Chapter 6: Classification

Jilles Vreeken

IRDM '15/16

17 Nov 2015

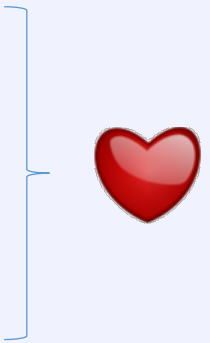
UNIVERSITÄT DES SAARLANDES

CLUSTER OF EXCELLENCE

max planck institut informatik

IRDM Chapter 6, overview

1. Basic idea
2. Instance-based classification
3. Decision trees
4. Probabilistic classification



You'll find this covered in
Aggarwal Ch. 10
Zaki & Meira, Ch. 18, 19, (22)

Chapter 6.1:
The Basic Idea

Aggarwal Ch. 10.1-10.2

Definitions

Data for classification comes in tuples (x, y)

- vector x is the attribute (feature) set
  - attributes can be binary, categorical or numerical
- value y is the class label
  - we concentrate on binary or nominal class labels
  - compare classification with regression!

A classifier is a function that maps attribute sets to class labels, $f(x) = y$

Attribute set Class

TID	Home Owner	Marital Status	Annual Income	Defaulted Borrower
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

Classification function as a black box

Descriptive vs. Predictive

In descriptive data mining the goal is to give a description of the data

- those who have bought diapers have also bought beer
- these are the clusters of documents from this corpus

In predictive data mining the goal is to predict the future

- those who will buy diapers will also buy beer
- if new documents arrive, they will be similar to one of the cluster centroids

The difference between predictive data mining and machine learning is hard to define

Descriptive vs. Predictive

In descriptive data mining the goal is to give a descripti

- those w
- these ar

In predic

- those w
- if new d
  centroid

In Data Mining we care more about **insightfulness** than prediction performance

The difference between predictive data mining and machine learning is hard to define

Descriptive vs. Predictive

Who are the borrowers that will default?

- descriptive

If a new borrower comes, will they default?

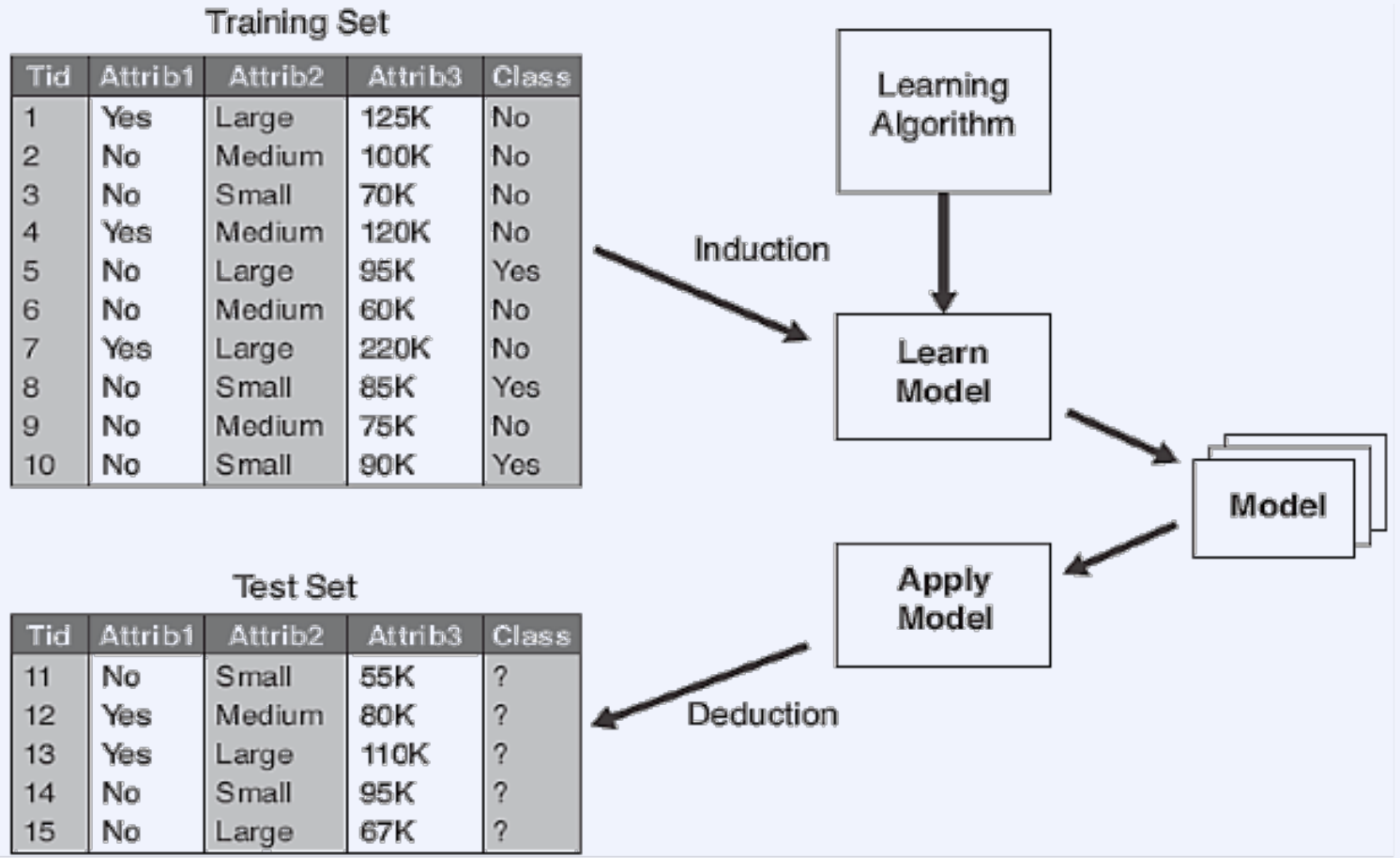
- predictive

Predictive classification is the usual application

- and what we concentrate on

TID	Home Owner	Marital Status	Annual Income	Defaulted Borrower
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

General classification Framework



Classification model evaluation

Recall contingency tables

- a **confusion matrix** is simply a contingency table between actual and predicted class labels

Predicted class

Actual class		**Class=1**	**Class=0**
	Class=1	s_{11}	s_{10}
	Class=0	s_{01}	s_{00}

Many measures available

- we focus on **accuracy** and **error rate**

$$accuracy = \frac{s_{11}+s_{00}}{s_{11}+s_{00}+s_{10}+s_{01}}$$

$$error\ rate = \frac{s_{10}+s_{01}}{s_{11}+s_{00}+s_{10}+s_{01}} =$$

$$P(f(x) \neq y) =$$
$$P(f(x) = 1, y = -1) + P(f(x) = -1, y = 1) =$$
$$p(\,f(x) = 1 \mid y = -1\,)P(y = -1) + P(\,f(x) = -1 \mid y = 1\,)P(y = 1)$$

- there's also precision, recall, F-scores, etc.

(here I use the s_{ij} notation to make clear we consider absolute numbers, in the wild f_{ij} can mean either absolute or relative – pay close attention)

Supervised vs. unsupervised learning

In **supervised learning**

- training data is accompanied by class labels
- new data is classified based on the training set
  - classification

In **unsupervised learning**

- the class labels are unknown
- the aim is to establish the existence of classes in the data, based on measurements, observations, etc.
  - clustering

Chapter 6.2:
Instance-based classification

Aggarwal Ch. 10.8

Classification per instance

Let us first consider the most simple effective classifier

"similar instances have similar labels"

Key idea is to find instances in the training data that are similar to the test instance.

k-Nearest Neighbors

The most basic classifier is **k-nearest neighbours**

Given database $\boldsymbol{D}$ of labeled instances, a distance function d, and parameter k, for test instance $\boldsymbol{x}$, find the k instances from $\boldsymbol{D}$ most similar to $\boldsymbol{x}$, and assign it the **majority label** over this top-k.

We can make it more locally-sensitive by weighing by distance δ

$$f(\delta) = e^{-\delta^2/t^2}$$

k-Nearest Neighbors, ctd.

kNN classifiers work surprisingly well in practice, iff we have ample training data and your distance function is chosen wisely

How to choose k?

- odd, to avoid ties.
- not too small, or it will not be robust against noise
- not too large, or it will lose local sensitivity

Computational complexity

- training is instant, $O(0)$
- testing is slow, $O(n)$

Chapter 6.3: Decision Trees

Aggarwal Ch. 10.3-10.4

Basic idea

We define the label by asking **series of questions** about the attributes

- each question depends on the answer to the previous one
- ultimately, all samples with satisfying attribute values have the same label and we're done

The flow-chart of the questions can be drawn as a tree

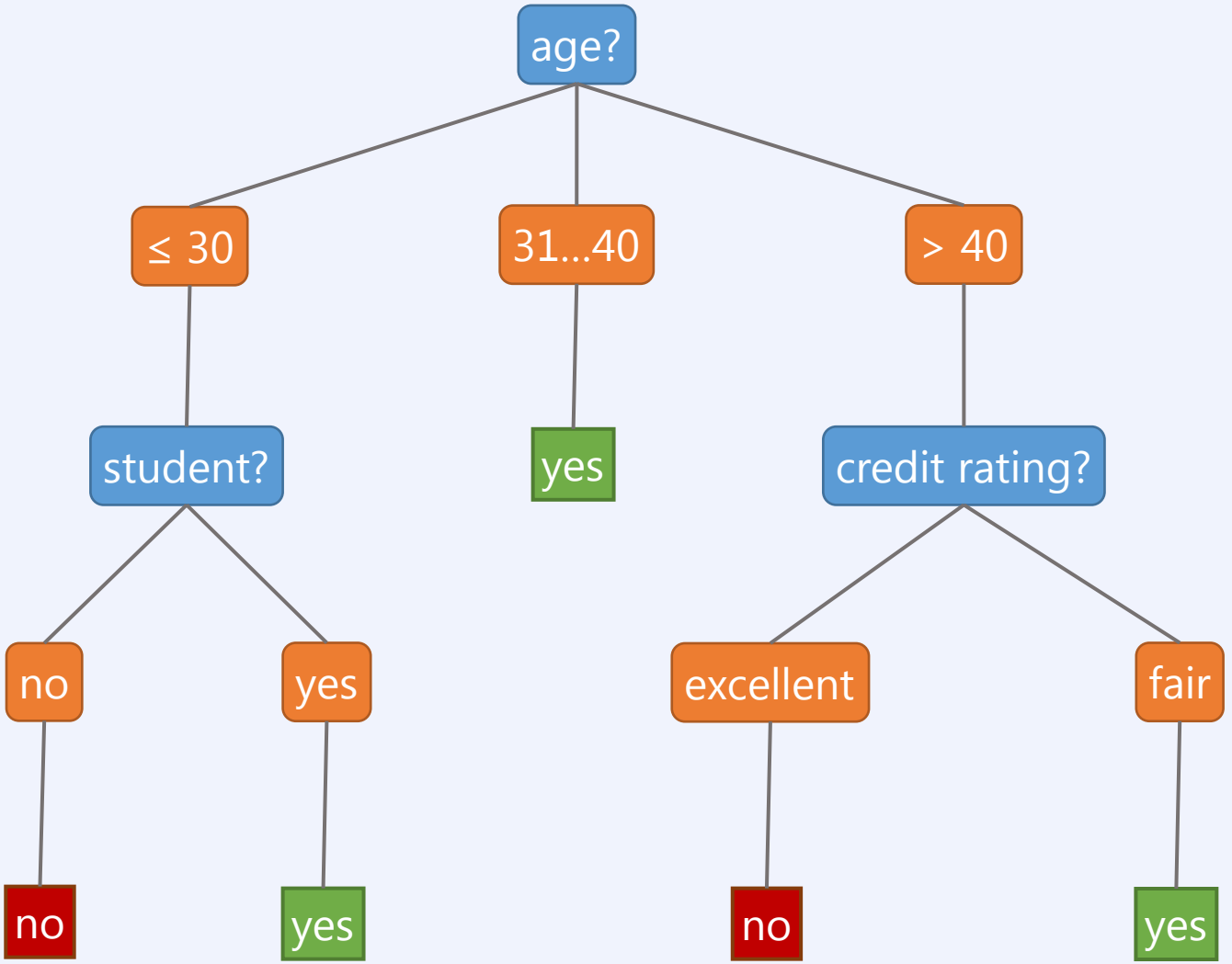
We can classify new instances by following the proper edges of the tree until we meet a leaf

- decision tree leafs are always class labels

Example: training data

age	income	student	credit_rating	buys PS4
≤ 30	high	no	fair	**no**
≤ 30	high	no	excellent	**no**
$30 \ldots 40$	high	no	fair	**yes**
> 40	medium	no	fair	**yes**
> 40	low	yes	fair	**yes**
> 40	low	yes	excellent	**no**
$30 \ldots 40$	low	yes	excellent	**yes**
≤ 30	medium	no	fair	**no**
≤ 30	low	Yes	fair	**yes**
> 40	medium	yes	fair	**yes**
≤ 30	medium	yes	excellent	**yes**
$30 \ldots 40$	medium	no	excellent	**yes**
$30 \ldots 40$	high	yes	fair	**yes**
> 40	medium	no	excellent	**no**

Example: decision tree



Hunt's algorithm

The number of decision trees for a given set of attributes is exponential

Finding the most accurate tree is NP-hard

Practical algorithms use **greedy heuristics**

- the decision tree is grown by making a series of locally optimal decisions on which attributes to use and how to split on them

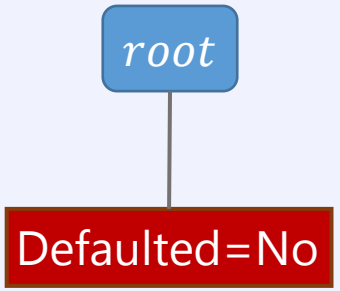
Most algorithms are based on Hunt's algorithm

Hunt's algorithm

1. **Let** X_t be the set of training records for node t
2. **Let** $y = \{y_1, \ldots, y_c\}$ be the class labels
3. **If** X_t contains records that belong to more than one class
    1. select attribute test condition to partition the records into smaller subsets
    2. create a child node for each outcome of test condition
    3. apply algorithm recursively to each child
4. **else if** all records in X_t belong to the same class y_i, **then** t is a leaf node with label y_i

Example: Decision tree

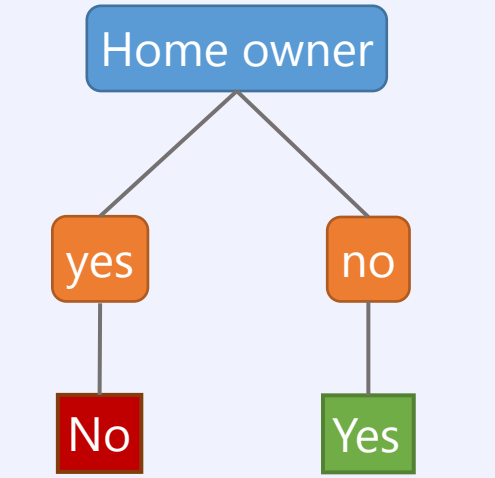
TID	Home Owner	Marital Status	Annual Income	Defaulted Borrower
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes



Has multiple labels,
best label = 'no'

Example: Decision tree

TID	Home Owner	Marital Status	Annual Income	Defaulted Borrower
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

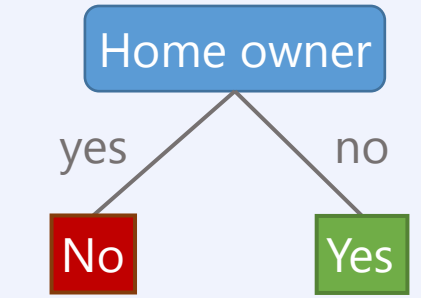


Only one label

Has multiple labels

Example: Decision tree

TID	Home Owner	Marital Status	Annual Income	Defaulted Borrower
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

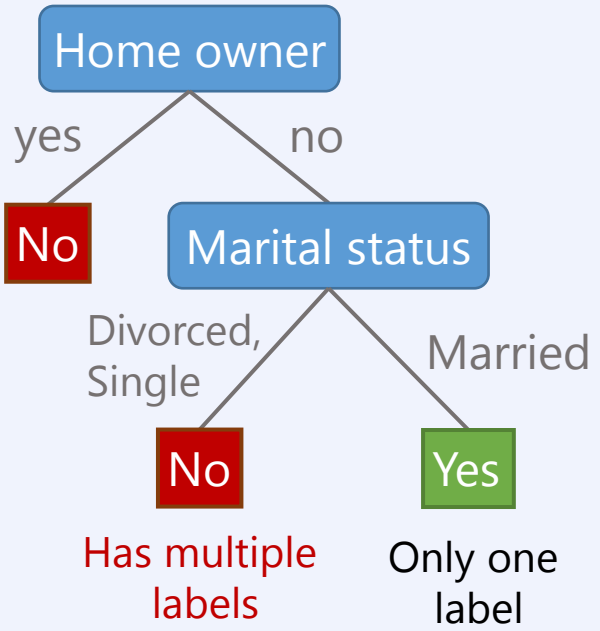


Only one label

Has multiple labels

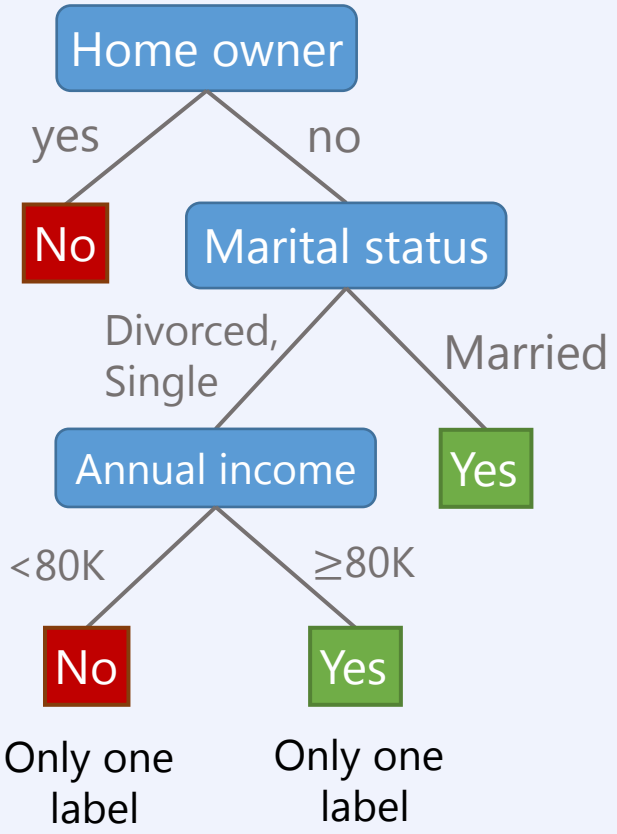
Example: Decision tree

TID	Home Owner	Marital Status	Annual Income	Defaulted Borrower
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes



Example: Decision tree

TID	Home Owner	Marital Status	Annual Income	Defaulted Borrower
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes



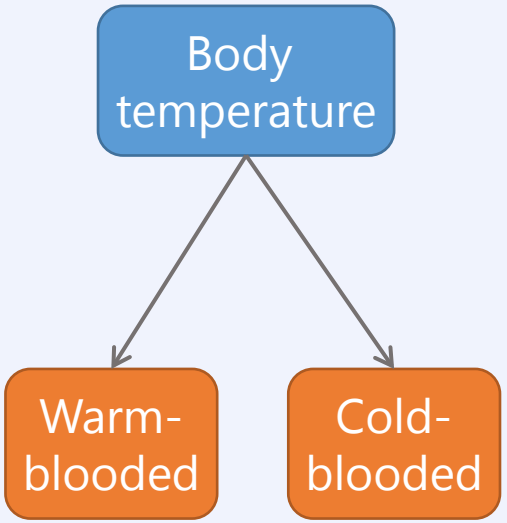
Selecting the split

Designing a decision-tree algorithm requires answering two questions

1. How should we split the training records?
2. How should we stop the splitting procedure?

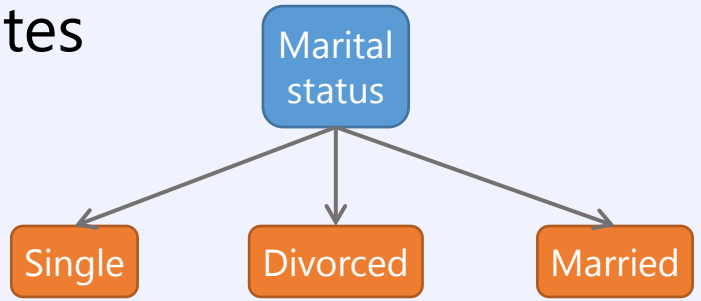
Splitting methods

Binary attributes

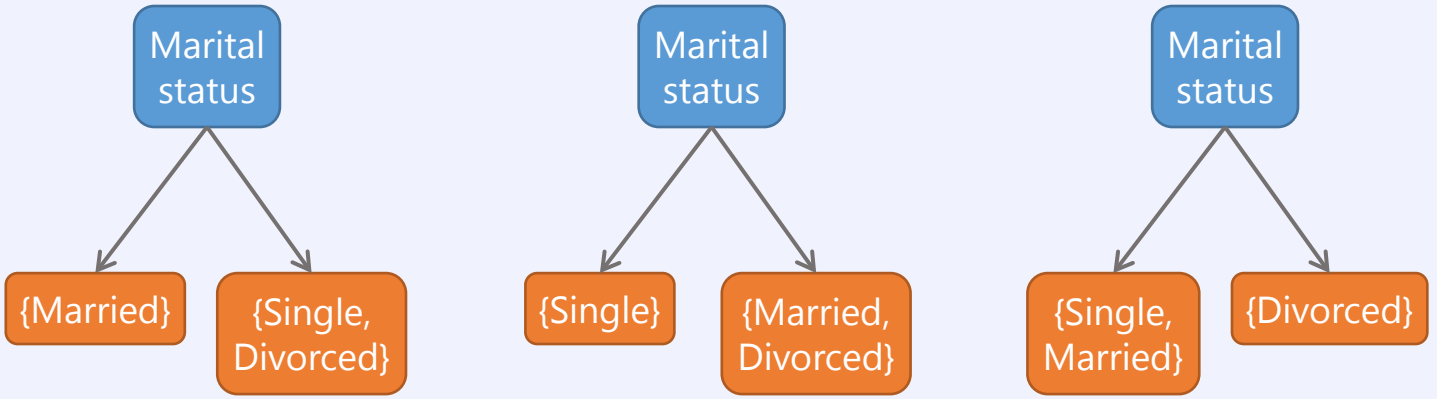


Splitting methods

Nominal attributes



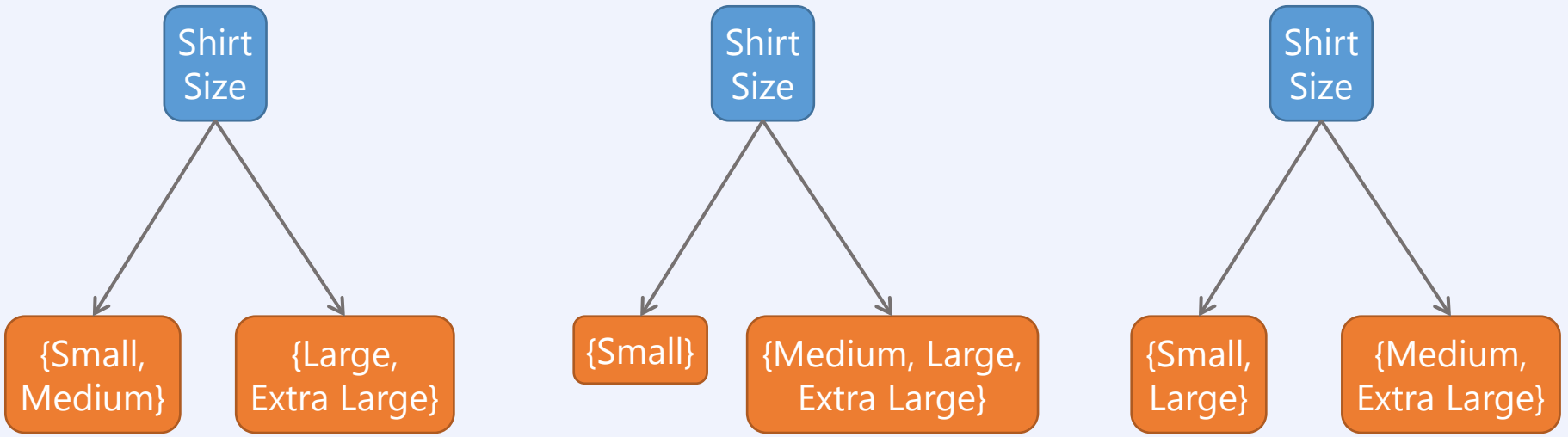
Multiway split



Binary split

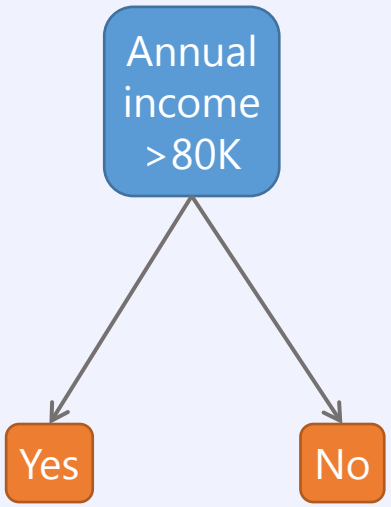
Splitting methods

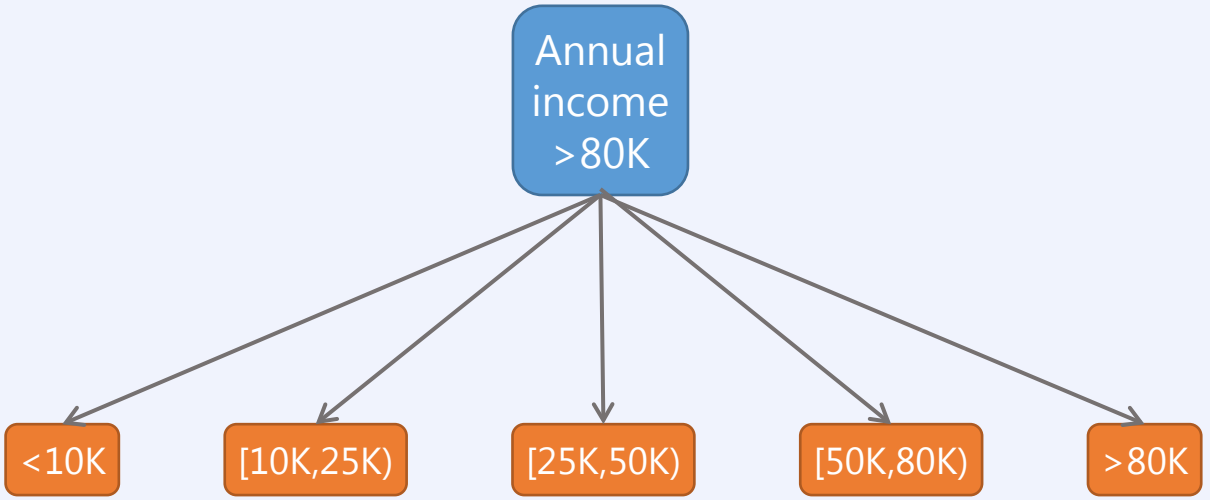
Ordinal attributes



Splitting methods

Numeric attributes





Selecting the best split

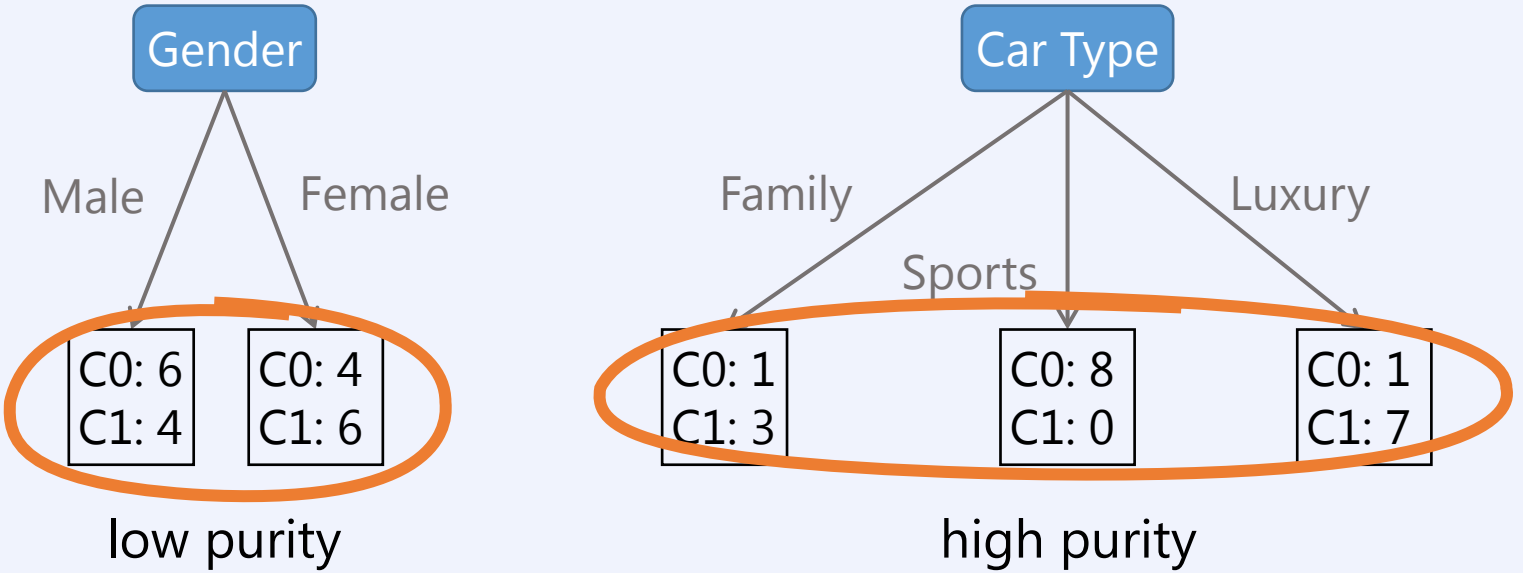
Let $p(i \mid t)$ be the fraction of records of class i in node t

The **best split** is selected based on the degree of impurity of the child nodes

- $p(0 \mid t) = 0$ and $p(1 \mid t) = 1$ has **high purity**
- $p(0 \mid t) = 1/2$ and $p(1 \mid t) = 1/2$ has the **smallest purity**

Intuition:
high purity → better split

Example of purity



Impurity measures

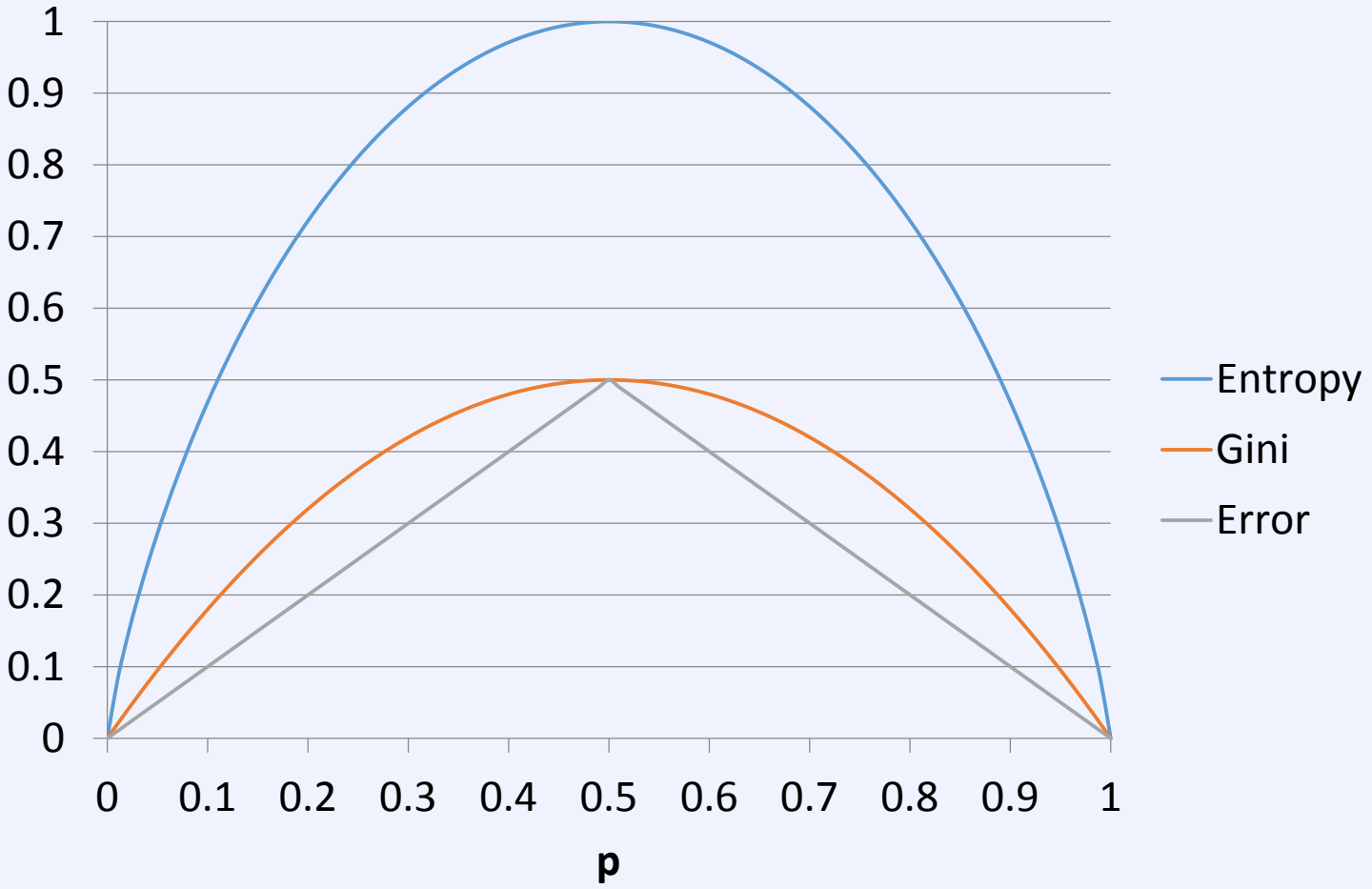
$0 \times \log_2(0) = 0$

$$Entropy(t) = -\sum_{c_i \in C} p(c_i \mid t) \log_2 p(c_i \mid t) \leq 0$$

$$Gini(t) = 1 - \sum_{c_i \in C} \left(p(c_i \mid t)\right)^2$$

$$Classification\ error(t) = 1 - \max_{c_i \in C}\{p(c_i \mid t)\}$$

Comparing impurity measures



(for binary classification, with p the probability for class 1, and $(1-p)$ the probability for class 2)

Comparing conditions

The quality of the split: the change in **impurity**

- called the **gain** of the test condition

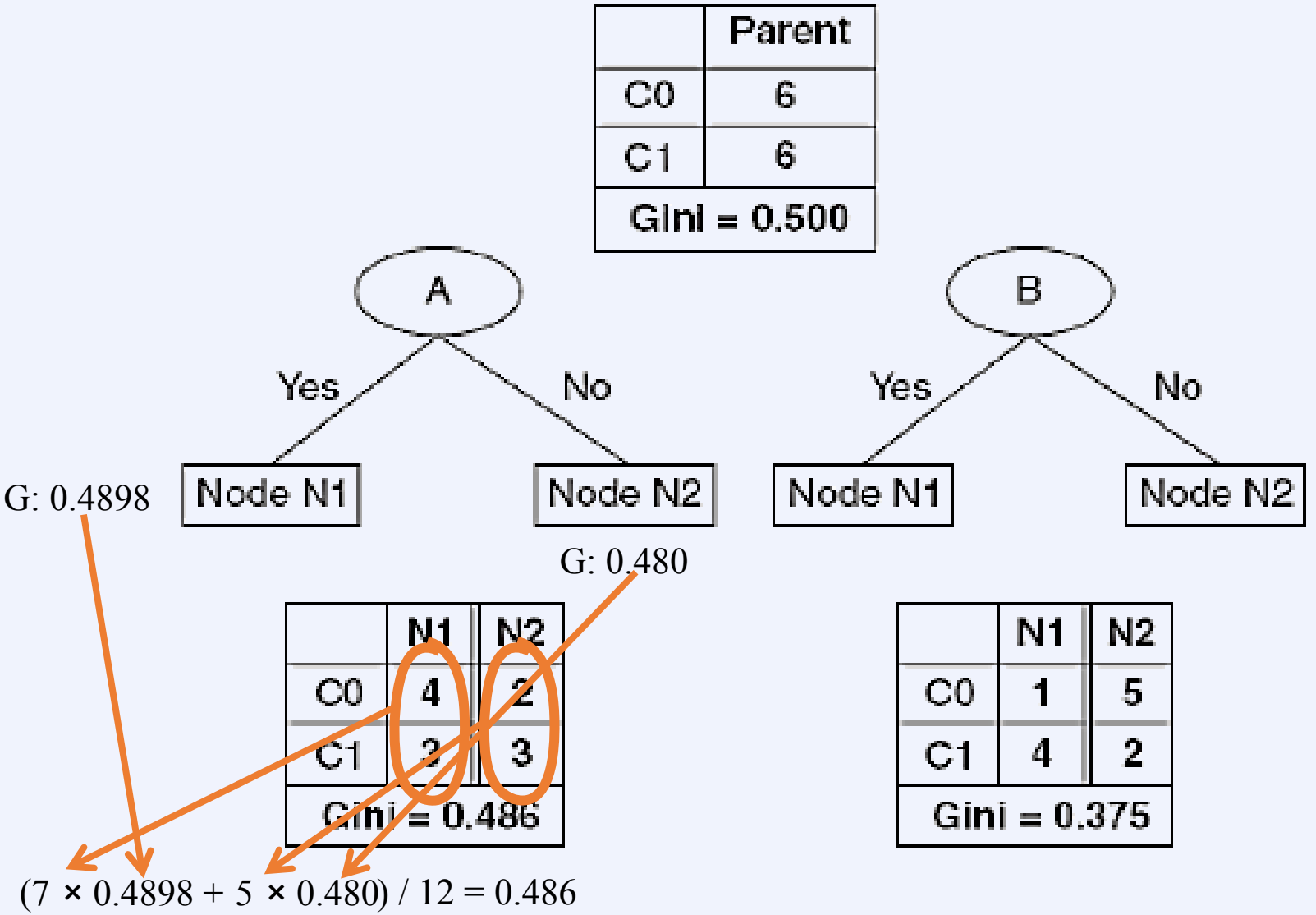
$$\Delta = I(p) - \sum_{j}^{k} \frac{N(v_j)}{N} I(v_j)$$

- $I(\cdot)$ is the impurity measure
- k is the number of attribute values
- p is the parent node, v_j is the child node
- N is the total number of records at the parent node
- $N(v_j)$ is the number of records associated with the child node

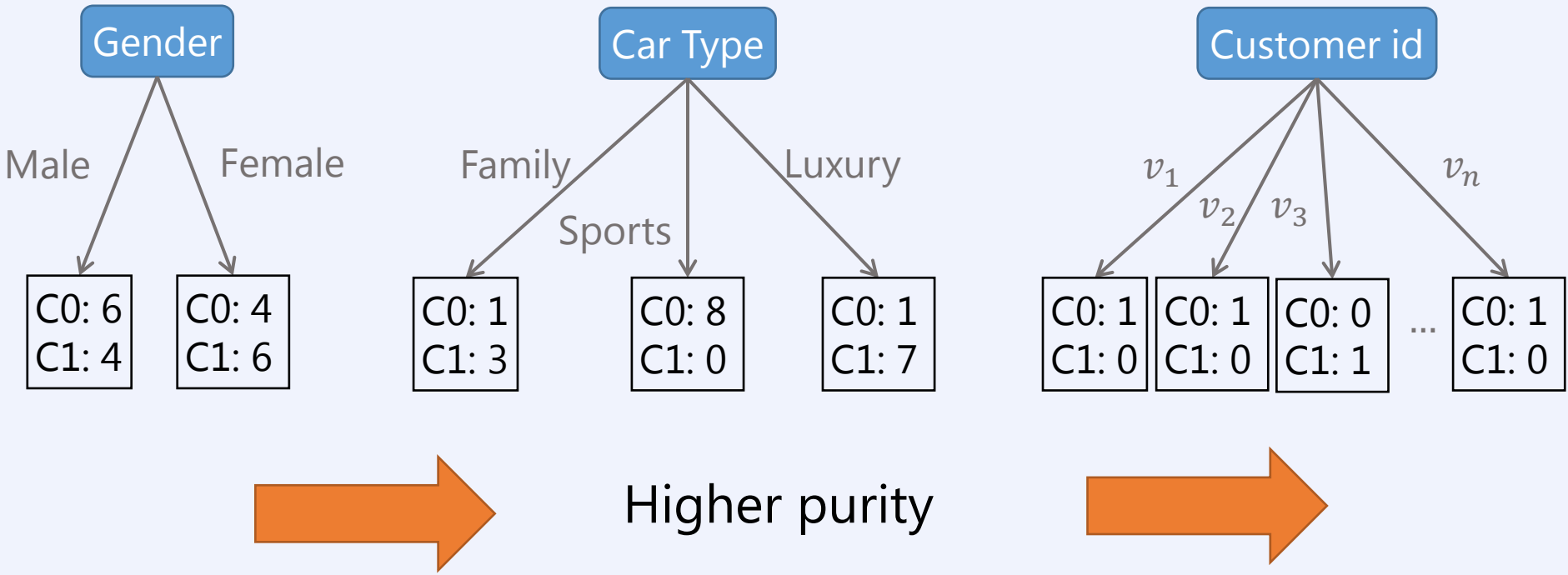
Maximizing the gain ↔ minimising the weighted average impurity measure of child nodes

If $I(\cdot) = entropy(\cdot)$, then $\Delta = \Delta_{info}$ is called **information gain**

Example: computing gain



Problem of maximising Δ



Stopping splitting

Stop expanding when all records belong to the same class

Stop expanding when all records have similar attribute values

Early termination

- e.g. gain ratio drops below certain threshold
- keeps trees simple
- helps with overfitting

Problems of maximising Δ

Impurity measures favor attributes with many values

Test conditions with many outcomes may not be desirable

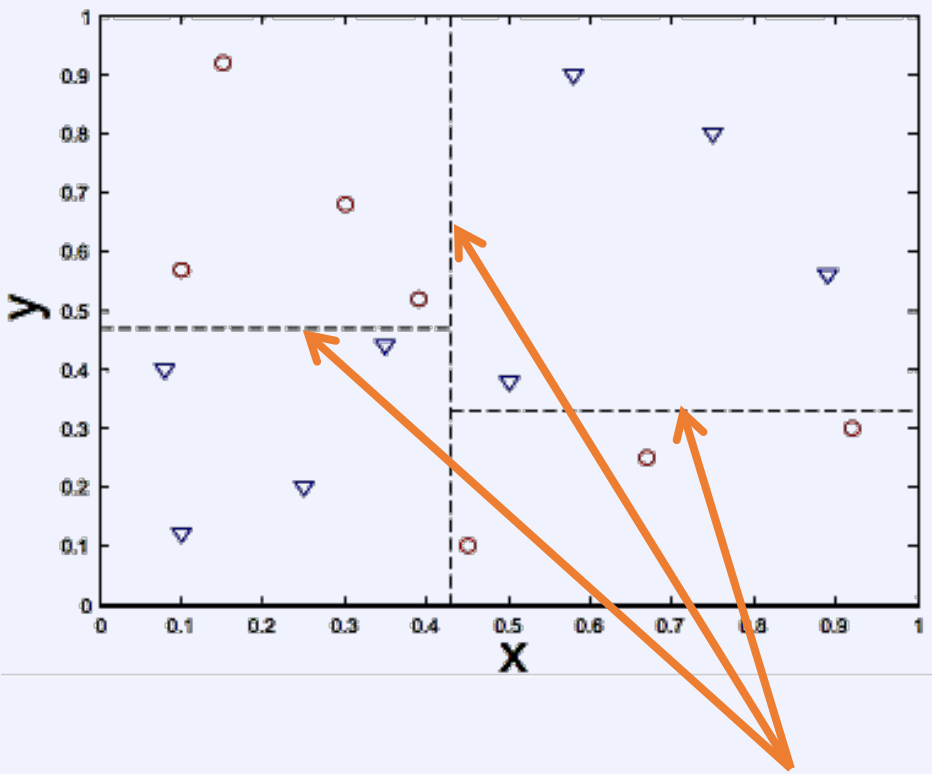
- number of records in each partition is too small to make predictions

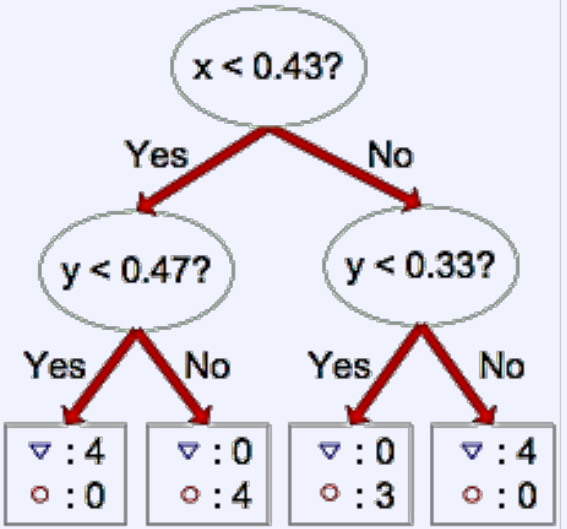
Solution 1: **gain ratio**

- $gain\ ratio = \frac{\Delta_{info}}{SplitInfo}$ $\quad SplitInfo = -\sum_{i=1}^{k} P(v_i) \log_2\big(P(v_i)\big)$
- $P(v_i)$ is the fraction of records at child; k = total number of splits
- used e.g. in C4.5

Solution 2: restrict the splits to binary

Geometry of single-attribute splits



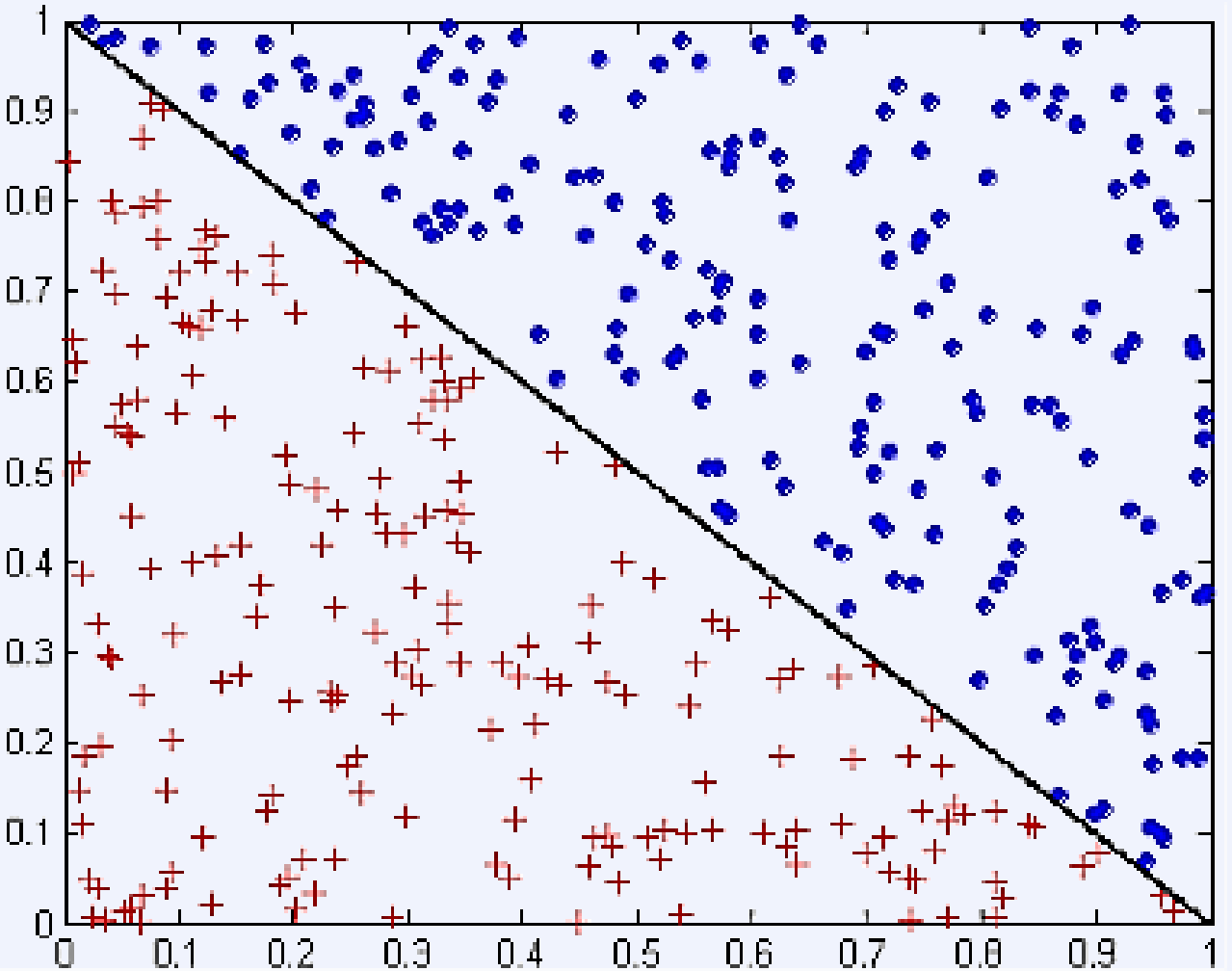


Decision boundaries are always axis-parallel for single-attribute splits

Geometry of single-attribute splits

Seems easy to classify, but…

How to split?



Combatting overfitting

Overfitting is a major problem with all classifiers

As decision trees are parameter-free, we need to stop building the tree before overfitting happens

- overfitting makes decision trees overly complex
- generalization error will be big

In practice, to prevent overfitting, we use

- test/train data
- perform cross-validation
- model selection (e.g. MDL)
- or simply choose a minimal-number of records per leaf

Handling overfitting

In **pre-pruning** we stop building the decision tree when a stopping criterion is satisfied

In **post-pruning** we trim a full-grown decision tree

- from bottom to up try replacing a decision node with a leaf
- if generalization error improves, replace the sub-tree with a leaf
- new leaf node's class label is the majority of the sub-tree

Summary of decision trees

Fast to build

Extremely fast to use

- small ones are easy to interpret
  - good for domain expert's verification
  - used e.g. in medicine

Redundant attributes are not (much of) a problem

Single-attribute splits cause axis-parallel decision boundaries

Requires post-pruning to avoid overfitting

Chapter 6.4:
Probabilistic classifiers

Aggarwal Ch. 10.5

Basic idea

Recall Bayes' theorem

$$\Pr[Y \mid X] = \frac{\Pr[X \mid Y]\Pr[Y]}{\Pr[X]}$$

In classification

- random variable X is the attribute set
- random variable Y is the class variable
- Y depends on X in a non-deterministic way (assumption)

The dependency between X and Y is captured by $\Pr[Y \mid X]$ and $\Pr[Y]$

- the **posterior** and **prior** probability

Building a classifier

Training phase

- learn the posterior probabilities $\Pr[Y \mid X]$ for every combination of X and Y based on training data

Test phase

- for a test record X', we compute the class Y' that maximizes the posterior probability $\Pr[Y' \mid X']$

$$Y' = \arg\max_j \{\Pr[c_j \mid X']\} = \arg\max_j \left\{\frac{\Pr[X' \mid c_j]\Pr[c_j]}{\Pr[X']}\right\} = \arg\max_j \{\Pr[X' \mid c_j]\Pr[c_j]\}$$

So, we need $\Pr[X' \mid c_j]$ and $\Pr[c_j]$

- $\Pr[c_j]$ is easy, it's the fraction of test records that belong to class c_j
- $\Pr[X' \mid c_j]$, however…

Computing the probabilities

Assume that the attributes are **conditionally independent** given the class label – the classifier is **naïve**

$$\Pr[X \mid Y = c_j] = \prod_{i=1}^{d} \Pr[X_i \mid Y = c_j]$$

- where X_i is the i-th attribute

Without independency there would be too many variables to estimate, with independency, it is enough to estimate $\Pr[X_i \mid Y]$

$$\Pr[Y \mid X] = \Pr[Y] \prod_{i=1}^{d} \Pr[X_i \mid Y] / \Pr[X]$$

- $\Pr[X]$ is fixed, so can be omitted

But how do we estimate the **likelihood** $\Pr[X_i \mid Y]$?

Categorical attributes

If X_i is categorical $\Pr[X_i = x_i \mid Y = c]$ is simply the **fraction** of training instances in class c that take value x_i on the i-th attribute

$$\Pr[\, HomeOwner = yes \mid No \,] = \frac{3}{7}$$

$$\Pr[\, MaritalStatus = S \mid Yes \,] = \frac{2}{3}$$

TID	Home Owner	Marital Status	Annual Income	Defaulted Borrower
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

Continuous attributes: discretisation

We can discretise continuous attributes to intervals

- these intervals act like ordinal attributes (because they are)

The problem is **how** to discretize

- too many intervals:
  too few training records per interval → unreliable estimates
- too few intervals:
  intervals merge ranges correlated to different classes,
  making distinguishing the classes more difficult (impossible)

Continuous attributes, continued

Alternatively we assume a distribution

- normally we assume a normal distribution

We need to estimate the distribution parameters

- for normal distribution, we use sample mean and sample variance
- for estimation, we consider the values of attribute X_i that are associated with class c_j in the test data

We hope that the parameters for distributions are different for different classes of the same attribute

- why?

Example – Naïve Bayes

TID	Home Owner	Marital Status	Annual Income	Defaulted Borrower
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

Annual income

Class = No
- sample mean = 110
- sample variance = 2975

Class = Yes
- sample mean = 90
- sample variance = 25

Test data: $X = (HO = No, MS = M, AI = €120K)$

$\Pr[Yes] = 0.3, \quad \Pr[No] = 0.7$

$$\Pr[X \mid No] = \Pr[HO = No \mid No] \times \Pr[MS = M \mid No] \times \Pr[AI = €120K \mid No]$$
$$= \frac{4}{7} \times \frac{4}{7} \times 0.0072 = 0.0024$$

$$\Pr[X \mid Yes] = \Pr[HO = No \mid Yes] \times \Pr[MS = M \mid Yes] \times \Pr[AI = €120K \mid Yes]$$
$$= 1 \times 0 \times \epsilon = 0$$

$\Pr[No \mid X] = \alpha \times \Pr[No] \times \Pr[X \mid No] = \alpha \times 0.7 \times 0.0024 = 0.0016\alpha, \qquad \alpha = 1/\Pr[X]$

→ $\Pr[No \mid X]$ has higher posterior and X should hence be classified as **non-defaulter**

Continuous distributions at fixed point

If X_i is continuous, $\Pr[\, X_i = x_i \mid Y = c_j \,] = 0$!

- but we still need to estimate that number…

Self-cancelling trick

$$\Pr[\, x_i - \epsilon \leq X_i \leq x_i + \epsilon \mid Y = c_j \,]$$
$$= \int_{x_i-\epsilon}^{x_i+\epsilon} (2\pi\sigma_{ij})^{-\frac{1}{2}} \exp\left(-\frac{(x-\mu_{ij})^2}{2\sigma_{ij}^2}\right)$$
$$\approx 2\epsilon f(x_i; \mu_{ij}, \sigma_{ij})$$

- but 2ϵ cancels out in the normalization constant…

Zero likelihood

We might have no samples with $X_i = x_i$ and $Y = c_j$

- naturally only a problem for categorical variables
- $\Pr[X_i = x_i \mid Y = c_j] = 0 \rightarrow$ zero posterior probability
- it can be that **all** classes have zero posterior probability for some data

Answer is smoothing (m-**estimate**):

$$\Pr[X_i = x_i \mid Y = c_j] = \frac{n_i + mp}{n + m}$$

- n = # of training instances from class c_j
- n_i = # training instances from c_j that take value x_i
- m = "equivalent sample size"
- p = user-set parameter

More on $\Pr[X_i = x_i \mid Y = c_j] = \frac{n_i + mp}{n+m}$

The parameters are p and m

- if $n = 0$, then likelihood is p
  - p is "prior" of observing x_i in class c_j
- parameter m governs the trade-off between p and observed probability n_i/n

Setting these parameters is again problematic…

Alternatively, we just add one pseudo-count to each class

- $\Pr[X_i = x_i \mid Y = c_j] = (n_i + 1) / (n + |dom(X_i)|)$
- $|dom(X_i)|$ = # values attribute X_i can take

Summary for Naïve Bayes

Robust to isolated noise

- it's averaged out

Can handle missing values

- example is ignored when building the model, and attribute is ignored when classifying new data

Robust to irrelevant attributes

- $\Pr(X_i \mid Y)$ is (almost) uniform for irrelevant X_i

Can have issues with correlated attributes

Chapter 6.5:
Many many more classifiers

Aggarwal Ch. 10.6, 11

It's a jungle out there

There is **no free lunch**

- there is no single best classifier for every problem setting
- there exist more classifiers than you can shake a stick at

Nice theory exists on the power of classes of classifiers

- support vector machines (kernel methods) can do anything
- so can artificial neural networks

Two heads know more than 1, and n-heads know more than 2

- if you're interested look into bagging and boosting
- ensemble methods combine multiple 'weak' classifiers into one big strong team

It's about insight

Most classifiers focus purely on prediction accuracy

- in data mining we care mostly about interpretability

The classifiers we have seen today work very well in practice, and are interpretable

- so are rule-based classifiers

Support vector machines, neural networks, and ensembles give good predictive performance, but are black boxes.

Conclusions

Classification is one of the most important and most used data analysis methods – **predictive analytics**

There exist many different types of classification

- we've seen instance-based, decision trees, and naïve Bayes
- these are (relatively) interpretable, and work well in practice,

There is no single best classifier

- if you're mainly interested in performance → go take Machine Learning
- if you're interested in the why, in explainability, stay here.

Thank you!

Classification is one of the most important and most used data analysis methods – **predictive analytics**

There exist many different types of classification

- we've seen instance-based, decision trees, and naïve Bayes
- these are (relatively) interpretable, and work well in practice,

There is no single best classifier

- if you're mainly interested in performance → go take Machine Learning
- if you're interested in the why, in explainability, stay here.