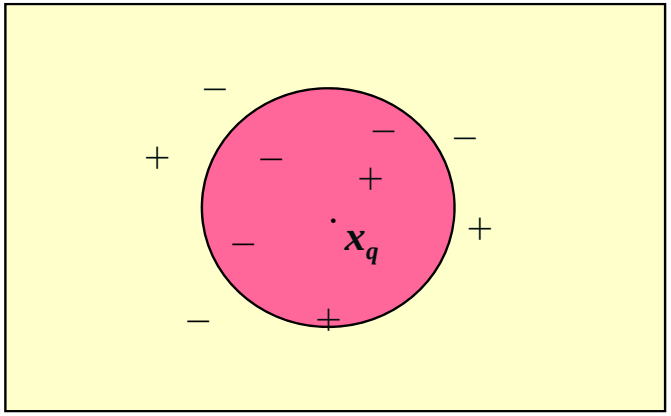
Big Data

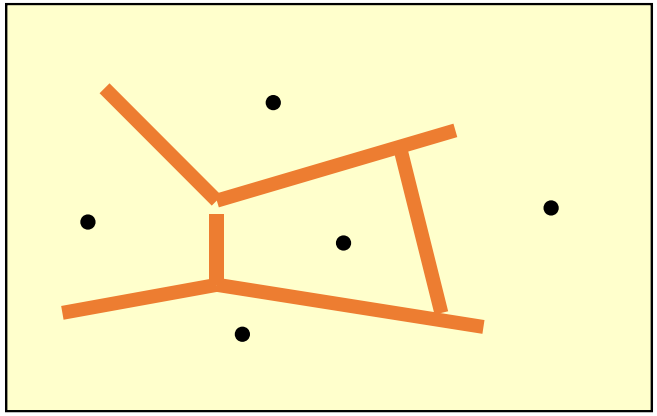
Lazy vs. Eager learning

- Lazy learning (e.g. instance-based learning):
  - Simply stores all the training data
  - Classify based on k nearest neighbors
  - Fast in training but slow in prediction
- The others:
  - Constructs a model from training data
  - Classify using the model
  - Slow in training but fast in prediction

k-nearest neighbor (kNN) algorithm

- The nearest neighbors are defined in terms of Euclidean distance, dist(X_1, X_2)
- Given a testing instance x, find kNN of x, and take the majority class of kNNs to classify x
- Vonoroi diagram: the decision surface induced by 1-NN





k-nearest neighbor (kNN) algorithm

- k-NN for real-valued prediction for a given unknown tuple
  - Returns the mean values of the *k* nearest neighbors
- Distance-weighted nearest neighbor algorithm
  - Weight the contribution of each of the k neighbors according to their distance to the query x_q
    - Give greater weight to closer neighbors $w \equiv \frac{1}{d(x_q, x_i)^2}$
- Robust to noisy data by averaging k-nearest neighbors
- Too small k => overfitting
- Too large k => underfitting