Chapter 3: Supervised Learning

Road Map

- **Basic concepts**
- Decision tree induction
- Evaluation of classifiers
- Rule induction
- Classification using association rules
- Naïve Bayesian classification
- Naïve Bayes for text classification
- Support vector machines
- K-nearest neighbor
- Ensemble methods: Bagging and Boosting
- Summary

An example application

- An emergency room in a hospital measures 17 variables (e.g., blood pressure, age, etc) of newly admitted patients.
- A decision is needed: whether to put a new patient in an intensive-care unit.
- Due to the high cost of ICU, those patients who may survive less than a month are given higher priority.
- Problem: to predict high-risk patients and discriminate them from low-risk patients.

Another application

- A credit card company receives thousands of applications for new cards. Each application contains information about an applicant,
  - age
  - Marital status
  - annual salary
  - outstanding debts
  - credit rating
  - etc.
- Problem: to decide whether an application should approved, or to classify applications into two categories, approved and not approved.

Machine learning and our focus

- Like human learning from past experiences.
- A computer does not have “experiences”.
- A computer system learns from data, which represent some “past experiences” of an application domain.
- Our focus: learn a target function that can be used to predict the values of a discrete class attribute, e.g., approve or not-approved, and high-risk or low risk.
- The task is commonly called: Supervised learning, classification, or inductive learning.

The data and the goal

- Data: A set of data records (also called examples, instances or cases) described by
  - *k* attributes: A_1, A_2, … A_k.
  - a class: Each example is labelled with a pre-defined class.
- Goal: To learn a classification model from the data that can be used to predict the classes of new (future, or test) cases/instances.

An example: data (loan application)

Approved or not

ID	Age	Has_Job	Own_House	Credit_Rating	Class
1	young	false	false	fair	No
2	young	false	false	good	No
3	young	true	false	good	Yes
4	young	true	true	fair	Yes
5	young	false	false	fair	No
6	middle	false	false	fair	No
7	middle	false	false	good	No
8	middle	true	true	good	Yes
9	middle	false	true	excellent	Yes
10	middle	false	true	excellent	Yes
11	old	false	true	excellent	Yes
12	old	false	true	good	Yes
13	old	true	false	good	Yes
14	old	true	false	excellent	Yes
15	old	false	false	fair	No

An example: the learning task

- Learn a classification model from the data
- Use the model to classify future loan applications into
  - Yes (approved) and
  - No (not approved)
- What is the class for following case/instance?

Age	Has_Job	Own_house	Credit-Rating	Class
young	false	false	good	?

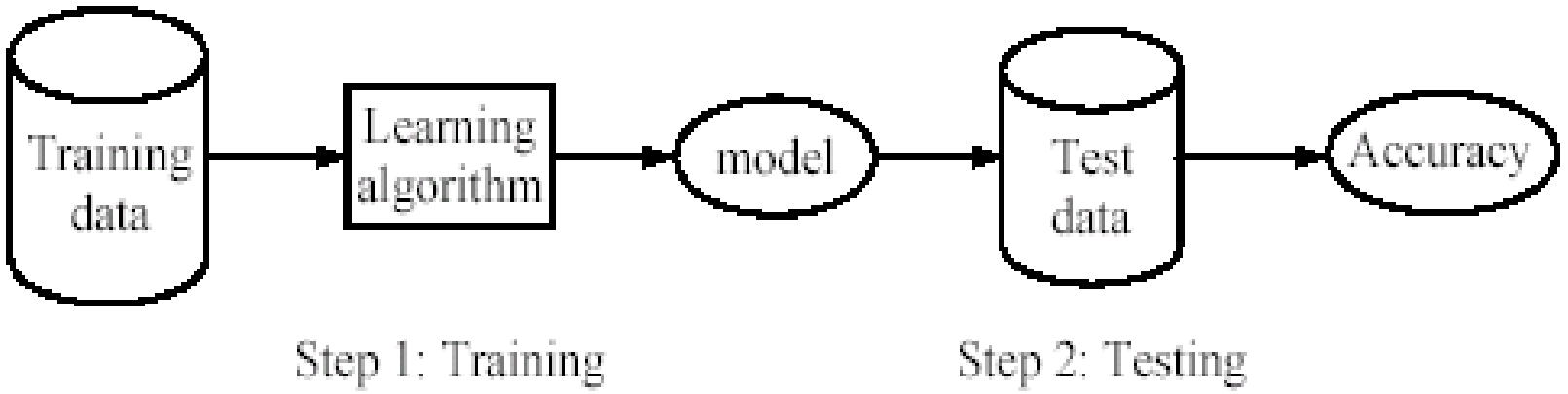
Supervised vs. unsupervised Learning

- Supervised learning: classification is seen as supervised learning from examples.
  - Supervision: The data (observations, measurements, etc.) are labeled with pre-defined classes. It is like that a “teacher” gives the classes (supervision).
  - Test data are classified into these classes too.
- Unsupervised learning (clustering)
  - Class labels of the data are unknown
  - Given a set of data, the task is to establish the existence of classes or clusters in the data

Supervised learning process: two steps

- Learning (training): Learn a model using the training data
- Testing: Test the model using unseen test data to assess the model accuracy

$$Accuracy = \frac{\text{Number of correct classifications}}{\text{Total number of test cases}},$$



What do we mean by learning?

- Given
  - a data set *D*,
  - a task *T*, and
  - a performance measure *M*,

  a computer system is said to **learn** from *D* to perform the task *T* if after learning the system's performance on *T* improves as measured by *M*.
- In other words, the learned model helps the system to perform *T* better as compared to no learning.

An example

- Data: Loan application data
- Task: Predict whether a loan should be approved or not.
- Performance measure: accuracy.

No learning: classify all future applications (test data) to the majority class (i.e., Yes):

Accuracy = 9/15 = 60%.

- We can do better than 60% with learning.

Fundamental assumption of learning

Assumption: The distribution of training examples is identical to the distribution of test examples (including future unseen examples).

- In practice, this assumption is often violated to certain degree.
- Strong violations will clearly result in poor classification accuracy.
- To achieve good accuracy on the test data, training examples must be sufficiently representative of the test data.

Road Map

- Basic concepts
- **Decision tree induction**
- Evaluation of classifiers
- Rule induction
- Classification using association rules
- Naïve Bayesian classification
- Naïve Bayes for text classification
- Support vector machines
- K-nearest neighbor
- Ensemble methods: Bagging and Boosting
- Summary

Introduction

- Decision tree learning is one of the most widely used techniques for classification.
  - Its classification accuracy is competitive with other methods, and
  - it is very efficient.
- The classification model is a tree, called decision tree.
- C4.5 by Ross Quinlan is perhaps the best known system. It can be downloaded from the Web.

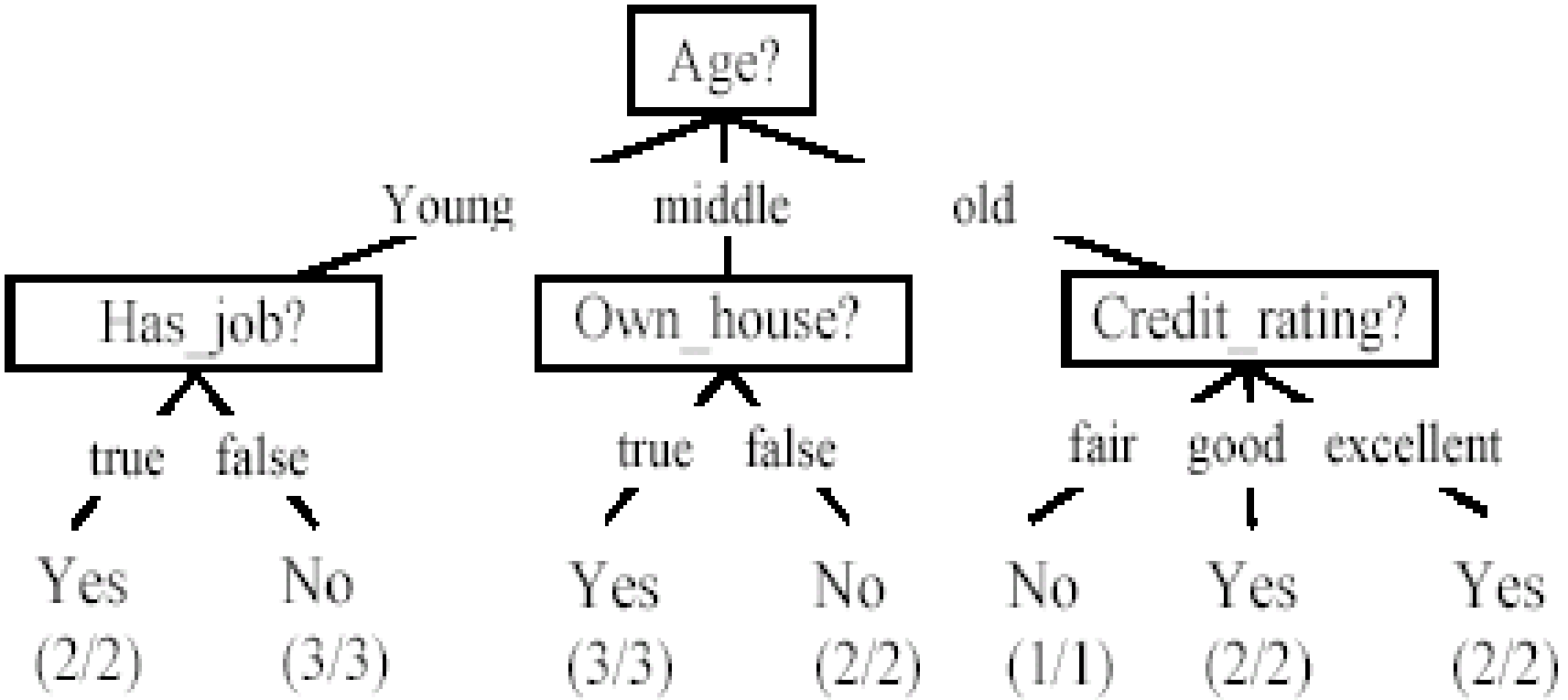
The loan data (reproduced)

Approved or not

ID	Age	Has_Job	Own_House	Credit_Rating	Class
1	young	false	false	fair	**No**
2	young	false	false	good	**No**
3	young	true	false	good	**Yes**
4	young	true	true	fair	**Yes**
5	young	false	false	fair	**No**
6	middle	false	false	fair	**No**
7	middle	false	false	good	**No**
8	middle	true	true	good	**Yes**
9	middle	false	true	excellent	**Yes**
10	middle	false	true	excellent	**Yes**
11	old	false	true	excellent	**Yes**
12	old	false	true	good	**Yes**
13	old	true	false	good	**Yes**
14	old	true	false	excellent	**Yes**
15	old	false	false	fair	**No**

A decision tree from the loan data

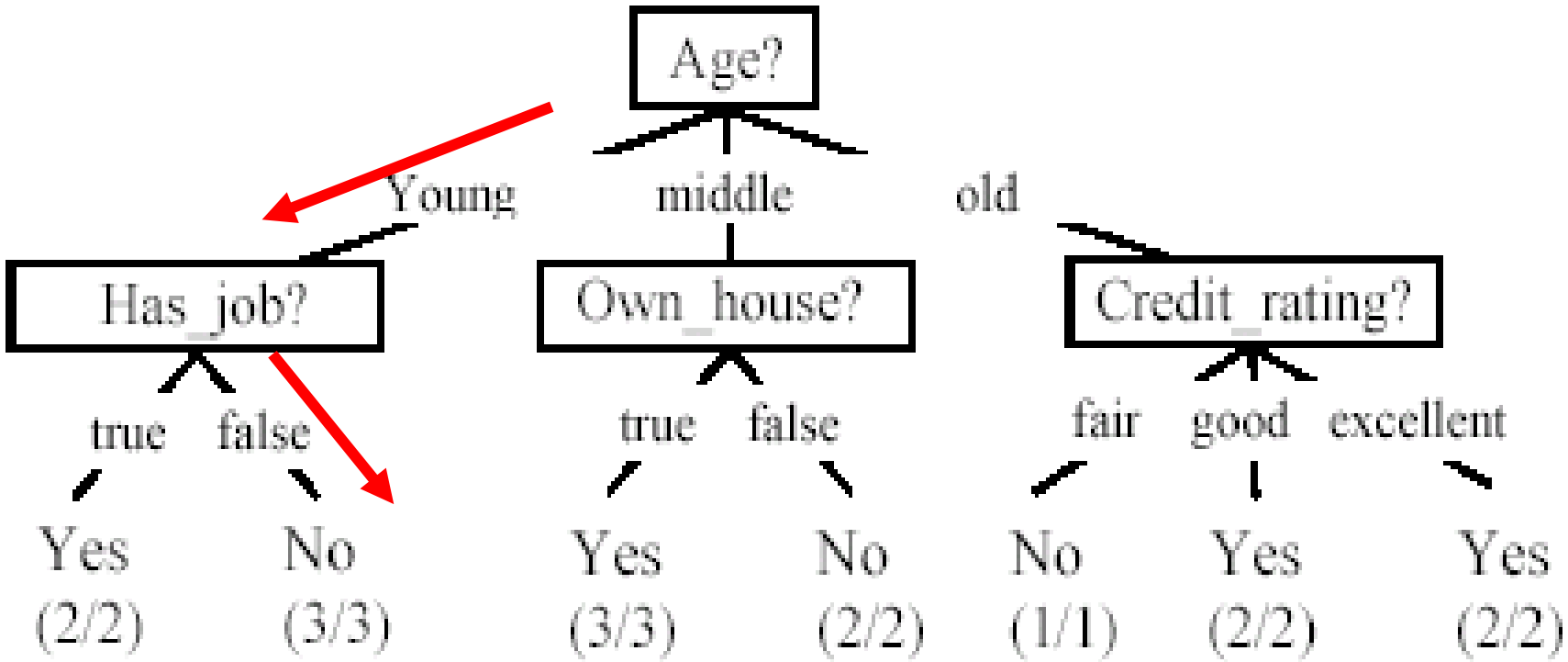
- Decision nodes and leaf nodes (classes)



Use the decision tree

Age	Has_Job	Own_house	Credit-Rating	Class
young	false	false	good	?

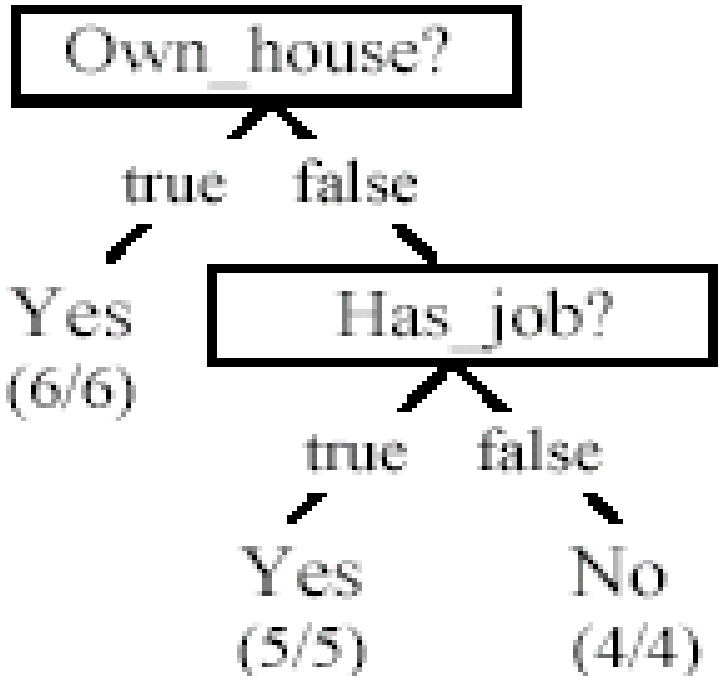
No



Is the decision tree unique?

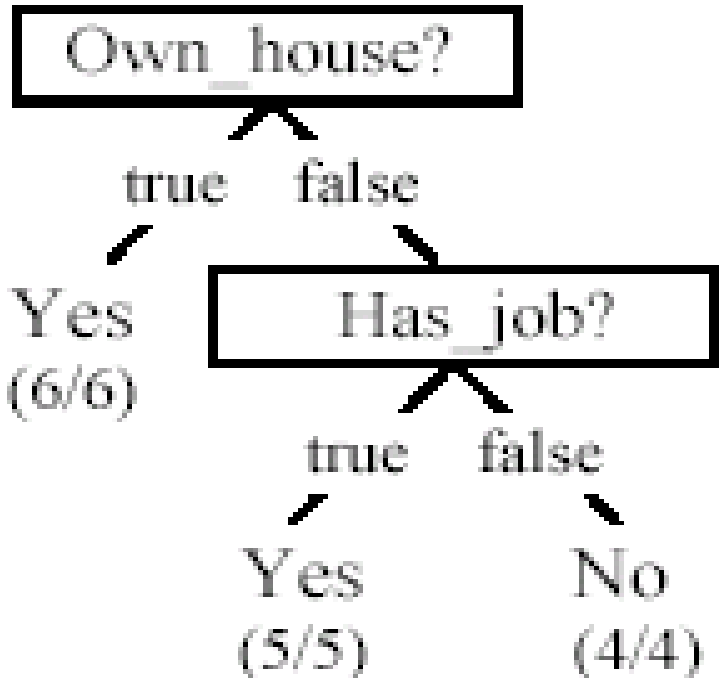
- No. Here is a simpler tree.
- We want smaller tree and accurate tree.
  - Easy to understand and perform better.

- Finding the best tree is NP-hard.
- All current tree building algorithms are heuristic algorithms



From a decision tree to a set of rules

- A decision tree can be converted to a set of rules
- Each path from the root to a leaf is a rule.



Own_house = true → Class =Yes [sup=6/15, conf=6/6]
Own_house = false, Has_job = true → Class = Yes [sup=5/15, conf=5/5]
Own_house = false, Has_job = false → Class = No [sup=4/15, conf=4/4]

Algorithm for decision tree

- Basic algorithm (a greedy **divide-and-conquer** algorithm)
  - Assume attributes are categorical now (continuous attributes can be handled too)
  - Tree is constructed in a top-down recursive manner
  - At start, all the training examples are at the root
  - Examples are partitioned recursively based on selected attributes
  - Attributes are selected on the basis of an impurity function (e.g., information gain)
- Conditions for stopping partitioning
  - All examples for a given node belong to the same class
  - There are no remaining attributes for further partitioning – majority class is the leaf
  - There are no examples left

Decision tree learning al

```
. Algorithm decisionTree(D, A, T)
1     if D contains only training examples of the same class c_j ∈ C then
2         make T a leaf node labeled with class c_j;
3     elseif A = ∅ then
4         make T a leaf node labeled with c_j, which is the most frequent class in D
5     else  // D contains examples belonging to a mixture of classes. We select a single
6           // attribute to partition D into subsets so that each subset is purer
7         p_0 = impurityEval-1(D);
8         for each attribute A_i ∈ {A_1, A_2, ..., A_k} do
9             p_i = impurityEval-2(A_i, D)
10        end
11        Select A_g ∈ {A_1, A_2, ..., A_k} that gives the biggest impurity reduction,
              computed using p_0 − p_i;
12        if p_0 − p_g < threshold then     // A_g does not significantly reduce impurity p_0
13          make T a leaf node labeled with c_j, the most frequent class in D.
14        else                              // A_g is able to reduce impurity p_0
15            Make T a decision node on A_g;
16            Let the possible values of A_g be v_1, v_2, ..., v_m. Partition D into m
                  disjoint subsets D_1, D_2, ..., D_m based on the m values of A_g.
17            for each D_j in {D_1, D_2, ..., D_m} do
18                if D_j ≠ ∅ then
19                    create a branch (edge) node T_j for v_j as a child node of T;
20                    decisionTree(D_j, A-{A_g}, T_j)// A_g is removed
21                end
22            end
23        end
24    end
```

Choose an attribute to partition data

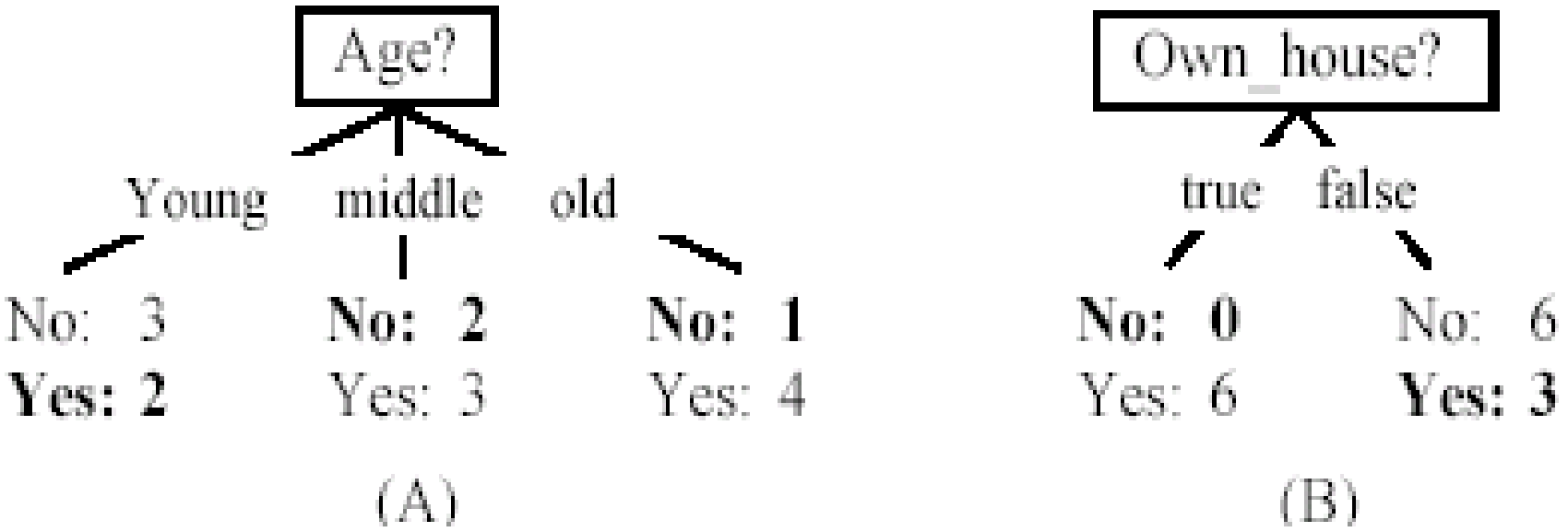
- The *key* to building a decision tree - which attribute to choose in order to branch.
- The objective is to reduce impurity or uncertainty in data as much as possible.
  - A subset of data is pure if all instances belong to the same class.
- The *heuristic* in C4.5 is to choose the attribute with the maximum Information Gain or Gain Ratio based on information theory.

The loan data (reproduced)

Approved or not

ID	Age	Has_Job	Own_House	Credit_Rating	Class
1	young	false	false	fair	**No**
2	young	false	false	good	**No**
3	young	true	false	good	**Yes**
4	young	true	true	fair	**Yes**
5	young	false	false	fair	**No**
6	middle	false	false	fair	**No**
7	middle	false	false	good	**No**
8	middle	true	true	good	**Yes**
9	middle	false	true	excellent	**Yes**
10	middle	false	true	excellent	**Yes**
11	old	false	true	excellent	**Yes**
12	old	false	true	good	**Yes**
13	old	true	false	good	**Yes**
14	old	true	false	excellent	**Yes**
15	old	false	false	fair	**No**

Two possible roots, which is better?



- Fig. (B) seems to be better.

Information theory

- Information theory provides a mathematical basis for measuring the information content.
- To understand the notion of information, think about it as providing the answer to a question, for example, whether a coin will come up heads.
  - If one already has a good guess about the answer, then the actual answer is less informative.
  - If one already knows that the coin is rigged so that it will come with heads with probability 0.99, then a message (advanced information) about the actual outcome of a flip is worth less than it would be for a honest coin (50-50).

Information theory (cont …)

- For a fair (honest) coin, you have no information, and you are willing to pay more (say in terms of $) for advanced information - less you know, the more valuable the information.
- Information theory uses this same intuition, but instead of measuring the value for information in dollars, it measures information contents in **bits**.
- One bit of information is enough to answer a yes/no question about which one has no idea, such as the flip of a fair coin

Information theory: Entropy measure

- The entropy formula,

$$entropy(D) = -\sum_{j=1}^{|C|} \Pr(c_j) \log_2 \Pr(c_j)$$

$$\sum_{j=1}^{|C|} \Pr(c_j) = 1,$$

- $\Pr(c_j)$ is the probability of class c_j in data set D
- We use entropy as a measure of impurity or disorder of data set D. (Or, a measure of information in a tree)

Entropy measure: let us get a feeling

1. The data set D has 50% positive examples ($\Pr(positive) = 0.5$) and 50% negative examples ($\Pr(negative) = 0.5$).

$$entropy(D) = -0.5 \times \log_2 0.5 - 0.5 \times \log_2 0.5 = 1$$

2. The data set D has 20% positive examples ($\Pr(positive) = 0.2$) and 80% negative examples ($\Pr(negative) = 0.8$).

$$entropy(D) = -0.2 \times \log_2 0.2 - 0.8 \times \log_2 0.8 = 0.722$$

3. The data set D has 100% positive examples ($\Pr(positive) = 1$) and no negative examples, ($\Pr(negative) = 0$).

$$entropy(D) = -1 \times \log_2 1 - 0 \times \log_2 0 = 0$$

- As the data become purer and purer, the entropy value becomes smaller and smaller. This is useful to us!

Information gain

- Given a set of examples D, we first compute its entropy:

$$entropy(D) = -\sum_{j=1}^{|C|} \Pr(c_j) \log_2 \Pr(c_j)$$

- If we make attribute A_i, with v values, the root of the current tree, this will partition D into v subsets D_1, D_2 …, D_v . The expected entropy if A_i is used as the current root:

$$entropy_{A_i}(D) = \sum_{j=1}^{v} \frac{|D_j|}{|D|} \times entropy(D_j)$$

Information gain (cont …)

- Information gained by selecting attribute A_i to branch or to partition the data is

$$gain(D, A_i) = entropy(D) - entropy_{A_i}(D)$$

- We choose the attribute with the highest gain to branch/split the current tree.

An example

$$entropy(D) = -\frac{6}{15}\times\log_2\frac{6}{15} - \frac{9}{15}\times\log_2\frac{9}{15} = 0.971$$

$$entropy_{Own\_house}(D) = \frac{6}{15}\times entropy(D_1) + \frac{9}{15}\times entropy(D_2)$$
$$= \frac{6}{15}\times 0 + \frac{9}{15}\times 0.918$$
$$= 0.551$$

$$entropy_{Age}(D) = \frac{5}{15}\times entropy(D_1) + \frac{5}{15}\times entropy(D_2) + \frac{5}{15}\times entropy(D_3)$$
$$= \frac{5}{15}\times 0.971 + \frac{5}{15}\times 0.971 + \frac{5}{15}\times 0.722$$
$$= 0.888$$

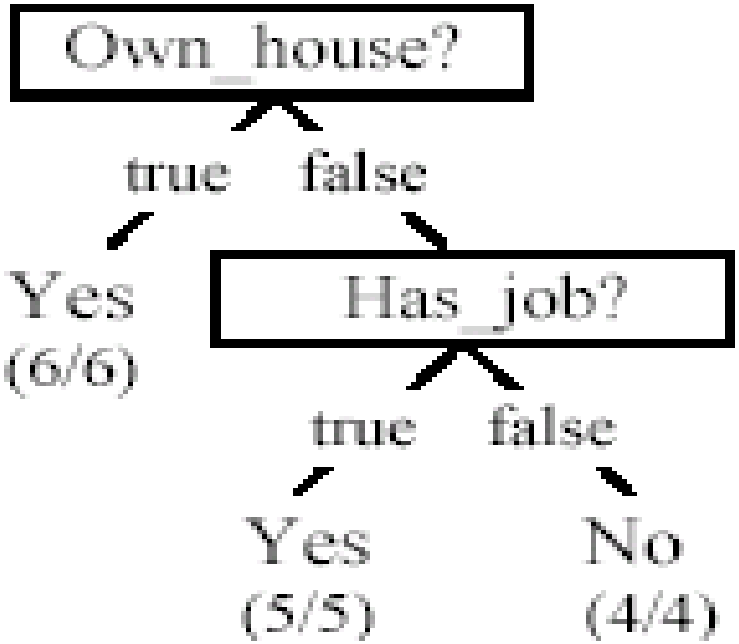
ID	Age	Has_Job	Own_House	Credit_Rating	Class
1	young	false	false	fair	No
2	young	false	false	excellent	No
3	young	true	false	good	Yes
4	young	true	true	good	Yes
5	young	false	false	fair	No
6	middle	false	false	fair	No
7	middle	false	false	good	No
8	middle	true	true	good	Yes
9	middle	false	true	excellent	Yes
10	middle	false	true	excellent	Yes
11	old	false	true	excellent	Yes
12	old	false	true	good	Yes
13	old	true	false	good	Yes
14	old	true	false	excellent	Yes
15	old	false	false	fair	No

Age	Yes	No	entropy(Di)
young	2	3	0.971
middle	3	2	0.971
old	4	1	0.722

- Own_house is the best choice for the root.

$$gain(D, \text{Age}) = 0.971 - 0.888 = 0.083$$
$$gain(D, \text{Own\_house}) = 0.971 - 0.551 = 0.420$$
$$gain(D, \text{Has\_Job}) = 0.971 - 0.647 = 0.324$$
$$gain(D, \text{Credit\_Rating}) = 0.971 - 0.608 = 0.363$$

We build the final tree

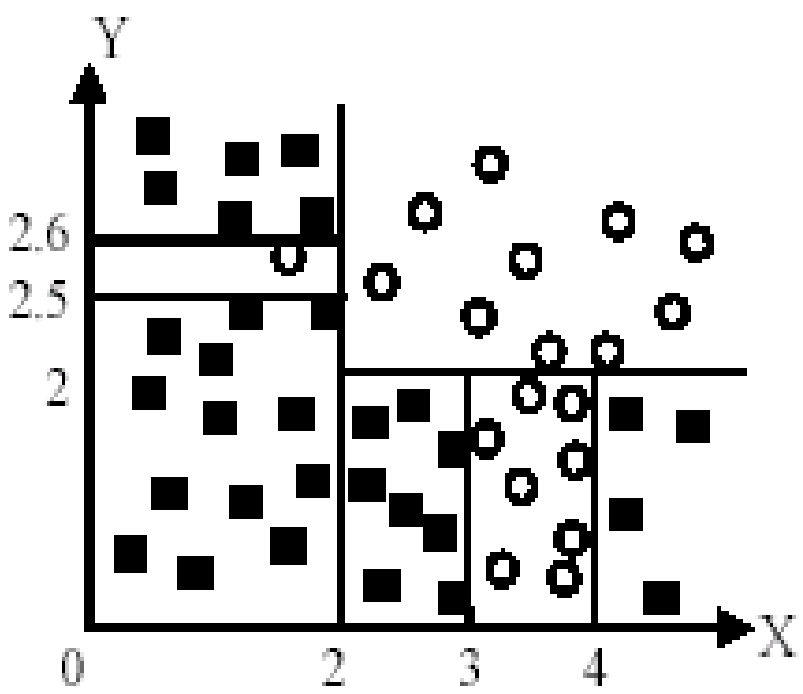


- We can use information gain ratio to evaluate the impurity as well (see the handout)

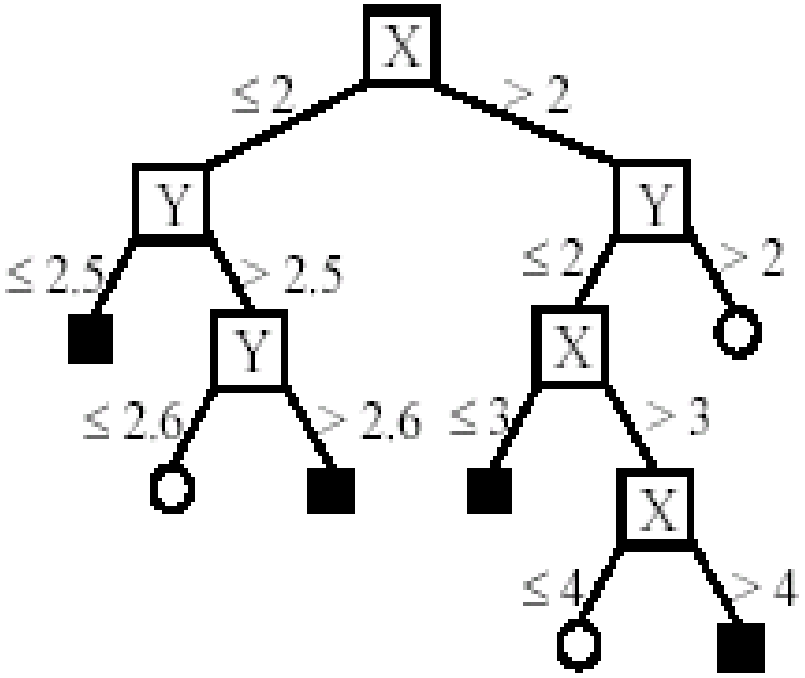
Handling continuous attributes

- Handle continuous attribute by splitting into two intervals (can be more) at each node.
- How to find the best threshold to divide?
  - Use information gain or gain ratio again
  - Sort all the values of an continuous attribute in increasing order $\{v_1, v_2, \ldots, v_r\}$,
  - One possible threshold between two adjacent values v_i and v_{i+1}. Try all possible thresholds and find the one that maximizes the gain (or gain ratio).

An example in a continuous space



(A) A partition of the data space

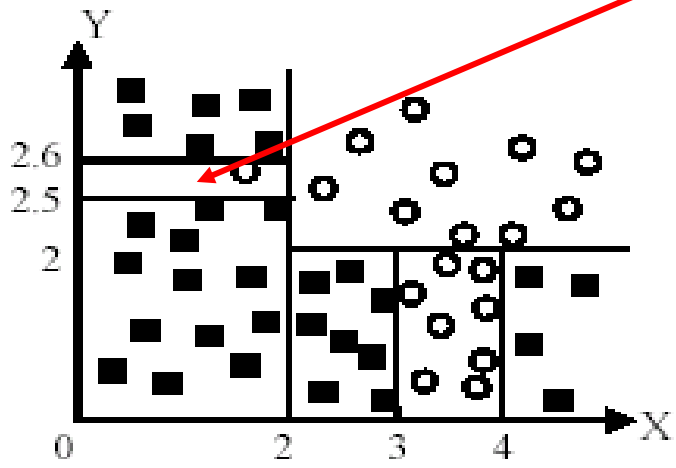


(B). The decision tree

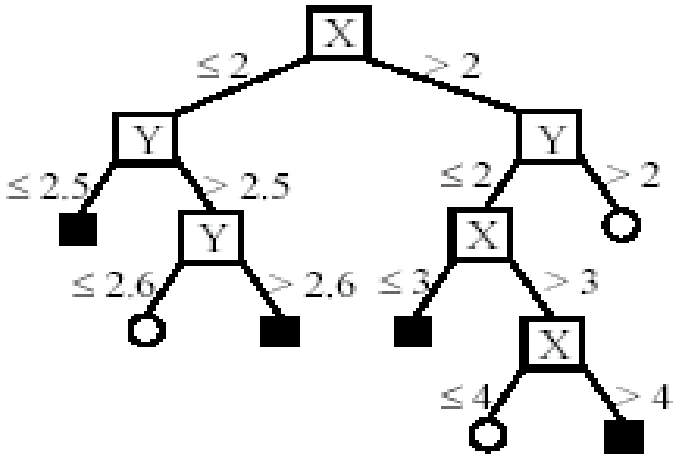
Avoid overfitting in classification

- Overfitting: A tree may overfit the training data
  - Good accuracy on training data but poor on test data
  - Symptoms: tree too deep and too many branches, some may reflect anomalies due to noise or outliers
- Two approaches to avoid overfitting
  - Pre-pruning: Halt tree construction early
    - Difficult to decide because we do not know what may happen subsequently if we keep growing the tree.
  - Post-pruning: Remove branches or sub-trees from a "fully grown" tree.
    - This method is commonly used. C4.5 uses a statistical method to estimates the errors at each node for pruning.
    - A validation set may be used for pruning as well.

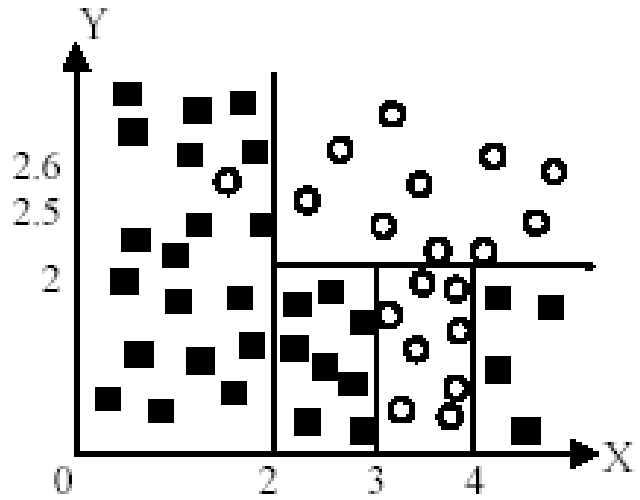
An example Likely to overfit the data

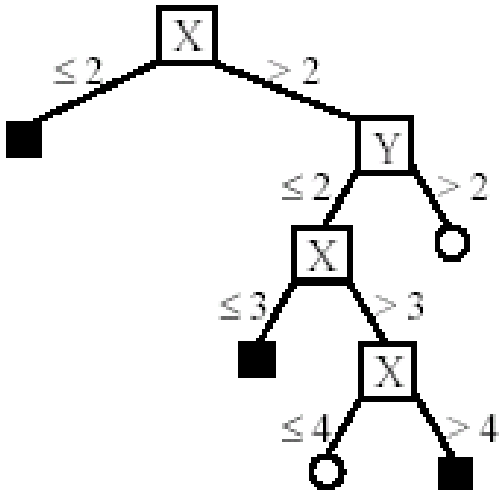


(A) A partition of the data space



(B). The decision tree





Other issues in decision tree learning

- From tree to rules, and rule pruning
- Handling of miss values
- Handing skewed distributions
- Handling attributes and classes with different costs.
- Attribute construction
- Etc.

Road Map

- Basic concepts
- Decision tree induction
- **Evaluation of classifiers**
- Rule induction
- Classification using association rules
- Naïve Bayesian classification
- Naïve Bayes for text classification
- Support vector machines
- K-nearest neighbor
- Ensemble methods: Bagging and Boosting
- Summary

Evaluating classification methods

- **Predictive accuracy**

$$Accuracy = \frac{\text{Number of correct classifications}}{\text{Total number of test cases}}$$

- Efficiency
  - time to construct the model
  - time to use the model
- Robustness: handling noise and missing values
- Scalability: efficiency in disk-resident databases
- Interpretability:
  - understandable and insight provided by the model
- Compactness of the model: size of the tree, or the number of rules.

Evaluation methods

- **Holdout set**: The available data set *D* is divided into two disjoint subsets,
  - the *training set* D_{train} (for learning a model)
  - the *test set* D_{test} (for testing the model)
- **Important:** training set should not be used in testing and the test set should not be used in learning.
  - Unseen test set provides a unbiased estimate of accuracy.
- The test set is also called the holdout set. (the examples in the original data set *D* are all labeled with classes.)
- This method is mainly used when the data set *D* is large.

Evaluation methods (cont...)

- **n-fold cross-validation**: The available data is partitioned into *n* equal-size disjoint subsets.
- Use each subset as the test set and combine the rest *n*-1 subsets as the training set to learn a classifier.
- The procedure is run *n* times, which give *n* accuracies.
- The final estimated accuracy of learning is the average of the *n* accuracies.
- 10-fold and 5-fold cross-validations are commonly used.
- This method is used when the available data is not large.

Evaluation methods (cont…)

- **Leave-one-out cross-validation**: This method is used when the data set is very small.
- It is a special case of cross-validation
- Each fold of the cross validation has only a single test example and all the rest of the data is used in training.
- If the original data has *m* examples, this is *m*-fold cross-validation

Evaluation methods (cont…)

- **Validation set**: the available data is divided into three subsets,
  - a training set,
  - a validation set and
  - a test set.
- A validation set is used frequently for estimating parameters in learning algorithms.
- In such cases, the values that give the best accuracy on the validation set are used as the final parameter values.
- Cross-validation can be used for parameter estimating as well.

Classification measures

- Accuracy is only one measure (error = 1-accuracy).
- **Accuracy is not suitable in some applications**.
- In text mining, we may only be interested in the documents of a particular topic, which are only a small portion of a big document collection.
- In classification involving skewed or highly imbalanced data, e.g., network intrusion and financial fraud detections, we are interested only in the minority class.
  - High accuracy does not mean any intrusion is detected.
  - E.g., 1% intrusion. Achieve 99% accuracy by doing nothing.
- The class of interest is commonly called the **positive class**, and the rest **negative classes**.

Precision and recall measures

- Used in information retrieval and text classification.
- We use a confusion matrix to introduce them.

	Classified Positive	Classified Negative
Actual Positive	TP	FN
Actual Negative	FP	TN

where

TP: the number of correct classifications of the positive examples (**true positive**),

FN: the number of incorrect classifications of positive examples (**false negative**),

FP: the number of incorrect classifications of negative examples (**false positive**), and

TN: the number of correct classifications of negative examples (**true negative**).

Precision and recall

	Classified Positive	Classified Negative
Actual Positive	TP	FN
Actual Negative	FP	TN

$$p = \frac{TP}{TP + FP}. \qquad r = \frac{TP}{TP + FN}.$$

- Precision *p* is the number of correctly classified positive examples divided by the total number of examples that are classified as positive.
- Recall *r* is the number of correctly classified positive examples divided by the total number of actual positive examples in the test set.

An example

	Classified Positive	Classified Negative
Actual Positive	1	99
Actual Negative	0	1000

- This confusion matrix gives
  - precision p = 100% and
  - recall r = 1%

  because we only classified one positive example correctly and no negative examples wrongly.
- Note: precision and recall only measure classification on the positive class.

F_1-value (also called F_1-score)

- It is hard to compare two classifiers using two measures. F_1 score combines precision and recall into one measure

$$F_1 = \frac{2pr}{p + r}$$

F_1-score is the harmonic mean of precision and recall.

$$F_1 = \frac{2}{\frac{1}{p} + \frac{1}{r}}$$

- The harmonic mean of two numbers tends to be closer to the smaller of the two.
- For F_1-value to be large, both *p* and *r* much be large.

Receive operating characteristics curve

- It is commonly called the ROC curve.
- It is a plot of the true positive rate (TPR) against the false positive rate (FPR).
- True positive rate:

$$TPR = \frac{TP}{TP + FN}$$

- False positive rate:

$$FPR = \frac{FP}{TN + FP}$$

Sensitivity and Specificity

- In statistics, there are two other evaluation measures:
  - Sensitivity: Same as TPR
  - Specificity: Also called True Negative Rate (TNR)

$$TNR = \frac{TN}{TN + FP}$$

- Then we have

$$FPR = 1 - specificity$$

Example ROC curves

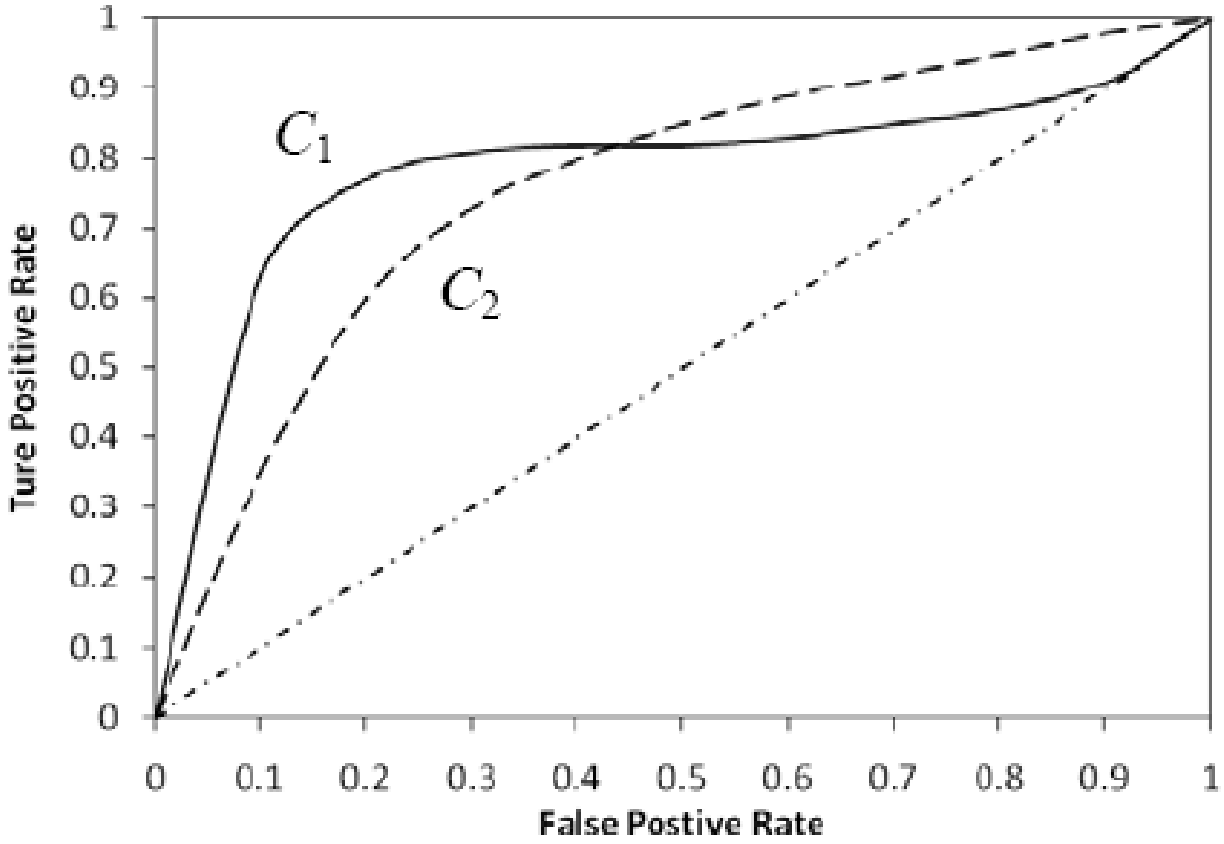


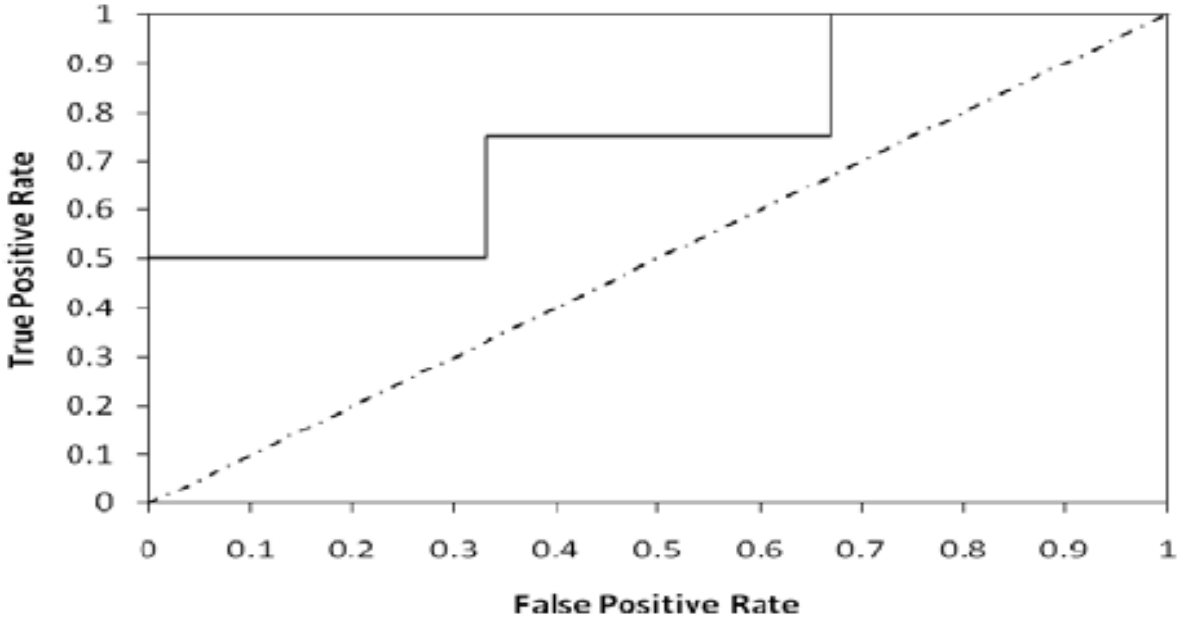
Fig. 3.8. ROC curves for two classifiers (C_1 and C_2) on the same data

Area under the curve (AUC)

- Which classifier is better, C_1 or C_2?
  - It depends on which region you talk about.
- Can we have one measure?
  - Yes, we compute the area under the curve (AUC)
- If AUC for C_i is greater than that of C_j, it is said that C_i is better than C_j.
  - If a classifier is perfect, its AUC value is 1
  - If a classifier makes all random guesses, its AUC value is 0.5.

Drawing an ROC curve

Rank		1	2	3	4	5	6	7	8	9	10
Actual class		+	+	–	–	+	–	–	+	–	–
TP	0	1	2	2	2	3	3	3	4	4	4
FP	0	0	0	1	2	2	3	4	4	5	6
TN	6	6	6	5	4	4	3	2	2	1	0
FN	4	3	2	2	2	1	1	1	0	0	0
TPR	0	0.25	0.5	0.5	0.5	0.75	0.75	0.75	1	1	1
FPR	0	0	0	0.17	0.33	0.33	0.50	0.67	0.67	0.83	1



Another evaluation method: Scoring and ranking

- Scoring is related to classification.
- We are interested in a single class (positive class), e.g., buyers class in a marketing database.
- Instead of assigning each test instance a definite class, scoring assigns a probability estimate (PE) to indicate the likelihood that the example belongs to the positive class.

Ranking and lift analysis

- After each example is given a PE score, we can rank all examples according to their PEs.
- We then divide the data into n (say 10) bins. A lift curve can be drawn according how many positive examples are in each bin. This is called lift analysis.
- Classification systems can be used for scoring. Need to produce a probability estimate.
  - E.g., in decision trees, we can use the confidence value at each leaf node as the score.

An example

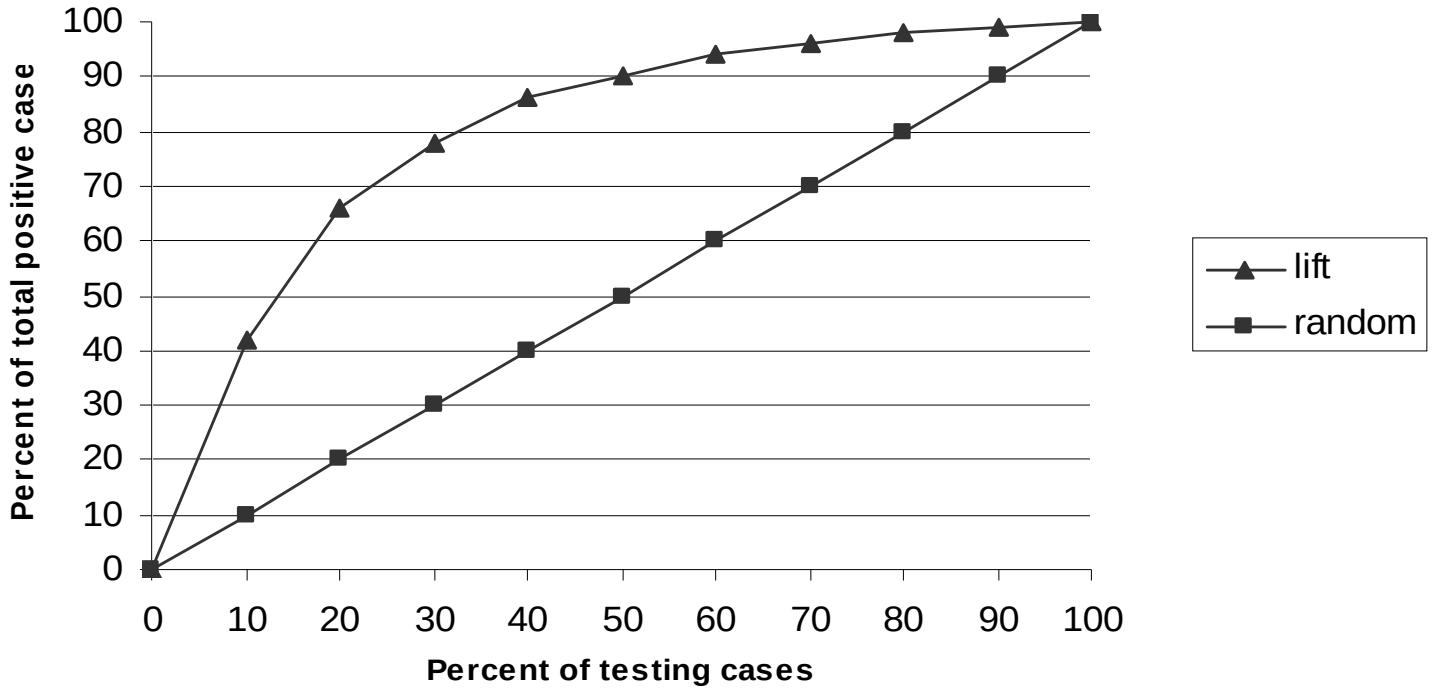
- We want to send promotion materials to potential customers to sell a watch.
- Each package cost $0.50 to send (material and postage).
- If a watch is sold, we make $5 profit.
- Suppose we have a large amount of past data for building a predictive/classification model. We also have a large list of potential customers.
- How many packages should we send and who should we send to?

An example

- Assume that the test set has 10000 instances. Out of this, 500 are positive cases.
- After the classifier is built, we score each test instance. We then rank the test set, and divide the ranked test set into 10 bins.
  - Each bin has 1000 test instances.
  - Bin 1 has 210 actual positive instances
  - Bin 2 has 120 actual positive instances
  - Bin 3 has 60 actual positive instances
  - …
  - Bin 10 has 5 actual positive instances

Lift curve

Bin	1	2	3	4	5	6	7	8	9	10
	210	120	60	40	22	18	12	7	6	5
	42%	24%	12%	8%	4.40%	3.60%	2.40%	1.40%	1.20%	1%
	42%	66%	78%	86%	90.40%	94%	96.40%	97.80%	99%	100%



Road Map

- Basic concepts
- Decision tree induction
- Evaluation of classifiers
- **Rule induction**
- Classification using association rules
- Naïve Bayesian classification
- Naïve Bayes for text classification
- Support vector machines
- K-nearest neighbor
- Summary

Introduction

- We showed that a decision tree can be converted to a set of rules.
- Can we find if-then rules directly from data for classification?
- Yes.
- Rule induction systems find a sequence of rules (also called a decision list) for classification.
- The commonly used strategy is sequential covering.

Sequential covering

- Learn one rule at a time, sequentially.
- After a rule is learned, the training examples covered by the rule are removed.
- Only the remaining data are used to find subsequent rules.
- The process repeats until some stopping criteria are met.

Note: a rule covers an example if the example satisfies the conditions of the rule.

- We introduce two specific algorithms.

Algorithm 1: ordered rules

Algorithm sequential-covering-1(D)

```
1   RuleList ← ∅;
2   Rule ← learn-one-rule-1(D);
3   while Rule is not NULL AND D ≠ ∅ do
4       RuleList ← insert Rule at the end of RuleList;
5       Remove from D the examples covered by Rule;
6       Rule ← learn-one-rule-1(D)
7   endwhile
8   insert a default class c at the end of RuleList, where c is the majority class
        in D;
9   return RuleList
```

- The final classifier:

  $<r_1, r_2, \ldots, r_k,$ *default-class*$>$

Algorithm 2: ordered classes

Algorithm sequential-covering-2(D, C)

```
1   RuleList ← ∅;                                  // empty rule set at the beginning
2   for each class c ∈ C do
3       prepare data (Pos, Neg), where Pos contains all the examples of class c
            from D, and Neg contains the rest of the examples in D;
4       while Pos ≠ ∅ do
5           Rule ← learn-one-rule-2(Pos, Neg, c);
6           if Rule is NULL then
7              exit-while-loop
8           else RuleList ← insert Rule at the end of RuleList;
9                Remove examples covered by Rule from (Pos, Neg)
10          endif
11      endwhile
12  endfor
13  return RuleList
```

- Rules of the same class are together.

Algorithm 1 vs. Algorithm 2

- Differences:
  - Algorithm 2: Rules of the same class are found together. The classes are ordered. Normally, minority class rules are found first.
  - Algorithm 1: In each iteration, a rule of any class may be found. Rules are ordered according to the sequence they are found.
- Use of rules: the same.
  - For a test instance, we try each rule sequentially. The first rule that covers the instance classifies it.
  - If no rule covers it, default class is used, which is the majority class in the data.

Learn-one-rule-1 function

- Let us consider only categorical attributes
- Let attributeValuePairs contains all possible attribute-value pairs ($A_i = a_i$) in the data.
- Iteration 1: Each attribute-value is evaluated as the condition of a rule. I.e., we compare all such rules $A_i = a_i \rightarrow c_j$ and keep the best one,
  - Evaluation: e.g., entropy
  - Also store the k best rules for beam search (to search more space). Called new candidates.

Learn-one-rule-1 function (cont. .)

- In iteration m, each (m-1)-condition rule in the new candidates set is expanded by attaching each attribute-value pair in attributeValuePairs as an additional condition to form candidate rules.
- These new candidate rules are then evaluated in the same way as 1-condition rules.
  - Update the best rule
  - Update the k-best rules
- The process repeats unless stopping criteria are met.

Learn-one-rule-1 algorithm

```
Function learn-one-rule-1(D),
1    BestCond ← ∅;                          // rule with no condition.
2    candidateCondSet ← {bestCond};
3    attributeValuePairs ← the set of all attribute-value pairs in D of the form
       (A_i op v), where A_i is an attribute and v is a value or an interval;
4    while candidateCondSet ≠ ∅ do
5        newCandidateCondSet ← ∅;
6        for each candidate cond in candidateCondSet do
7            for each attribute-value pair a in attributeValuePairs do
8                newCond ← cond ∪ {a};
9                newCandidateCondSet ← newCandidateCondSet ∪ {newCond}
10           endfor
11       endfor
12       remove duplicates and inconsistencies, e.g., {A_i = v_1, A_i = v_2};
13       for each candidate newCond in newCandidateCondSet do
14           if evaluation(newCond, D) > evaluation(BestCond, D) then
15               BestCond ← newCond;
16           endif
17       endfor
18       candidateCondSet ← the k best members of newCandidateCondSet
             according to the results of the evaluation function;
19   endwhile
20   if evaluation(BestCond, D) − evaluation(∅, D) > threshold then
21       return the rule: "BestCond → c" where is c the majority class of the
         data covered by BestCond;
22   else return NULL
23   endif
```

Learn-one-rule-2 function

- Split the data:
  - Pos -> GrowPos and PrunePos
  - Neg -> GrowNeg and PruneNeg
- Grow sets are used to find a rule (*BestRule*), and the Prune sets are used to prune the rule.
- GrowRule works similarly as in learn-one-rule-1, but the class is fixed in this case. Recall the second algorithm finds all rules of a class first (Pos) and then moves to the next class.

Learn-one-rule-2 algorithm

```
Function learn-one-rule-2(Pos, Neg, class)
1   split (Pos, Neg) into (GrowPos, GrowNeg) and (PrunePos, PruneNeg)
2   BestRule ← GrowRule(GrowPos, GrowNeg, class)        // grow a new rule
3   BestRule ← PruneRule(BestRule, PrunePos, PruneNeg)  // prune the rule
4   if the error rate of BestRule on (PrunePos, PruneNeg) exceeds 50% then
5      return NULL
6   endif
7   return BestRule
```

Rule evaluation in learn-one-rule-2

- Let the current partially developed rule be:

  R: $av_1, .., av_k \rightarrow class$

  - where each av_j is a condition (an attribute-value pair).
- By adding a new condition av_{k+1}, we obtain the rule

  $R+$: $av_1, .., av_k, av_{k+1} \rightarrow class$.
- The evaluation function for $R+$ is the following information **gain** criterion (which is different from the gain function used in decision tree learning).

$$gain(R, R^+) = p_1 \times \left[\log_2 \frac{p_1}{p_1 + n_1} - \log_2 \frac{p_0}{p_0 + n_0} \right]$$

- Rule with the best gain is kept for further extension.

Rule pruning in learn-one-rule-2

- Consider deleting every subset of conditions from the *BestRule*, and choose the deletion that maximizes the function:

$$v(BestRule, PrunePos, PruneNeg) = \frac{p - n}{p + n}$$

where p (n) is the number of examples in *PrunePos* (*PruneNeg*) covered by the current rule (after a deletion).

Discussions

- Accuracy: similar to decision tree
- Efficiency: Run much slower than decision tree induction because
  - To generate each rule, all possible rules are tried on the data (not really all, but still a lot).
  - When the data is large and/or the number of attribute-value pairs are large. It may run very slowly.
- Rule interpretability: Can be a problem because each rule is found after data covered by previous rules are removed. Thus, each rule may not be treated as independent of other rules.

Road Map

- Basic concepts
- Decision tree induction
- Evaluation of classifiers
- Rule induction
- **Classification using association rules**
- Naïve Bayesian classification
- Naïve Bayes for text classification
- Support vector machines
- K-nearest neighbor
- Ensemble methods: Bagging and Boosting
- Summary

Three approaches

- Three main approaches of using association rules for classification.
  - Using class association rules to build classifiers
  - Using class association rules as attributes/features
  - Using normal association rules for classification

Using Class Association Rules

- **Classification:** mine a small set of rules existing in the data to form a classifier or predictor.
  - It has a target attribute: Class attribute
- **Association rules:** have no fixed target, but we can fix a target.
- Class association rules (CAR): has a target class attribute. E.g.,

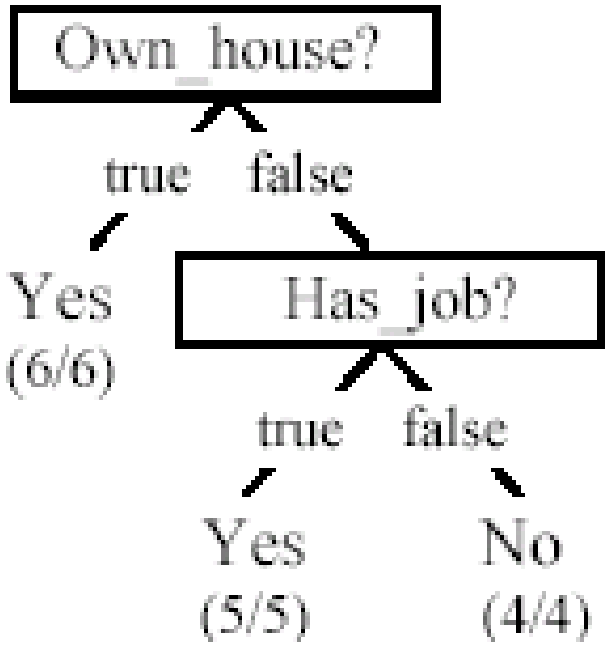
  Own_house = true → Class =Yes [sup=6/15, conf=6/6]
  - CARs can obviously be used for classification.

Decision tree vs. CARs

- **The decision tree below generates the following 3 rules**.

Own_house = true → Class =Yes [sup=6/15, conf=6/6]
Own_house = false, Has_job = true → Class=Yes [sup=5/15, conf=5/5]
Own_house = false, Has_job = false → Class=No [sup=4/15, conf=4/4]

- But there are many other rules that are not found by the decision tree



There are many more rules

Age = young, Has_job = true → Class=Yes [sup=2/15, conf=2/2]
Age = young, Has_job = false → Class=No [sup=3/15, conf=3/3]
Credit_Rating = fair → Class=No [sup=4/15, conf=4/4]
Credit_Rating = good → Class=Yes [sup=5/15, conf=5/6]

and many more, if we use minsup = 2/15 = 13.3% and minconf = 80%.

- CAR mining finds all of them.
- In many cases, rules not in the decision tree (or a rule list) may perform classification better.
- Such rules may also be actionable in practice

ID	Age	Has_Job	Own_House	Credit_Rating	Class
1	young	false	false	fair	No
2	young	false	false	excellent	No
3	young	true	false	good	Yes
4	young	true	true	good	Yes
5	young	false	false	fair	No
6	middle	false	false	fair	No
7	middle	false	false	good	No
8	middle	true	true	good	Yes
9	middle	false	true	excellent	Yes
10	middle	false	true	excellent	Yes
11	old	false	true	excellent	Yes
12	old	false	true	good	Yes
13	old	true	false	good	Yes
14	old	true	false	excellent	Yes
15	old	false	false	fair	No

Decision tree vs. CARs (cont …)

- Association mining require discrete attributes. Decision tree learning uses both discrete and continuous attributes.
  - CAR mining requires continuous attributes discretized. There are several such algorithms.
- Decision tree is not constrained by minsup or minconf, and thus is able to find rules with very low support. Of course, such rules may be pruned due to the possible overfitting.

Considerations in CAR mining

- Multiple minimum class supports
  - Deal with imbalanced class distribution, e.g., some class is rare, 98% negative and 2% positive.
  - We can set the minsup(positive) = 0.2% and minsup(negative) = 2%.
  - If we are not interested in classification of negative class, we may not want to generate rules for negative class. We can set minsup(negative)=100% or more.
- Rule pruning may be performed.

Building classifiers

- There are many ways to build classifiers using CARs. Several existing systems available.
- Strongest rules: After CARs are mined, do nothing.
  - For each test case, we simply choose the most confident rule that covers the test case to classify it. Microsoft SQL Server has a similar method.
  - Or, using a combination of rules.
- Selecting a subset of Rules
  - used in the CBA system.
  - similar to sequential covering.

CBA: Rules are sorted first

Definition: Given two rules, r_i and r_j, $r_i \succ r_j$ (also called *ri* precedes r_j or r_i has a higher precedence than r_j) if

- the confidence of r_i is greater than that of r_j, or
- their confidences are the same, but the support of r_i is greater than that of r_j, or
- both the confidences and supports of r_i and r_j are the same, but r_i is generated earlier than r_j.

A CBA classifier L is of the form:

$L = <r_1, r_2, \ldots, r_k, \textit{default-class}>$

Classifier building using CARs

Algorithm CBA(S, D)

```
1  S = sort(S);                    // sorting is done according to the precedence ≻
2  RuleList = ∅;                   // the rule list classifier
3  for each rule r ∈ S in sequence do
4      if D ≠ ∅ AND r classifies at least one example in D correctly then
5          delete from D all training examples covered by r;
6          add r at the end of RuleList
7      end
8  end
9  add the majority class as the default class at the end of RuleList
```

- This algorithm is very inefficient
- CBA has a very efficient algorithm (quite sophisticated) that scans the data at most two times.

Using rules as features

- Most classification methods do not fully explore multi-attribute correlations, e.g., naïve Bayesian, decision trees, rules induction, etc.
- This method creates extra attributes to augment the original data by
  - Using the conditional parts of rules
  - Each rule forms an new attribute
  - If a data record satisfies the condition of a rule, the attribute value is 1, and 0 otherwise
- One can also use only rules as attributes
  - Throw away the original data

Using normal association rules for classification

- A widely used approach
- Main approach: strongest rules
- Main application
  - Recommendation systems in e-commerce Web site (e.g., amazon.com).
  - Each rule consequent is the recommended item.
- Major advantage: any item can be predicted.
- Main issue:
  - Coverage: rare item rules are not found using classic algo.
  - Multiple min supports and support difference constraint help a great deal.

Road Map

- Basic concepts
- Decision tree induction
- Evaluation of classifiers
- Rule induction
- Classification using association rules
- **Naïve Bayesian classification**
- Naïve Bayes for text classification
- Support vector machines
- K-nearest neighbor
- Ensemble methods: Bagging and Boosting
- Summary

Bayesian classification

- Probabilistic view: Supervised learning can naturally be studied from a probabilistic point of view.
- Let A_1 through A_k be attributes with discrete values. The class is C.
- Given a test example *d* with observed attribute values a_1 through a_k.
- Classification is basically to compute the following posteriori probability. The prediction is the class c_j such that

$$\Pr(C = c_j \mid A_1 = a_1, \ldots, A_{|A|} = a_{|A|})$$

is maximal

Apply Bayes' Rule

$$\Pr(C = c_j \mid A_1 = a_1, \ldots, A_{|A|} = a_{|A|})$$

$$= \frac{\Pr(A_1 = a_1, \ldots, A_{|A|} = a_{|A|} \mid C = c_j)\Pr(C = c_j)}{\Pr(A_1 = a_1, \ldots, A_{|A|} = a_{|A|})}$$

$$= \frac{\Pr(A_1 = a_1, \ldots, A_{|A|} = a_{|A|} \mid C = c_j)\Pr(C = c_j)}{\sum_{r=1}^{|C|} \Pr(A_1 = a_1, \ldots, A_{|A|} = a_{|A|} \mid C = c_r)\Pr(C = c_r)}$$

- $\Pr(C=c_j)$ is the class *prior* probability: easy to estimate from the training data.

Computing probabilities

- The denominator $P(A_1=a_1,...,A_k=a_k)$ is irrelevant for decision making since it is the same for every class.
- We only need $P(A_1=a_1,...,A_k=a_k \mid C=c_i)$, which can be written as

  $Pr(A_1=a_1|A_2=a_2,...,A_k=a_k, C=c_j)* Pr(A_2=a_2,...,A_k=a_k \mid C=c_j)$
- Recursively, the second factor above can be written in the same way, and so on.
- Now an assumption is needed.

Conditional independence assumption

- All attributes are conditionally independent given the class $C = c_j$.
- Formally, we assume,

  $\Pr(A_1=a_1 \mid A_2=a_2, ..., A_{|A|}=a_{|A|}, C=c_j) = \Pr(A_1=a_1 \mid C=c_j)$

  and so on for A_2 through $A_{|A|}$. I.e.,

$$\Pr(A_1 = a_1, ..., A_{|A|} = a_{|A|} \mid C = c_i) = \prod_{i=1}^{|A|} \Pr(A_i = a_i \mid C = c_j)$$

Final naïve Bayesian classifier

$$\Pr(C = c_j \mid A_1 = a_1, \ldots, A_{|A|} = a_{|A|})$$

$$= \frac{\Pr(C = c_j) \prod_{i=1}^{|A|} \Pr(A_i = a_i \mid C = c_j)}{\sum_{r=1}^{|C|} \Pr(C = c_r) \prod_{i=1}^{|A|} \Pr(A_i = a_i \mid C = c_r)}$$

- We are done!
- How do we estimate $P(A_i = a_i \mid C = c_j)$? Easy!.

Classify a test instance

- If we only need a decision on the most probable class for the test instance, we only need the numerator as its denominator is the same for every class.
- Thus, given a test example, we compute the following to decide the most probable class for the test instance

$$c = \arg\max_{c_j} \Pr(c_j) \prod_{i=1}^{|A|} \Pr(A_i = a_i \mid C = c_j)$$

An example

- Compute all probabilities required for classification

A	B	C
m	b	t
m	s	t
g	q	t
h	s	t
g	q	t
g	q	f
g	s	f
h	b	f
h	q	f
m	b	f

$Pr(C = t) = 1/2$, $Pr(C = f) = 1/2$

$Pr(A=m \mid C=t) = 2/5$
$Pr(A=m \mid C=f) = 1/5$
$Pr(B=b \mid C=t) = 1/5$
$Pr(B=b \mid C=f) = 2/5$

$Pr(A=g \mid C=t) = 2/5$
$Pr(A=g \mid C=f) = 2/5$
$Pr(B=s \mid C=t) = 2/5$
$Pr(B=s \mid C=f) = 1/5$

$Pr(A=h \mid C=t) = 1/5$
$Pr(A=h \mid C=n) = 2/5$
$Pr(B=q \mid C=t) = 2/5$
$Pr(B=q \mid C=f) = 2/5$

Now we have a test example:

$A = m \quad B = q \quad C = ?$

An Example (cont …)

- For C = t, we have

$$\Pr(C=t)\prod_{j=1}^{2}\Pr(A_j=a_j\,|\,C=t)=\frac{1}{2}\times\frac{2}{5}\times\frac{2}{5}=\frac{2}{25}$$

- For class C = f, we have

$$\Pr(C=f)\prod_{j=1}^{2}\Pr(A_j=a_j\,|\,C=f)=\frac{1}{2}\times\frac{1}{5}\times\frac{2}{5}=\frac{1}{25}$$

- C = t is more probable. t is the final class.

Additional issues

- **Numeric attributes**: Naïve Bayesian learning assumes that all attributes are categorical. Numeric attributes need to be discretized.
- **Zero counts**: An particular attribute value never occurs together with a class in the training set. We need smoothing.

$$\Pr(A_i = a_i \mid C = c_j) = \frac{n_{ij} + \lambda}{n_j + \lambda n_i}$$

- **Missing values**: Ignored

On naïve Bayesian classifier

- Advantages:
  - Easy to implement
  - Very efficient
  - Good results obtained in many applications
- Disadvantages
  - Assumption: class conditional independence, therefore loss of accuracy when the assumption is seriously violated (those highly correlated data sets)

Road Map

- Basic concepts
- Decision tree induction
- Evaluation of classifiers
- Rule induction
- Classification using association rules
- Naïve Bayesian classification
- **Naïve Bayes for text classification**
- Support vector machines
- K-nearest neighbor
- Ensemble methods: Bagging and Boosting
- Summary

Text classification/categorization

- Due to the rapid growth of online documents in organizations and on the Web, automated document classification has become an important problem.
- Techniques discussed previously can be applied to text classification, but they are not as effective as the next three methods.
- We first study a naïve Bayesian method specifically formulated for texts, which makes use of some text specific features.
- However, the ideas are similar to the preceding method.

Probabilistic framework

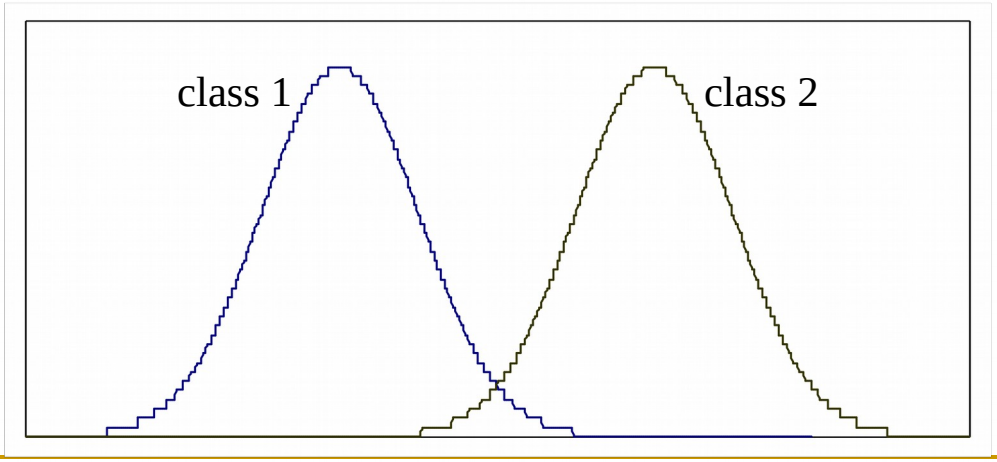
- Generative model: Each document is generated by a parametric distribution governed by a set of hidden parameters.
- The generative model makes two assumptions
  - The data (or the text documents) are generated by a mixture model,
  - There is one-to-one correspondence between mixture components and document classes.

Mixture model

- A **mixture model** models the data with a number of statistical distributions.
  - Intuitively, each distribution corresponds to a data cluster and the parameters of the distribution provide a description of the corresponding cluster.
- Each distribution in a mixture model is also called a **mixture component**.
- The distribution/component can be of any kind

An example

- The figure shows a plot of the **probability density function** of a 1-dimensional data set (with two classes) generated by
  - a mixture of two Gaussian distributions,
  - one per class, whose parameters (denoted by θ_i) are the mean (μ_i) and the standard deviation (σ_i), i.e., $\theta_i = (\mu_i, \sigma_i)$.



Mixture model (cont …)

- Let the number of mixture components (or distributions) in a mixture model be K.
- Let the jth distribution have the parameters θ_j.
- Let Θ be the set of parameters of all components, $\Theta = \{\varphi_1, \varphi_2, \ldots, \varphi_K, \theta_1, \theta_2, \ldots, \theta_K\}$, where φ_j is the *mixture weight* (or *mixture probability*) of the mixture component j and θ_j is the parameters of component j.
- How does the model generate documents?

Document generation

- Due to one-to-one correspondence, each class corresponds to a mixture component. The mixture weights are *class prior probabilities*, i.e., $\varphi_j = \Pr(c_j|\Theta)$.
- The mixture model generates each document d_i by:
  - first selecting a mixture component (or class) according to class prior probabilities (i.e., mixture weights), $\varphi_j = \Pr(c_j|\Theta)$.
  - then having this selected mixture component (c_j) generate a document d_i according to its parameters, with distribution $\Pr(d_i|c_j; \Theta)$ or more precisely $\Pr(d_i|c_j; \theta_j)$.

$$\Pr(d_i \mid \Theta) = \sum_{j=1}^{|C|} \Pr(c_j \mid \Theta) \Pr(d_i \mid c_j; \Theta) \qquad (23)$$

Model text documents

- The naïve Bayesian classification treats each document as a "bag of words". The generative model makes the following further assumptions:
  - Words of a document are generated independently of context given the class label. The familiar naïve Bayes assumption used before.
  - The probability of a word is independent of its position in the document. The document length is chosen independent of its class.

Multinomial distribution

- With the assumptions, each document can be regarded as generated by a multinomial distribution.
- In other words, each document is drawn from a multinomial distribution of words with as many independent trials as the length of the document.
- The words are from a given vocabulary $V = \{w_1, w_2, \ldots, w_{|V|}\}$.

Use probability function of multinomial distribution

$$\Pr(d_i \mid c_j;\Theta) = \Pr(|d_i|)\,|d_i|!\prod_{t=1}^{|V|}\frac{\Pr(w_t \mid c_j;\Theta)^{N_{ti}}}{N_{ti}!} \quad (24)$$

where N_{ti} is the number of times that word w_t occurs in document d_i and

$$\sum_{t=1}^{|V|} N_{it} = |d_i| \qquad \sum_{t=1}^{|V|} \Pr(w_t \mid c_j;\Theta) = 1. \quad (25)$$

Parameter estimation

- The parameters are estimated based on empirical counts.

$$\Pr(w_t \mid c_j; \hat{\Theta}) = \frac{\sum_{i=1}^{|D|} N_{ti} \Pr(c_j \mid d_i)}{\sum_{s=1}^{|V|} \sum_{i=1}^{|D|} N_{si} \Pr(c_j \mid d_i)}. \quad (26)$$

- In order to handle 0 counts for infrequent occurring words that do not appear in the training set, but may appear in the test set, we need to smooth the probability. *Lidstone* smoothing, $0 \leq \lambda \leq 1$

$$\Pr(w_t \mid c_j; \hat{\Theta}) = \frac{\lambda + \sum_{i=1}^{|D|} N_{ti} \Pr(c_j \mid d_i)}{\lambda |V| + \sum_{s=1}^{|V|} \sum_{i=1}^{|D|} N_{si} \Pr(c_j \mid d_i)}. \quad (27)$$

Parameter estimation (cont …)

- Class prior probabilities, which are mixture weights φ_j, can be easily estimated using training data

$$\Pr(c_j \mid \hat{\Theta}) = \frac{\sum_{i=1}^{|D|} \Pr(c_j \mid d_i)}{|D|} \quad (28)$$

Classification

- Given a test document d_i, from Eq. (23) (27) and (28)

$$\Pr(c_j \mid d_i; \hat{\Theta}) = \frac{\Pr(c_j \mid \hat{\Theta}) \Pr(d_i \mid c_j; \hat{\Theta})}{\Pr(d_i \mid \hat{\Theta})}$$

$$= \frac{\Pr(c_j \mid \hat{\Theta}) \prod_{k=1}^{|d_i|} \Pr(w_{d_i,k} \mid c_j; \hat{\Theta})}{\sum_{r=1}^{|C|} \Pr(c_r \mid \hat{\Theta}) \prod_{k=1}^{|d_i|} \Pr(w_{d_i,k} \mid c_r; \hat{\Theta})}$$

where $w_{d_i,k}$ is the word in position k of document d_i. If the final classifier is to classify each document into a single class, then the class with the highest posterior probability is selected:

$$\arg\max_{c_j \in C} \Pr(c_j \mid d_i; \hat{\Theta}) \qquad (30)$$

Discussions

- Most assumptions made by naïve Bayesian learning are violated to some degree in practice.
- Despite such violations, researchers have shown that naïve Bayesian learning produces very accurate models.
  - The main problem is the mixture model assumption. When this assumption is seriously violated, the classification performance can be poor.
- Naïve Bayesian learning is extremely efficient.

Road Map

- Basic concepts
- Decision tree induction
- Evaluation of classifiers
- Rule induction
- Classification using association rules
- Naïve Bayesian classification
- Naïve Bayes for text classification
- **Support vector machines**
- K-nearest neighbor
- Ensemble methods: Bagging and Boosting
- Summary

Introduction

- Support vector machines were invented by V. Vapnik and his co-workers in 1970s in Russia and became known to the West in 1992.
- SVMs are linear classifiers that find a hyperplane to separate two class of data, positive and negative.
- Kernel functions are used for nonlinear separation.
- SVM not only has a rigorous theoretical foundation, but also performs classification more accurately than most other methods in applications, especially for high dimensional data.
- It is perhaps the best classifier for text classification.

Basic concepts

- Let the set of training examples D be

  $$\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_r, y_r)\},$$

  where $\mathbf{x}_i = (x_1, x_2, \ldots, x_n)$ is an **input vector** in a real-valued space $X \subseteq R^n$ and y_i is its **class label** (output value), $y_i \in \{1, -1\}$.

  1: positive class and -1: negative class.

- SVM finds a linear function of the form ($\mathbf{w}$: weight vector)

$$f(\mathbf{x}) = \langle \mathbf{w} \cdot \mathbf{x} \rangle + b$$

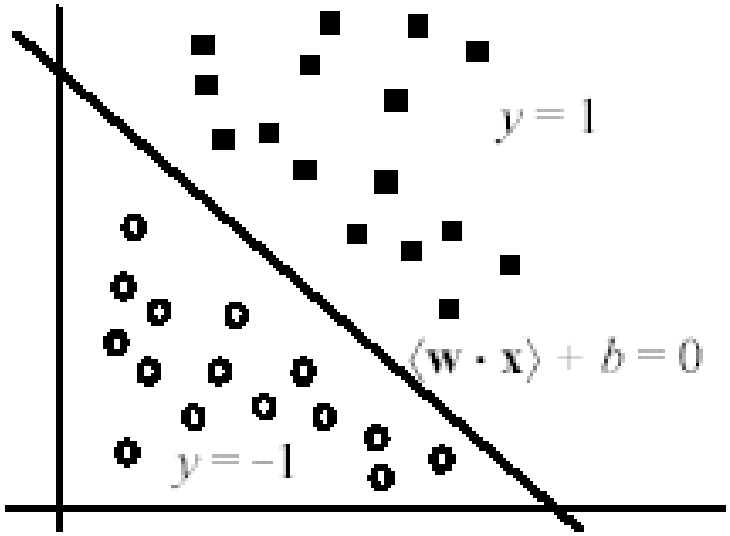
$$y_i = \begin{cases} 1 & if \ \langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b \geq 0 \\ -1 & if \ \langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b < 0 \end{cases}$$

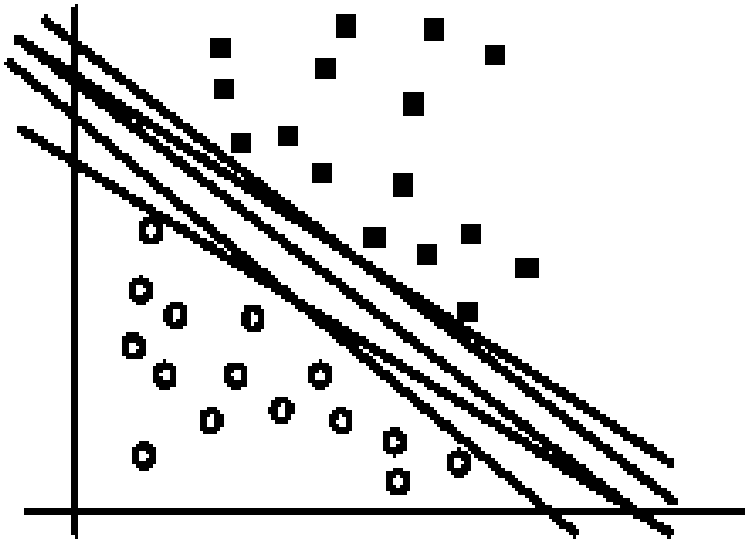
The hyperplane

- The hyperplane that separates positive and negative training data is

  $\langle \mathbf{w} \cdot \mathbf{x} \rangle + b = 0$

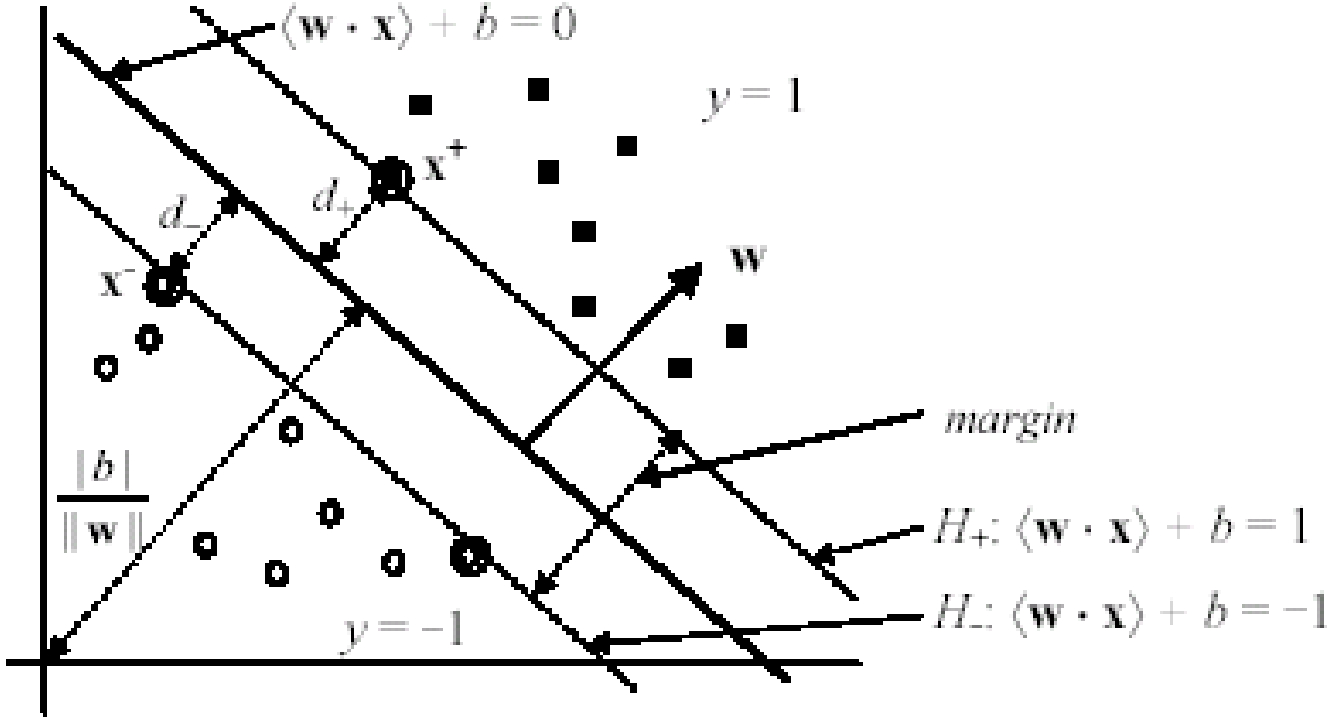
- It is also called the decision boundary (surface).
- So many possible hyperplanes, which one to choose?





Maximal margin hyperplane

- SVM looks for the separating hyperplane with the largest margin.
- Machine learning theory says this hyperplane minimizes the error bound



Linear SVM: separable case

- Assume the data are linearly separable.
- Consider a positive data point ($\mathbf{x}^+$, 1) and a negative ($\mathbf{x}^-$, -1) that are closest to the hyperplane

  $<\mathbf{w} \cdot \mathbf{x}> + b = 0$.
- We define two parallel hyperplanes, H_+ and H_-, that pass through $\mathbf{x}^+$ and $\mathbf{x}^-$ respectively. H_+ and H_- are also parallel to $<\mathbf{w} \cdot \mathbf{x}> + b = 0$.

$$H_+: \quad \langle \mathbf{w} \cdot \mathbf{x}^+ \rangle + b = 1$$

$$H_-: \quad \langle \mathbf{w} \cdot \mathbf{x}^- \rangle + b = -1$$

$$\text{such that} \quad \begin{cases} \langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b \geq 1 & \text{if } y_i = 1 \\ \langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b \leq -1 & \text{if } y_i = -1, \end{cases}$$

Compute the margin

- Now let us compute the distance between the two margin hyperplanes H_+ and H_-. Their distance is the **margin** ($d_+ + d_-$ in the figure).
- Recall from vector space in algebra that the (perpendicular) **distance** from a point $\mathbf{x}_i$ to the hyperplane $\langle \mathbf{w} \cdot \mathbf{x} \rangle + b = 0$ is:

$$\frac{|\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b|}{\|\mathbf{w}\|} \quad (36)$$

where $\|\mathbf{w}\|$ is the norm of $\mathbf{w}$,

$$\|\mathbf{w}\| = \sqrt{<\mathbf{w} \cdot \mathbf{w}>} = \sqrt{w_1^2 + w_2^2 + \ldots + w_n^2} \quad (37)$$

Compute the margin (cont.)

- Let us compute d_+.
- Instead of computing the distance from $\mathbf{x}^+$ to the separating hyperplane $\langle \mathbf{w} \cdot \mathbf{x} \rangle + b = 0$, we pick up any point $\mathbf{x}_s$ on $\langle \mathbf{w} \cdot \mathbf{x} \rangle + b = 0$ and compute the distance from $\mathbf{x}_s$ to $\langle \mathbf{w} \cdot \mathbf{x}^+ \rangle + b = 1$ by applying the distance Eq. (36) and noticing $\langle \mathbf{w} \cdot \mathbf{x}_s \rangle + b = 0$,

$$d_+ = \frac{|\langle \mathbf{w} \cdot \mathbf{x}_s \rangle + b - 1|}{\|\mathbf{w}\|} = \frac{1}{\|\mathbf{w}\|} \quad (38)$$

$$margin = d_+ + d_- = \frac{2}{\|\mathbf{w}\|} \quad (39)$$

A optimization problem!

Definition (Linear SVM: separable case): Given a set of linearly separable training examples,

$D = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_r, y_r)\}$

Learning is to solve the following constrained minimization problem,

$$\text{Minimize}: \quad \frac{\langle \mathbf{w} \cdot \mathbf{w} \rangle}{2} \tag{40}$$

$$\text{Subject to}: \quad y_i(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b) \geq 1, \quad i = 1, 2, ..., r$$

summarize

$y_i(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b) \geq 1, \quad i = 1, 2, ..., r$

$\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b \geq 1$ for $y_i = 1$

$\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b \leq -1$ for $y_i = -1$.

Solve the constrained minimization

- Standard Lagrangian method

$$L_P = \frac{1}{2} \langle \mathbf{w} \cdot \mathbf{w} \rangle - \sum_{i=1}^{r} \alpha_i [y_i (\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b) - 1] \qquad (41)$$

  where $\alpha_i \geq 0$ are the **Lagrange multipliers**.

- Optimization theory says that an optimal solution to (41) must satisfy certain conditions, called **Kuhn-Tucker conditions**, which are necessary (but not sufficient)
- Kuhn-Tucker conditions play a central role in constrained optimization.

Kuhn-Tucker conditions

$$\frac{\partial L_P}{\partial w_j} = w_j - \sum_{i=1}^{r} y_i \alpha_i \mathbf{x}_i = 0, \quad j = 1, 2, \ldots, m \tag{48}$$

$$\frac{\partial L_P}{\partial b} = -\sum_{i=1}^{r} y_i \alpha_i = 0 \tag{49}$$

$$y_i((\mathbf{w} \cdot \mathbf{x}_i) + b) - 1 \geq 0, \quad i = 1, 2, \ldots, r \tag{50}$$

$$\alpha_i \geq 0, \quad i = 1, 2, \ldots, r \tag{51}$$

$$\alpha_i (y_i((\mathbf{w} \cdot \mathbf{x}_i) + b) - 1) = 0, \quad i = 1, 2, \ldots, r \tag{52}$$

- Eq. (50) is the original set of constraints.
- The complementarity condition (52) shows that only those data points on the margin hyperplanes (i.e., H_+ and H_-) can have $\alpha_i > 0$ since for them $y_i(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b) - 1 = 0$.
- These points are called the **support vectors**, All the other parameters $\alpha_i = 0$.

Solve the problem

- In general, Kuhn-Tucker conditions are necessary for an optimal solution, but not sufficient.
- However, for our minimization problem with a convex objective function and linear constraints, the Kuhn-Tucker conditions are both **necessary** and **sufficient** for an optimal solution.
- Solving the optimization problem is still a difficult task due to the inequality constraints.
- However, the Lagrangian treatment of the convex optimization problem leads to an alternative **dual** formulation of the problem, which is easier to solve than the original problem (called the **primal**).

Dual formulation

- From primal to a dual: Setting to zero the partial derivatives of the Lagrangian (41) with respect to the **primal variables** (i.e., **w** and *b*), and substituting the resulting relations back into the Lagrangian.
  - I.e., substitute (48) and (49), into the original Lagrangian (41) to eliminate the primal variables

$$L_D = \sum_{i=1}^{r} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{r} y_i y_j \alpha_i \alpha_j \langle \mathbf{x}_i \cdot \mathbf{x}_j \rangle, \qquad (55)$$

Dual optimization prolem

$$\text{Maximize: } L_D = \sum_{i=1}^{r} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{r} y_i y_j \alpha_i \alpha_j \langle \mathbf{x}_i \cdot \mathbf{x}_j \rangle. \tag{56}$$

$$\text{Subject to: } \sum_{i=1}^{r} y_i \alpha_i = 0$$

$$\alpha_i \geq 0, \quad i = 1, 2, \ldots, r.$$

- This dual formulation is called the **Wolfe dual**.
- For the convex objective function and linear constraints of the primal, it has the property that the maximum of L_D occurs at the same values of **w**, b and α_i, as the minimum of L_P (the primal).
- Solving (56) requires numerical techniques and clever strategies, which are beyond our scope.

The final decision boundary

- After solving (56), we obtain the values for α_i, which are used to compute the weight vector **w** and the bias *b* using Equations (48) and (52) respectively.
- **The decision boundary**

$$\langle \mathbf{w} \cdot \mathbf{x} \rangle + b = \sum_{i \in sv} y_i \alpha_i \langle \mathbf{x}_i \cdot \mathbf{x} \rangle + b = 0 \qquad (57)$$

- **Testing**: Use (57). Given a test instance **z**,

$$sign(\langle \mathbf{w} \cdot \mathbf{z} \rangle + b) = sign\left[\sum_{i \in sv} \alpha_i y_i \langle \mathbf{x}_i \cdot \mathbf{z} \rangle + b \right] \qquad (58)$$

- If (58) returns 1, then the test instance **z** is classified as positive; otherwise, it is classified as negative.

Linear SVM: Non-separable case

- Linear separable case is the ideal situation.
- Real-life data may have noise or errors.
  - Class label incorrect or randomness in the application domain.
- Recall in the separable case, the problem was

$$\text{Minimize}: \frac{\langle \mathbf{w} \cdot \mathbf{w} \rangle}{2}$$

$$\text{Subject to}: y_i(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b) \geq 1, \quad i = 1, 2, ..., r$$

- With noisy data, the constraints may not be satisfied. Then, no solution!

Relax the constraints

- To allow errors in data, we relax the margin constraints by introducing **slack** variables, ξ_i (≥ 0) as follows:

  $\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b \geq 1 - \xi_i$ for $y_i = 1$

  $\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b \leq -1 + \xi_i$ for $y_i = -1$.

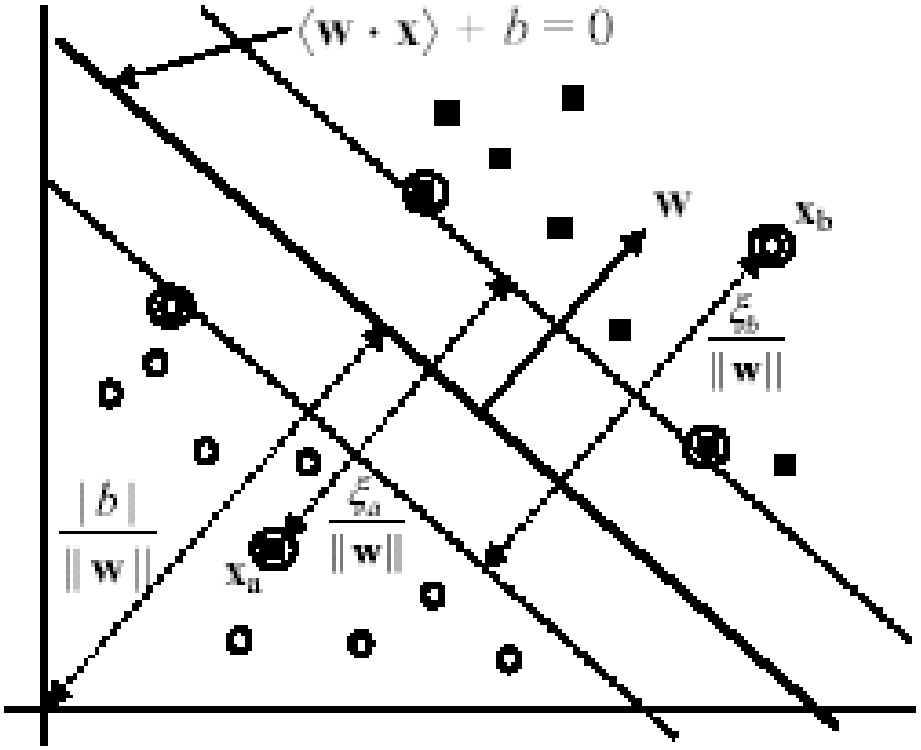
- The new constraints:

  Subject to: $y_i(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b) \geq 1 - \xi_i$, $i = 1, \ldots, r$,

  $\xi_i \geq 0$, $i = 1, 2, \ldots, r$.

Geometric interpretation

- Two error data points $\mathbf{x}_a$ and $\mathbf{x}_b$ (circled) in wrong regions



Penalize errors in objective function

- We need to penalize the errors in the objective function.
- A natural way of doing it is to assign an extra cost for errors to change the objective function to

$$\text{Minimize}: \frac{\langle \mathbf{w} \cdot \mathbf{w} \rangle}{2} + C\left(\sum_{i=1}^{r} \xi_i\right)^k \qquad (60)$$

- $k = 1$ is commonly used, which has the advantage that neither ξ_i nor its Lagrangian multipliers appear in the dual formulation.

New optimization problem

$$\text{Minimize}: \quad \frac{\langle \mathbf{w} \cdot \mathbf{w} \rangle}{2} + C\sum_{i=1}^{r} \xi_i \tag{61}$$

$$\text{Subject to}: \quad y_i(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b) \geq 1 - \xi_i, \quad i = 1, 2, ..., r$$

$$\xi_i \geq 0, \quad i = 1, 2, ..., r$$

- This formulation is called the **soft-margin SVM**. The primal Lagrangian is

$$L_P = \frac{1}{2}\langle \mathbf{w} \cdot \mathbf{w} \rangle + C\sum_{i=1}^{r} \xi_i - \sum_{i=1}^{r} \alpha_i [y_i(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b) - 1 + \xi_i] - \sum_{i=1}^{r} \mu_i \xi_i \tag{62}$$

where α_i, $\mu_i \geq 0$ are the **Lagrange multipliers**

Kuhn-Tucker conditions

$$\frac{\partial L_p}{\partial w_j} = w_j - \sum_{i=1}^{r} y_i \alpha_i \mathbf{x}_i = 0, \quad j = 1, 2, \ldots, m \tag{63}$$

$$\frac{\partial L_p}{\partial b} = -\sum_{i=1}^{r} y_i \alpha_i = 0 \tag{64}$$

$$\frac{\partial L_p}{\partial \xi_i} = C - \alpha_i - \mu_i = 0, \quad i = 1, 2, \ldots, r \tag{65}$$

$$y_i(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b) - 1 + \xi_i \geq 0, \quad i = 1, 2, \ldots, r \tag{66}$$

$$\xi_i \geq 0, \quad i = 1, 2, \ldots, r \tag{67}$$

$$\alpha_i \geq 0, \quad i = 1, 2, \ldots, r \tag{68}$$

$$\mu_i \geq 0, \quad i = 1, 2, \ldots, r \tag{69}$$

$$\alpha_i (y_i(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b) - 1 + \xi_i) = 0, \quad i = 1, 2, \ldots, r \tag{70}$$

$$\mu_i \xi_i = 0, \quad i = 1, 2, \ldots, r \tag{71}$$

From primal to dual

- As the linear separable case, we transform the primal to a dual by setting to zero the partial derivatives of the Lagrangian (62) with respect to the **primal variables** (i.e., **w**, *b* and ξ_i), and substituting the resulting relations back into the Lagrangian.
- Ie.., we substitute Equations (63), (64) and (65) into the primal Lagrangian (62).
- From Equation (65), $C - \alpha_i - \mu_i = 0$, we can deduce that $\alpha_i \leq C$ because $\mu_i \geq 0$.

Dual

- The dual of (61) is

$$\text{Maximize: } L_D(\boldsymbol{\alpha}) = \sum_{i=1}^{r} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{r} y_i y_j \alpha_i \alpha_j \langle \mathbf{x}_i \cdot \mathbf{x}_j \rangle. \qquad (72)$$

$$\text{Subject to: } \sum_{i=1}^{r} y_i \alpha_i = 0$$

$$0 \le \alpha_i \le C, \quad i = 1, 2, \ldots, r.$$

- Interestingly, ξ_i and its Lagrange multipliers μ_i are not in the dual. The objective function is identical to that for the separable case.
- The only difference is the constraint $\alpha_i \le C$.

Find primal variable values

- The dual problem (72) can be solved numerically.
- The resulting α_i values are then used to compute **w** and *b*. **w** is computed using Equation (63) and *b* is computed using the Kuhn-Tucker complementarity conditions (70) and (71).
- Since no values for ξ_i, we need to get around it.
  - From Equations (65), (70) and (71), we observe that if $0 < \alpha_i < C$ then both $\xi_i = 0$ and $y_i(\mathbf{w} \cdot \mathbf{x}_i + b - 1 + \xi_i = 0$. Thus, we can use any training data point for which $0 < \alpha_i < C$ and Equation (69) (with $\xi_i = 0$) to compute *b.*

$$b = \frac{1}{y_i} - \sum_{i=1}^{r} y_i \alpha_i (\mathbf{x}_i \cdot \mathbf{x}_j) = 0. \quad (73)$$

(65), (70) and (71) in fact tell us more

$$\begin{aligned} \alpha_i = 0 &\Rightarrow y_i(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b) \geq 1 \text{ and } \xi_i = 0 \\ 0 < \alpha_i < C &\Rightarrow y_i(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b) = 1 \text{ and } \xi_i = 0 \\ \alpha_i = C &\Rightarrow y_i(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b) \leq 1 \text{ and } \xi_i \geq 0 \end{aligned} \quad (74)$$

- (74) shows a very important property of SVM.
  - The solution is **sparse** in α_i. Many training data points are outside the margin area and their α_i's in the solution are 0.
  - Only those data points that are on the margin (i.e., $y_i(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b) = 1$, which are support vectors in the separable case), inside the margin (i.e., $\alpha_i = C$ and $y_i(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b) < 1$), or errors are non-zero.
  - Without this sparsity property, SVM would not be practical for large data sets.

The final decision boundary

- The final decision boundary is (we note that many α_i's are 0)

$$\langle \mathbf{w} \cdot \mathbf{x} \rangle + b = \sum_{i=1}^{r} y_i \alpha_i \langle \mathbf{x}_i \cdot \mathbf{x} \rangle + b = 0 \qquad (75)$$

- The decision rule for classification (testing) is the same as the separable case, i.e.,

$$sign(\langle \mathbf{w} \cdot \mathbf{x} \rangle + b).$$

- Finally, we also need to determine the parameter *C* in the objective function. It is normally chosen through the use of a validation set or cross-validation.

How to deal with nonlinear separation?

- The SVM formulations require linear separation.
- Real-life data sets may need nonlinear separation.
- To deal with nonlinear separation, the same formulation and techniques as for the linear case are still used.
- We only transform the input data into another space (usually of a much higher dimension) so that
  - a linear decision boundary can separate positive and negative examples in the transformed space,
- The transformed space is called the **feature space**. The original data space is called the **input space**.

Space transformation

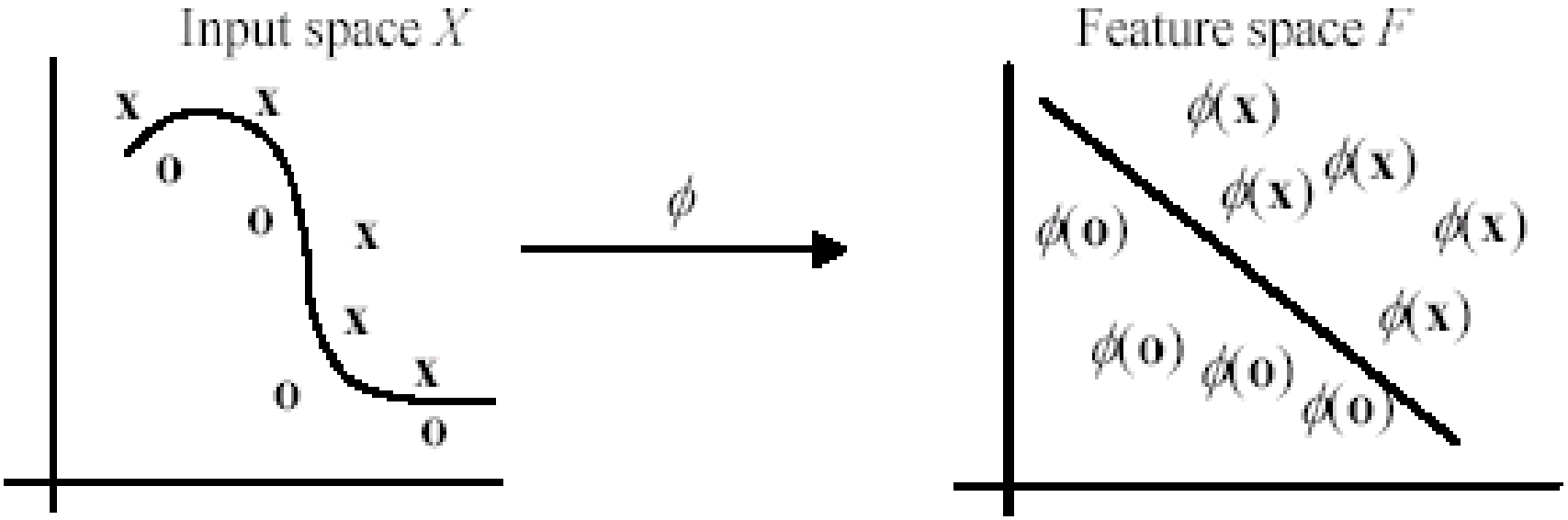
- The basic idea is to map the data in the input space X to a feature space F via a nonlinear mapping ϕ,

$$\phi : X \to F$$
$$\mathbf{x} \mapsto \phi(\mathbf{x}) \qquad (76)$$

- After the mapping, the original training data set $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_r, y_r)\}$ becomes:

$$\{(\phi(\mathbf{x}_1), y_1), (\phi(\mathbf{x}_2), y_2), \ldots, (\phi(\mathbf{x}_r), y_r)\} \qquad (77)$$

Geometric interpretation



- In this example, the transformed space is also 2-D. But usually, the number of dimensions in the feature space is much higher than that in the input space

Optimization problem in (61) b

With the transformation, the optimization problem in (61) becomes

$$\text{Minimize}: \quad \frac{\langle \mathbf{w} \cdot \mathbf{w} \rangle}{2} + C\sum_{i=1}^{r} \xi_i \tag{78}$$

$$\text{Subject to}: \quad y_i(\langle \mathbf{w} \cdot \phi(\mathbf{x}_i) \rangle + b) \geq 1 - \xi_i, \quad i = 1, 2, ..., r$$

$$\xi_i \geq 0, \quad i = 1, 2, ..., r$$

The dual is

$$\text{Maximize}: \quad L_D = \sum_{i=1}^{r} \alpha_i - \frac{1}{2}\sum_{i,j=1}^{r} y_i y_j \alpha_i \alpha_j \langle \phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}_j) \rangle. \tag{79}$$

$$\text{Subject to}: \quad \sum_{i=1}^{r} y_i \alpha_i = 0$$

$$0 \leq \alpha_i \leq C, \quad i = 1, 2, ..., r.$$

The final decision rule for classification (testing) is

$$\sum_{i=1}^{r} y_i \alpha_i \langle \phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}) \rangle + b \tag{80}$$

An example space transformation

- Suppose our input space is 2-dimensional, and we choose the following transformation (mapping) from 2-D to 3-D:

$$(x_1, x_2) \rightarrow (x_1^2, x_2^2, \sqrt{2}x_1x_2)$$

- The training example ((2, 3), -1) in the input space is transformed to the following in the feature space:

((4, 9, 8.5), -1)

Problem with explicit transformation

- The potential problem with this explicit data transformation and then applying the linear SVM is that it may suffer from the curse of dimensionality.
- The number of dimensions in the feature space can be huge with some useful transformations even with reasonable numbers of attributes in the input space.
- This makes it computationally infeasible to handle.
- Fortunately, explicit transformation is not needed.

Kernel functions

- We notice that in the dual formulation both
  - the construction of the optimal hyperplane (79) in *F* and
  - the evaluation of the corresponding decision function (80)

  only require dot products $\langle \phi(\mathbf{x}) \cdot \phi(\mathbf{z}) \rangle$ and never the mapped vector $\phi(\mathbf{x})$ in its explicit form. This is a crucial point.
- Thus, if we have a way to compute the dot product $\langle \phi(\mathbf{x}) \cdot \phi(\mathbf{z}) \rangle$ using the input vectors **x** and **z** directly,
  - no need to know the feature vector $\phi(\mathbf{x})$ or even ϕ itself.
- In SVM, this is done through the use of **kernel functions**, denoted by *K*,

$$K(\mathbf{x}, \mathbf{z}) = \langle \phi(\mathbf{x}) \cdot \phi(\mathbf{z}) \rangle \quad (82)$$

An example kernel function

- Polynomial kernel

$$K(\mathbf{x}, \mathbf{z}) = \langle \mathbf{x} \cdot \mathbf{z} \rangle^d \quad (83)$$

- Let us compute the kernel with degree $d = 2$ in a 2-dimensional space: $\mathbf{x} = (x_1, x_2)$ and $\mathbf{z} = (z_1, z_2)$.

$$\begin{aligned}
\langle \mathbf{x} \cdot \mathbf{z} \rangle^2 &= (x_1 z_1 + x_2 z_2)^2 \\
&= x_1^2 z_1^2 + 2 x_1 z_1 x_2 z_2 + x_2^2 z_2^2 \\
&= \langle (x_1^2, x_2^2, \sqrt{2} x_1 x_2) \cdot (z_1^2, z_2^2, \sqrt{2} z_1 z_2) \rangle \\
&= \langle \phi(\mathbf{x}) \cdot \phi(\mathbf{z}) \rangle,
\end{aligned} \quad (84)$$

- This shows that the kernel $\langle \mathbf{x} \cdot \mathbf{z} \rangle^2$ is a dot product in a transformed feature space

Kernel trick

- The derivation in (84) is only for illustration purposes.
- We do not need to find the mapping function.
- We can simply apply the kernel function directly by
  - replace all the dot products $\langle \phi(\mathbf{x}) \cdot \phi(\mathbf{z}) \rangle$ in (79) and (80) with the kernel function $K(\mathbf{x}, \mathbf{z})$ (e.g., the polynomial kernel $\langle \mathbf{x} \cdot \mathbf{z} \rangle^d$ in (83)).
- This strategy is called the **kernel trick**.

Is it a kernel function?

- The question is: how do we know whether a function is a kernel without performing the derivation such as that in (84)? I.e,
  - How do we know that a kernel function is indeed a dot product in some feature space?
- This question is answered by a theorem called the Mercer's theorem, which we will not discuss here.

Commonly used kernels

- It is clear that the idea of kernel generalizes the dot product in the input space. This dot product is also a kernel with the feature map being the identity

$$K(\mathbf{x}, \mathbf{z}) = \langle \mathbf{x} \cdot \mathbf{z} \rangle. \quad (85)$$

Commonly used kernels include

$$\text{Polynomial:} \quad K(\mathbf{x}, \mathbf{z}) = (\langle \mathbf{x} \cdot \mathbf{z} \rangle + \theta)^d \quad (86)$$

$$\text{Gaussian RBF:} \quad K(\mathbf{x}, \mathbf{z}) = e^{-\|\mathbf{x}-\mathbf{z}\|^2/2\sigma} \quad (87)$$

$$\text{Sigmoidal:} \quad K(\mathbf{x}, \mathbf{z}) = \tanh(k\langle \mathbf{x} \cdot \mathbf{z} \rangle - \delta) \quad (88)$$

where $\theta \in R$, $d \in N$, $\sigma > 0$, and $k, \delta \in R$.

Some other issues in SVM

- SVM works only in a real-valued space. For a categorical attribute, we need to convert its categorical values to numeric values.
- SVM does only two-class classification. For multi-class problems, some strategies can be applied, e.g., one-against-rest, and error-correcting output coding.
- The hyperplane produced by SVM is hard to understand by human users. The matter is made worse by kernels. Thus, SVM is commonly used in applications that do not required human understanding.

Road Map

- Basic concepts
- Decision tree induction
- Evaluation of classifiers
- Rule induction
- Classification using association rules
- Naïve Bayesian classification
- Naïve Bayes for text classification
- Support vector machines
- **K-nearest neighbor**
- Ensemble methods: Bagging and Boosting
- Summary

k-Nearest Neighbor Classification (kNN)

- Unlike all the previous learning methods, kNN does not build model from the training data.
- To classify a test instance *d*, define *k*-neighborhood *P* as *k* nearest neighbors of *d*
- Count number *n* of training instances in *P* that belong to class c_j
- Estimate Pr(c_j|*d*) as *n*/*k*
- No training is needed. Classification time is linear in training set size for each test case.

kNNAlgorithm

Algorithm kNN(D, d, k)

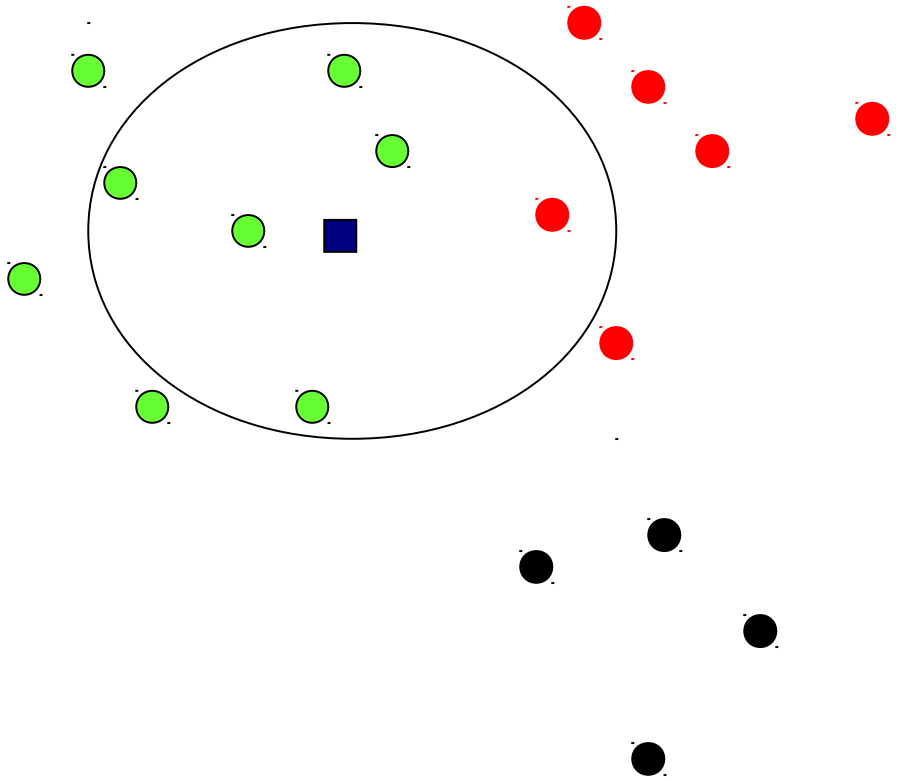
1 Compute the distance between d and every example in D;

2 Choose the k examples in D that are nearest to d, denote the set by P ($\subseteq D$);

3 Assign d the class that is the most frequent class in P (or the majority class);

- k is usually chosen empirically via a validation set or cross-validation by trying a range of k values.
- Distance function is crucial, but depends on applications.

Example: k=6 (6NN)



Government

Science

Arts

A new point ■

Pr(science| ■)

?

Discussions

- kNN can deal with complex and arbitrary decision boundaries.
- Despite its simplicity, researchers have shown that the classification accuracy of kNN can be quite strong and in many cases as accurate as those elaborated methods.
- kNN is slow at the classification time
- kNN does not produce an understandable model

Road Map

- Basic concepts
- Decision tree induction
- Evaluation of classifiers
- Rule induction
- Classification using association rules
- Naïve Bayesian classification
- Naïve Bayes for text classification
- Support vector machines
- K-nearest neighbor
- **Ensemble methods: Bagging and Boosting**
- Summary

Combining classifiers

- So far, we have only discussed individual classifiers, i.e., how to build them and use them.
- Can we combine multiple classifiers to produce a better classifier?
- Yes, sometimes
- We discuss two main algorithms:
  - Bagging
  - Boosting

Bagging

- Breiman, 1996
- Bootstrap Aggregating = Bagging
  - Application of bootstrap sampling
    - Given: set *D* containing *m* training examples
    - Create a sample *S*[*i*] of *D* by drawing *m* examples at random *with replacement* from *D*
    - *S*[*i*] of size *m*: expected to leave out 0.37 of examples from *D*

Bagging (cont…)

- Training
  - Create *k* bootstrap samples *S*[1], *S*[2], …, *S*[*k*]
  - Build a distinct classifier on each *S*[*i*] to produce *k* classifiers, using the same learning algorithm.
- Testing
  - Classify each new instance by voting of the *k* classifiers (equal weights)

Bagging Example

Original	1	2	3	4	5	6	7	8
Training set 1	2	7	8	3	7	6	3	1
Training set 2	7	8	5	6	4	2	7	1
Training set 3	3	6	2	7	5	6	2	2
Training set 4	4	5	1	4	6	4	3	8

Bagging (cont …)

- When does it help?
  - When learner is unstable
    - Small change to training set causes large change in the output classifier
    - True for decision trees, neural networks; not true for *k*-nearest neighbor, naïve Bayesian, class association rules
  - Experimentally, bagging can help substantially for unstable learners, may somewhat degrade results for stable learners

Boosting

- A family of methods:
  - We only study **AdaBoost** (Freund & Schapire, 1996)
- Training
  - Produce a sequence of classifiers (the same base learner)
  - Each classifier is dependent on the previous one, and focuses on the previous one's errors
  - Examples that are incorrectly predicted in previous classifiers are given higher weights
- Testing
  - For a test case, the results of the series of classifiers are combined to determine the final class of the test case.

AdaBoost

Weighted training set

(x_1, y_1, w_1)
(x_2, y_2, w_2)
...
(x_n, y_n, w_n)

called a weaker classifier

- Build a classifier h_t whose accuracy on training set > ½ (better than random)

Non-negative weights sum to 1

Change weights

AdaBoost algorithm

Algorithm AdaBoost.M1

Input: sequence of m examples $\langle (x_1, y_1), \ldots, (x_m, y_m) \rangle$
with labels $y_i \in Y = \{1, \ldots, k\}$
weak learning algorithm **WeakLearn**
integer T specifying number of iterations

Initialize $D_1(i) = 1/m$ for all i.

Do for $t = 1, 2, \ldots, T$:

1. Call **WeakLearn**, providing it with the distribution D_t.
2. Get back a hypothesis $h_t : X \rightarrow Y$.
3. Calculate the error of h_t: $\epsilon_t = \sum_{i:h_t(x_i) \neq y_i} D_t(i)$.

   If $\epsilon_t > 1/2$, then set $T = t - 1$ and abort loop.
4. Set $\beta_t = \epsilon_t / (1 - \epsilon_t)$.
5. Update distribution D_t:

$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} \beta_t & \text{if } h_t(x_i) = y_i \\ 1 & \text{otherwise} \end{cases}$$

   where Z_t is a normalization constant (chosen so that D_{t+1} will be a distribution).

Output the final hypothesis:

$$h_{fin}(x) = \arg\max_{y \in Y} \sum_{t:h_t(x)=y} \log \frac{1}{\beta_t}.$$

Bagging, Boosting and C4.5

C4.5's mean error rate over the 10 cross-validation.

Bagged C4.5 vs. C4.5.

Boosted C4.5 vs. C4.5.

Boosting vs. Bagging

	C4.5	Bagged C4.5 vs C4.5			Boosted C4.5 vs C4.5			Boosting vs Bagging	
	err (%)	err (%)	w-l	ratio	err (%)	w-l	ratio	w-l	ratio
anneal	7.67	6.25	10-0	.814	4.73	10-0	.617	10-0	.758
audiology	22.12	19.29	9-0	.872	15.71	10-0	.710	10-0	.814
auto	17.66	19.66	2-8	1.113	15.22	9-1	.862	9-1	.774
breast-w	5.28	4.23	9-0	.802	4.09	9-0	.775	7-2	.966
chess	8.55	8.33	6-2	.975	4.59	10-0	.537	10-0	.551
colic	14.92	15.19	0-6	1.018	18.83	0-10	1.262	0-10	1.240
credit-a	14.70	14.13	8-2	.962	15.64	1-9	1.064	0-10	1.107
credit-g	28.44	25.81	10-0	.908	29.14	2-8	1.025	0-10	1.129
diabetes	25.39	23.63	9-1	.931	28.18	0-10	1.110	0-10	1.192
glass	32.48	27.01	10-0	.832	23.55	10-0	.725	9-1	.872
heart-c	22.94	21.52	7-2	.938	21.39	8-0	.932	5-4	.994
heart-h	21.53	20.31	8-1	.943	21.05	5-4	.978	3-6	1.037
hepatitis	20.39	18.52	9-0	.908	17.68	10-0	.867	6-1	.955
hypo	.48	.45	7-2	.928	.36	9-1	.746	9-1	.804
iris	4.80	5.13	2-6	1.069	6.53	0-10	1.361	0-8	1.273
labor	19.12	14.39	10-0	.752	13.86	9-1	.725	5-3	.963
letter	11.99	7.51	10-0	.626	4.66	10-0	.389	10-0	.621
lymphography	21.69	20.41	8-2	.941	17.43	10-0	.804	10-0	.854
phoneme	19.44	18.73	10-0	.964	16.36	10-0	.842	10-0	.873
segment	3.21	2.74	9-1	.853	1.87	10-0	.583	10-0	.684
sick	1.34	1.22	7-1	.907	1.05	10-0	.781	9-1	.861
sonar	25.62	23.80	7-1	.929	19.62	10-0	.766	10-0	.824
soybean	7.73	7.58	6-3	.981	7.16	8-2	.926	8-1	.944
splice	5.91	5.58	9-1	.943	5.43	9-0	.919	6-4	.974
vehicle	27.09	25.54	10-0	.943	22.72	10-0	.839	10-0	.889
vote	5.06	4.37	9-0	.864	5.29	3-6	1.046	1-9	1.211
waveform	27.33	19.77	10-0	.723	18.53	10-0	.678	8-2	.938
average	*15.66*	*14.11*		*.905*	*13.36*		*.847*		*.930*

Does AdaBoost always work?

- The actual performance of boosting depends on the data and the base learner.
  - It requires the base learner to be unstable as bagging.
- Boosting seems to be susceptible to noise.
  - When the number of outliners is very large, the emphasis placed on the hard examples can hurt the performance.

Road Map

- Basic concepts
- Decision tree induction
- Evaluation of classifiers
- Rule induction
- Classification using association rules
- Naïve Bayesian classification
- Naïve Bayes for text classification
- Support vector machines
- K-nearest neighbor
- **Summary**

Summary

- Applications of supervised learning are in almost any field or domain.
- We studied 8 classification techniques.
- There are still many other methods, e.g.,
  - Bayesian networks
  - Neural networks
  - Genetic algorithms
  - Fuzzy classification

  This large number of methods also show the importance of classification and its wide applicability.
- It remains to be an active research area.