Machine Learning, Data Mining, and Knowledge Discovery: An Introduction

Lesson Outline

- **Introduction: Data Flood**
- Data Mining Application Examples
- Data Mining & Knowledge Discovery
- Data Mining Tasks

Trends leading to Data Flood

- More data is generated:
  - Bank, telecom, other business transactions ...
  - Scientific data: astronomy, biology, etc
  - Web, text, and e-commerce

Big Data Examples

- Europe's Very Long Baseline Interferometry (VLBI) has 16 telescopes, each of which produces **1 Gigabit/second** of astronomical data over a 25-day observation session
  - storage and analysis a big problem
- AT&T handles billions of calls per day
  - so much data, it cannot be all stored -- analysis has to be done "on the fly", on streaming data

Largest databases in 2003

- Commercial databases:
  - Winter Corp. 2003 Survey: France Telecom has largest decision-support DB, ~30TB; AT&T ~ 26 TB
- Web
  - Alexa internet archive: 7 years of data, 500 TB
  - Google searches 4+ Billion pages, many hundreds TB
  - IBM WebFountain, 160 TB (2003)
  - Internet Archive (www.archive.org),~ 300 TB

5 million terabytes created in 2002

- UC Berkeley 2003 estimate: 5 exabytes (5 million terabytes) of new data was created in 2002.

www.sims.berkeley.edu/research/projects/how-much-info-2003/

- US produces ~40% of new stored data worldwide

Data Growth Rate

- Twice as much information was created in 2002 as in 1999 (~30% growth rate)
- Other growth rate estimates even higher
- Very little data will ever be looked at by a human
- Knowledge Discovery is **NEEDED** to make sense and use of data.

Lesson Outline

- Introduction: Data Flood
- **Data Mining Application Examples**
- Data Mining & Knowledge Discovery
- Data Mining Tasks

Machine Learning / Data Mining Application areas

- Science
  - astronomy, bioinformatics, drug discovery, …
- Business
  - advertising, CRM (Customer Relationship management), investments, manufacturing, sports/entertainment, telecom, e-Commerce, targeted marketing, health care, …
- Web:
  - search engines, bots, …
- Government
  - law enforcement, profiling tax cheaters, anti-terror(?)

Data Mining for Customer Modeling

- Customer Tasks:
  - attrition prediction
  - targeted marketing:
    - cross-sell, customer acquisition
  - credit-risk
  - fraud detection
- Industries
  - banking, telecom, retail sales, …

Customer Attrition: Case Study

- Situation: Attrition rate at for mobile phone customers is around 25-30% a year!

Task:

- Given customer information for the past N months, predict who is likely to attrite next month.
- Also, estimate customer value and what is the cost-effective offer to be made to this customer.

Customer Attrition Results

- Verizon Wireless built a customer data warehouse
- Identified potential attriters
- Developed multiple, regional models
- Targeted customers with high propensity to accept the offer
- Reduced attrition rate from over 2%/month to under 1.5%/month (huge impact, with >30 M subscribers)

(Reported in 2003)

Assessing Credit Risk: Case Study

- Situation: Person applies for a loan
- Task: Should a bank approve the loan?
- Note: People who have the best credit don't need the loans, and people with worst credit are not likely to repay. Bank's best customers are in the middle

Credit Risk - Results

- Banks develop credit models using variety of machine learning methods.
- Mortgage and credit card proliferation are the results of being able to successfully predict if a person is likely to default on a loan
- Widely deployed in many countries

Successful e-commerce – Case Study

- A person buys a book (product) at Amazon.com.
- Task: Recommend other books (products) this person is likely to buy
- Amazon does clustering based on books bought:
  - customers who bought "**Advances in Knowledge Discovery and Data Mining**", also bought "**Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations**"
- Recommendation program is quite successful

Problems Suitable for Data-Mining

- require knowledge-based decisions
- have a changing environment
- have sub-optimal current methods
- have accessible, sufficient, and relevant data
- provides high payoff for the right decisions!

Privacy considerations important if personal data is involved

Lesson Outline

- Introduction: Data Flood
- Data Mining Application Examples
- **Data Mining & Knowledge Discovery**
- Data Mining Tasks

Knowledge Discovery Definition

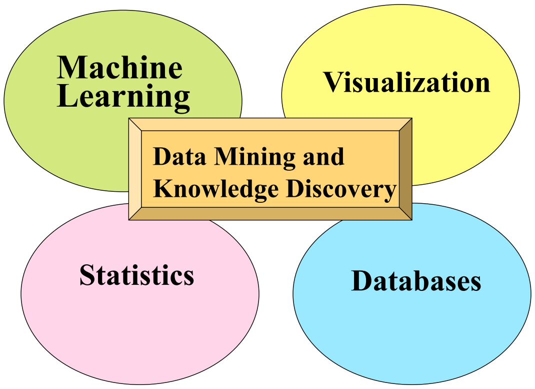
Knowledge Discovery in Data is the

non-trivial process of identifying

- *valid*
- *novel*
- potentially *useful*
- and ultimately *understandable patterns* in data.

from *Advances in Knowledge Discovery and Data Mining,* Fayyad, Piatetsky-Shapiro, Smyth, and Uthurusamy, (Chapter 1), AAAI/MIT Press 1996

Related Fields

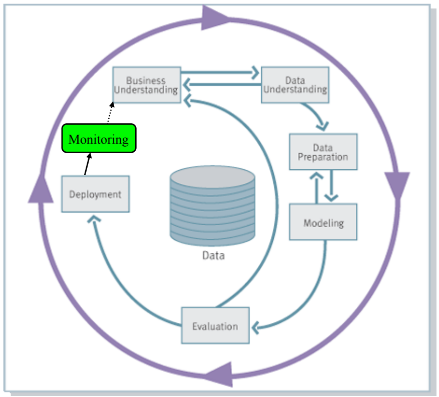


Statistics, Machine Learning and Data Mining

- Statistics:
  - more theory-based
  - more focused on testing hypotheses
- Machine learning
  - more heuristic
  - focused on improving performance of a learning agent
  - also looks at real-time learning and robotics – areas not part of data mining
- Data Mining and Knowledge Discovery
  - integrates theory and heuristics
  - focus on the entire process of knowledge discovery, including data cleaning, learning, and integration and visualization of results
- Distinctions are fuzzy

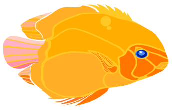
witten&eibe

Knowledge Discovery Process flow, according to CRISP-DM



see www.crisp-dm.org for more information

Historical Note: Many Names of Data Mining



- Data Fishing, Data Dredging: 1960-
  - used by Statistician (as bad name)
- Data Mining :1990 --
  - used DB, business
  - in 2003 – bad image because of TIA
- Knowledge Discovery in Databases (1989-)
  - used by AI, Machine Learning Community
- also Data Archaeology, Information Harvesting, Information Discovery, Knowledge Extraction, ...

Currently: Data Mining and Knowledge Discovery are used interchangeably

Lesson Outline

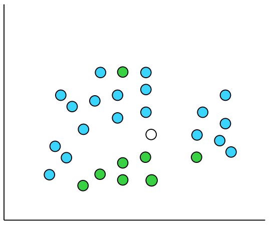
- Introduction: Data Flood
- Data Mining Application Examples
- Data Mining & Knowledge Discovery
- **Data Mining Tasks**

Major Data Mining Tasks

- **Classification:** predicting an item class
- **Clustering:** finding clusters in data
- **Associations:** e.g. A & B & C occur frequently
- **Visualization:** to facilitate human discovery
- **Summarization:** describing a group
- **Deviation Detection**: finding changes
- Estimation: predicting a continuous value
- Link Analysis: finding relationships
- ...

Data Mining Tasks: Classification

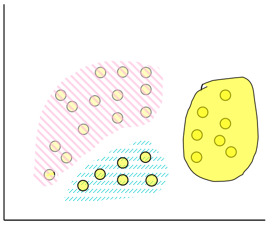
Learn a method for predicting the instance class from pre-labeled (classified) instances



Many approaches:
Statistics,
Decision Trees,
Neural Networks,
...

Data Mining Tasks: Clustering

Find "natural" grouping of instances given un-labeled data



Summary:

- Technology trends lead to data flood
  - data mining is needed to make sense of data
- Data Mining has many applications, successful and not
- Knowledge Discovery Process
- Data Mining Tasks
  - classification, clustering, …

More on Data Mining and Knowledge Discovery

KDnuggets.com

- News, Publications
- Software, Solutions
- Courses, Meetings, Education
- Publications, Websites, Datasets
- Companies, Jobs
- …

Machine Learning: finding patterns

Outline

- **Machine learning and Classification**
- Examples
- *Learning as Search
- Bias
- Weka

Finding patterns

- Goal: programs that detect patterns and regularities in the data
- Strong patterns ⇒ good predictions
  - Problem 1: most patterns are not interesting
  - Problem 2: patterns may be inexact (or spurious)
  - Problem 3: data may be garbled or missing

Machine learning techniques

- *Algorithms for acquiring structural descriptions from examples*
- Structural descriptions represent patterns explicitly
  - Can be used to predict outcome in new situation
  - Can be used to understand and explain how prediction is derived (*may be even more important*)
- Methods originate from artificial intelligence, statistics, and research on databases

Can machines really learn?

- Definitions of "learning" from dictionary:

To get knowledge of by study, experience, or being taught } Difficult to measure

To become aware by information or from observation
To commit to memory
To be informed of, ascertain; to receive instruction } Trivial for computers

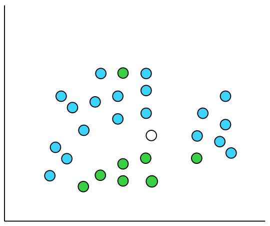
- Operational definition:

Things learn when they change their behavior in a way that makes them perform better in the future. } Does a slipper learn?

- Does learning imply intention?

Classification

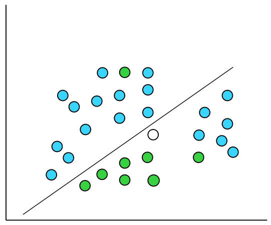
Learn a method for predicting the instance class from pre-labeled (classified) instances



Many approaches:
Regression,
Decision Trees,
Bayesian,
Neural Networks,
...

Given a set of points from classes 🔵 🟢
what is the class of new point ○?

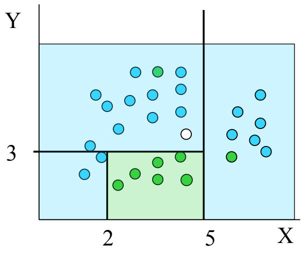
Classification: Linear Regression



- Linear Regression

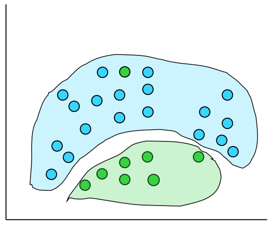
  $w_0 + w_1 x + w_2 y >= 0$
- Regression computes w_i from data to minimize squared error to 'fit' the data
- Not flexible enough

Classification: Decision Trees



if X > 5 then blue
else if Y > 3 then blue
else if X > 2 then green
else blue

Classification: Neural Nets



- Can select more complex regions
- Can be more accurate
- Also can overfit the data – find patterns in random noise

Outline

- Machine learning and Classification
- **Examples**
- *Learning as Search
- Bias
- Weka

The weather problem

Outlook	Temperature	Humidity	Windy	Play
sunny	85	85	false	no
sunny	80	90	true	no
overcast	83	86	false	yes
rainy	70	96	false	yes
rainy	68	80	false	yes
rainy	65	70	true	no
overcast	64	65	true	yes
sunny	72	95	false	no
sunny	69	70	false	yes
rainy	75	80	false	yes
sunny	75	70	true	yes
overcast	72	90	true	yes
overcast	81	75	false	yes
rainy	71	91	true	no

Given past data, Can you come up with the rules for Play/Not Play ?

What is the game?

The weather problem

- Conditions for playing golf

Outlook	Temperature	Humidity	Windy	Play
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Overcast	Hot	High	False	Yes
Rainy	Mild	Normal	False	Yes
…	…	…	…	…

```
If outlook = sunny and humidity = high then play = no
If outlook = rainy and windy = true then play = no
If outlook = overcast then play = yes
If humidity = normal then play = yes
If none of the above then play = yes
```

witten&eibe

Weather data with mixed attributes

- Some attributes have numeric values

Outlook	Temperature	Humidity	Windy	Play
Sunny	85	85	False	No
Sunny	80	90	True	No
Overcast	83	86	False	Yes
Rainy	75	80	False	Yes
…	…	…	…	…

```
If outlook = sunny and humidity > 83 then play = no
If outlook = rainy and windy = true then play = no
If outlook = overcast then play = yes
If humidity < 85 then play = yes
If none of the above then play = yes
```

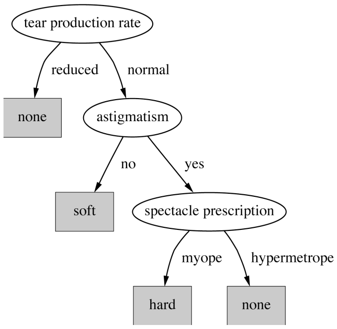
The contact lenses data

Age	Spectacle prescription	Astigmatism	Tear production rate	Recommended lenses
Young	Myope	No	Reduced	None
Young	Myope	No	Normal	Soft
Young	Myope	Yes	Reduced	None
Young	Myope	Yes	Normal	Hard
Young	Hypermetrope	No	Reduced	None
Young	Hypermetrope	No	Normal	Soft
Young	Hypermetrope	Yes	Reduced	None
Young	Hypermetrope	Yes	Normal	hard
Pre-presbyopic	Myope	No	Reduced	None
Pre-presbyopic	Myope	No	Normal	Soft
Pre-presbyopic	Myope	Yes	Reduced	None
Pre-presbyopic	Myope	Yes	Normal	Hard
Pre-presbyopic	Hypermetrope	No	Reduced	None
Pre-presbyopic	Hypermetrope	No	Normal	Soft
Pre-presbyopic	Hypermetrope	Yes	Reduced	None
Pre-presbyopic	Hypermetrope	Yes	Normal	None
Presbyopic	Myope	No	Reduced	None
Presbyopic	Myope	No	Normal	None
Presbyopic	Myope	Yes	Reduced	None
Presbyopic	Myope	Yes	Normal	Hard
Presbyopic	Hypermetrope	No	Reduced	None
Presbyopic	Hypermetrope	No	Normal	Soft
Presbyopic	Hypermetrope	Yes	Reduced	None
Presbyopic	Hypermetrope	Yes	Normal	None

A complete and correct rule set

```
If tear production rate = reduced then recommendation = none

If age = young and astigmatic = no
    and tear production rate = normal then recommendation = soft

If age = pre-presbyopic and astigmatic = no
    and tear production rate = normal then recommendation = soft

If age = presbyopic and spectacle prescription = myope
    and astigmatic = no  then recommendation = none

If spectacle prescription = hypermetrope and astigmatic = no
    and tear production rate = normal then recommendation = soft

If spectacle prescription = myope and astigmatic = yes
    and tear production rate = normal then recommendation = hard

If age young and astigmatic = yes
    and tear production rate = normal then recommendation = hard

If age = pre-presbyopic
    and spectacle prescription = hypermetrope
    and astigmatic = yes then recommendation = none

If age = presbyopic and spectacle prescription = hypermetrope
    and astigmatic = yes then recommendation = none
```

A decision tree for this problem



Classifying iris flowers

	Sepal length	Sepal width	Petal length	Petal width	Type
1	5.1	3.5	1.4	0.2	Iris setosa
2	4.9	3.0	1.4	0.2	Iris setosa
…					
51	7.0	3.2	4.7	1.4	Iris versicolor
52	6.4	3.2	4.5	1.5	Iris versicolor
…					
101	6.3	3.3	6.0	2.5	Iris virginica
102	5.8	2.7	5.1	1.9	Iris virginica
…					

```
If petal length < 2.45 then Iris setosa
If sepal width < 2.10 then Iris versicolor
...
```

Predicting CPU performance

- Example: 209 different computer configurations

	Cycle time (ns)	Main memory (Kb)		Cache (Kb)	Channels		Performance
	MYCT	MMIN	MMAX	CACH	CHMIN	CHMAX	PRP
1	125	256	6000	256	16	128	198
2	29	8000	32000	32	8	32	269
…							
208	480	512	8000	32	0	0	67
209	480	1000	4000	0	0	0	45

- Linear regression function

```
PRP =  -55.9 + 0.0489 MYCT + 0.0153 MMIN + 0.0056 MMAX
       + 0.6410 CACH - 0.2700 CHMIN + 1.480 CHMAX
```

Soybean classification

	Attribute	Number of values	Sample value
Environment	Time of occurrence	7	July
	Precipitation	3	Above normal
…			
Seed	Condition	2	Normal
	Mold growth	2	Absent
…			
Fruit	Condition of fruit pods	4	Normal
	Fruit spots	5	?
Leaves	Condition	2	Abnormal
	Leaf spot size	3	?
…			
Stem	Condition	2	Abnormal
	Stem lodging	2	Yes
…			
Roots	Condition	3	Normal
Diagnosis		19	Diaporthe stem canker

The role of domain knowledge

```
If leaf condition is normal
   and stem condition is abnormal
   and stem cankers is below soil line
   and canker lesion color is brown
then
   diagnosis is rhizoctonia root rot
```

```
If leaf malformation is absent
   and stem condition is abnormal
   and stem cankers is below soil line
   and canker lesion color is brown
then
   diagnosis is rhizoctonia root rot
```

But in this domain, “leaf condition is normal” implies “leaf malformation is absent”!

Outline

- Machine learning and Classification
- Examples
- ***Learning as Search**
- Bias
- Weka

Learning as search

- Inductive learning: find a concept description that fits the data
- Example: rule sets as description language
  - Enormous, but finite, search space
- Simple solution:
  - enumerate the concept space
  - eliminate descriptions that do not fit examples
  - surviving descriptions contain target concept

Enumerating the concept space

- Search space for weather problem
  - 4 x 4 x 3 x 3 x 2 = 288 possible combinations
  - With 14 rules ⇒ 2.7x10^{34} possible rule sets
- Solution: candidate-elimination algorithm
- Other practical problems:
  - More than one description may survive
  - No description may survive
    - Language is unable to describe target concept
    - *or* data contains noise

The version space

- Space of consistent concept descriptions
- Completely determined by two sets
  - *L*: most specific descriptions that cover all positive examples and no negative ones
  - *G*: most general descriptions that do not cover any negative examples and all positive ones
- Only *L* and *G* need be maintained and updated
- But: still computationally very expensive
- And: does not solve other practical problems

*Version space example

- Given: red or green cows or chicken

L={} *G*={<*, *>}

<green,cow>: positive

L={<green, cow>} *G*={<*, *>}

<red,chicken>: negative

L={<green, cow>}
G={<green,*>,<*,cow>}

<green, chicken>: positive

L={<green, *>} *G*={<green, *>}

*Candidate-elimination algorithm

```
Initialize L and G
For each example e:
  If e is positive:
   Delete all elements from G that do not cover e
   For each element r in L that does not cover e:
     Replace r by all of its most specific generalizations
       that    1. cover e and
         2. are more specific than some element in G
   Remove elements from L that
     are more general than some other element in L
  If e is negative:
   Delete all elements from L that cover e
   For each element r in G that covers e:
     Replace r by all of its most general specializations
       that    1. do not cover e and
         2. are more general than some element in L
   Remove elements from G that
     are more specific than some other element in G
```

Outline

- Machine learning and Classification
- Examples
- *Learning as Search
- **Bias**
- Weka

Bias

- Important decisions in learning systems:
  - Concept description language
  - Order in which the space is searched
  - Way that overfitting to the particular training data is avoided
- These form the "bias" of the search:
  - Language bias
  - Search bias
  - Overfitting-avoidance bias

Language bias

- Important question:
  - is language universal
    or does it restrict what can be learned?
- Universal language can express arbitrary subsets of examples
- If language includes logical *or* ("disjunction"), it is universal
- Example: rule sets
- Domain knowledge can be used to exclude some concept descriptions *a priori* from the search

Search bias

- Search heuristic
  - "Greedy" search: performing the best single step
  - "Beam search": keeping several alternatives
  - …
- Direction of search
  - *General-to-specific*
    - E.g. specializing a rule by adding conditions
  - *Specific-to-general*
    - E.g. generalizing an individual instance into a rule

Overfitting-avoidance bias

- Can be seen as a form of search bias
- Modified evaluation criterion
  - E.g. balancing simplicity and number of errors
- Modified search strategy
  - E.g. pruning (simplifying a description)
    - Pre-pruning: stops at a simple description before search proceeds to an overly complex one
    - Post-pruning: generates a complex description first and simplifies it afterwards

Weka

Waikato Environment for Knowledge Analysis

(c) 1999 - 2003
University of Waikato
New Zealand

Input:
Concepts, Attributes, Instances

Module Outline

- Terminology
- What's a concept?
  - Classification, association, clustering, numeric prediction
- What's in an example?
  - Relations, flat files, recursion
- What's in an attribute?
  - Nominal, ordinal, interval, ratio
- Preparing the input
  - ARFF, attributes, missing values, getting to know data

Terminology

- Components of the input:
  - Concepts: kinds of things that can be learned
    - Aim: intelligible and operational concept description
  - Instances: the individual, independent examples of a concept
    - Note: more complicated forms of input are possible
  - Attributes: measuring aspects of an instance
    - We will focus on nominal and numeric ones

What's a concept?

- Data Mining Tasks (Styles of learning):
  - Classification learning:
    predicting a discrete class
  - Association learning:
    detecting associations between features
  - Clustering:
    grouping similar instances into clusters
  - Numeric prediction:
    predicting a numeric quantity
- Concept: thing to be learned
- Concept description: output of learning scheme

Classification learning

- Example problems: attrition prediction, using DNA data for diagnosis, weather data to predict play/not play
- Classification learning is supervised
  - Scheme is being provided with actual outcome
- Outcome is called the *class* of the example
- Success can be measured on fresh data for which class labels are known (test data)
- In practice success is often measured subjectively

Association learning

- Examples: supermarket basket analysis -what items are bought together (e.g. milk+cereal, chips+salsa)
- Can be applied if no class is specified and any kind of structure is considered "interesting"
- Difference with classification learning:
  - Can predict any attribute's value, not just the class, and more than one attribute's value at a time
  - Hence: far more association rules than classification rules
  - Thus: constraints are necessary
    - Minimum coverage and minimum accuracy

Clustering

- Examples: customer grouping
- Finding groups of items that are similar
- Clustering is *unsupervised*
  - The class of an example is not known
- Success often measured subjectively

	Sepal length	Sepal width	Petal length	Petal width	Type
1	5.1	3.5	1.4	0.2	~~Iris setosa~~
2	4.9	3.0	1.4	0.2	~~Iris setosa~~
…					
51	7.0	3.2	4.7	1.4	~~Iris versicolor~~
52	6.4	3.2	4.5	1.5	~~Iris versicolor~~
…					
101	6.3	3.3	6.0	2.5	~~Iris virginica~~
102	5.8	2.7	5.1	1.9	~~Iris virginica~~
…					

Numeric prediction

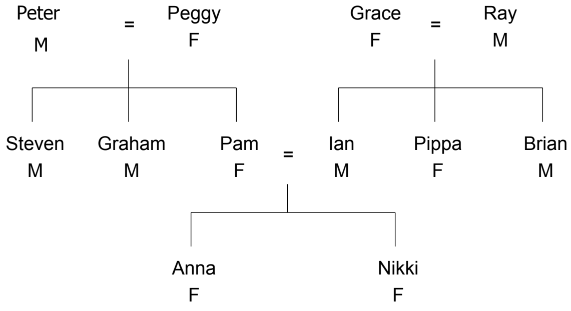
- Classification learning, but "class" is numeric
- Learning is supervised
  - Scheme is being provided with target value
- Measure success on test data

Outlook	Temperature	Humidity	Windy	Play-time
Sunny	Hot	High	False	5
Sunny	Hot	High	True	0
Overcast	Hot	High	False	55
Rainy	Mild	Normal	False	40
…	…	…	…	…

What's in an example?

- Instance: specific type of example
  - Thing to be classified, associated, or clustered
  - Individual, independent example of target concept
  - Characterized by a predetermined set of attributes
- Input to learning scheme: set of instances/dataset
  - Represented as a single relation/flat file
- Rather restricted form of input
  - No relationships between objects
- Most common form in practical data mining

A family tree



Family tree represented as a table

Name	Gender	Parent1	parent2
Peter	Male	?	?
Peggy	Female	?	?
Steven	Male	Peter	Peggy
Graham	Male	Peter	Peggy
Pam	Female	Peter	Peggy
Ian	Male	Grace	Ray
Pippa	Female	Grace	Ray
Brian	Male	Grace	Ray
Anna	Female	Pam	Ian
Nikki	Female	Pam	Ian

The "sister-of" relation

First person	Second person	Sister of?
Peter	Peggy	No
Peter	Steven	No
…	…	…
Steven	Peter	No
Steven	Graham	No
Steven	Pam	Yes
…	…	…
Ian	Pippa	Yes
…	…	…
Anna	Nikki	Yes
…	…	…
Nikki	Anna	yes

First person	Second person	Sister of?
Steven	Pam	Yes
Graham	Pam	Yes
Ian	Pippa	Yes
Brian	Pippa	Yes
Anna	Nikki	Yes
Nikki	Anna	Yes
All the rest		No

Closed-world assumption

A full representation in one table

First person				Second person				Sister of?
Name	Gender	Parent1	Parent2	Name	Gender	Parent1	Parent2	
Steven	Male	Peter	Peggy	Pam	Female	Peter	Peggy	Yes
Graham	Male	Peter	Peggy	Pam	Female	Peter	Peggy	Yes
Ian	Male	Grace	Ray	Pippa	Female	Grace	Ray	Yes
Brian	Male	Grace	Ray	Pippa	Female	Grace	Ray	Yes
Anna	Female	Pam	Ian	Nikki	Female	Pam	Ian	Yes
Nikki	Female	Pam	Ian	Anna	Female	Pam	Ian	Yes
All the rest								No

```
If second person's gender = female
   and first person's parent = second person's parent
   then sister-of = yes
```

Generating a flat file

- Process of flattening a file is called "denormalization"
  - Several relations are joined together to make one
- Possible with any finite set of finite relations
- Problematic: relationships without pre-specified number of objects
  - Example: concept of *nuclear-family*
- Denormalization may produce spurious regularities that reflect structure of database
  - Example: "supplier" predicts "supplier address"

*The "ancestor-of" relation

First person				Second person				Sister of?
Name	Gender	Parent1	Parent2	Name	Gender	Parent1	Parent2	
Peter	Male	?	?	Steven	Male	Peter	Peggy	Yes
Peter	Male	?	?	Pam	Female	Peter	Peggy	Yes
Peter	Male	?	?	Anna	Female	Pam	Ian	Yes
Peter	Male	?	?	Nikki	Female	Pam	Ian	Yes
Pam	Female	Peter	Peggy	Nikki	Female	Pam	Ian	Yes
Grace	Female	?	?	Ian	Male	Grace	Ray	Yes
Grace	Female	?	?	Nikki	Female	Pam	Ian	Yes
Other positive examples here								Yes
All the rest								No

*Recursion

- Infinite relations require recursion

```
If person1 is a parent of person2
   then person1 is an ancestor of person2

If person1 is a parent of person2
   and person2 is an ancestor of person3
   then person1 is an ancestor of person3
```

- Appropriate techniques are known as "inductive logic programming"
  - (e.g. Quinlan's FOIL)
  - Problems: (a) noise and (b) computational complexity

*Multi-instance problems

- Each example consists of several instances
- E.g. predicting drug activity
  - Examples are molecules that are active/not active
  - Instances are confirmations of a molecule
  - Molecule active (example positive)
    ⇨ at least one of its confirmations (instances) is active (positive)
  - Molecule not active (example negative)
    ⇨ all of its confirmations (instances) are not active (negative)
- Problem:
  identifying the "truly" positive instances

What's in an attribute?

- Each instance is described by a fixed predefined set of features, its "attributes"
- But: number of attributes may vary in practice
  - Possible solution: "irrelevant value" flag
- Related problem: existence of an attribute may depend of value of another one
- Possible attribute types ("levels of measurement"):
  - *Nominal, ordinal, interval* and *ratio*

Nominal quantities

- Values are distinct symbols
  - Values themselves serve only as labels or names
  - *Nominal* comes from the Latin word for name
- Example: attribute "outlook" from weather data
  - Values: "sunny","overcast", and "rainy"
- No relation is implied among nominal values (no ordering or distance measure)
- Only equality tests can be performed

Ordinal quantities

- Impose order on values
- But: no distance between values defined
- Example:
  attribute "temperature" in weather data
  - Values: "hot" > "mild" > "cool"
- Note: addition and subtraction don't make sense
- Example rule:
  temperature < hot ⇨ play = yes
- Distinction between nominal and ordinal not always clear (e.g. attribute "outlook")

Interval quantities (Numeric)

- Interval quantities are not only ordered but measured in fixed and equal units
- Example 1: attribute "temperature" expressed in degrees Fahrenheit
- Example 2: attribute "year"
- Difference of two values makes sense
- Sum or product doesn't make sense
  - Zero point is not defined!

Ratio quantities

- Ratio quantities are ones for which the measurement scheme defines a zero point
- Example: attribute "distance"
  - Distance between an object and itself is zero
- Ratio quantities are treated as real numbers
  - All mathematical operations are allowed
- But: is there an "inherently" defined zero point?
  - Answer depends on scientific knowledge (e.g. Fahrenheit knew no lower limit to temperature)

Attribute types used in practice

- Most schemes accommodate just two levels of measurement: nominal and ordinal
- Nominal attributes are also called "categorical", "enumerated", or "discrete"
  - But: "enumerated" and "discrete" imply order
- Special case: dichotomy ("boolean" attribute)
- Ordinal attributes are called "numeric", or "continuous"
  - But: "continuous" implies mathematical continuity

Attribute types: Summary

- Nominal, e.g. eye color=brown, blue, …
  - only equality tests
  - important special case: boolean (True/False)
- Ordinal, e.g. grade=k,1,2,..,12
- Continuous (numeric), e.g. year
  - interval quantities – integer
  - ratio quantities -- real

Why specify attribute types?

- ***Q: Why Machine Learning algorithms need to know about attribute type?***
- A: To be able to make right comparisons and learn correct concepts, e.g.
  - **Outlook > "sunny"** does not make sense, while
  - **Temperature > "cool"** or
  - **Humidity > 70** does
- Additional uses of attribute type: check for valid values, deal with missing, etc.

Transforming ordinal to boolean

- Simple transformation allows ordinal attribute with *n* values to be coded using *n–1* boolean attributes
- Example: attribute "temperature"

Original data

Temperature
Cold
Medium
Hot

⇨

Transformed data

Temperature > cold	Temperature > medium
False	False
True	False
True	True

- Better than coding it as a nominal attribute

Metadata

- Information about the data that encodes background knowledge
- Can be used to restrict search space
- Examples:
  - Dimensional considerations (i.e. expressions must be dimensionally correct)
  - Circular orderings (e.g. degrees in compass)
  - Partial orderings (e.g. generalization/specialization relations)

Preparing the input

- Problem: different data sources (e.g. sales department, customer billing department, …)
  - Differences: styles of record keeping, conventions, time periods, data aggregation, primary keys, errors
  - Data must be assembled, integrated, cleaned up
  - "Data warehouse": consistent point of access
- Denormalization is not the only issue
- External data may be required ("overlay data")
- Critical: type and level of data aggregation

The ARFF format

```
%
% ARFF file for weather data with some numeric features
%
@relation weather

@attribute outlook {sunny, overcast, rainy}
@attribute temperature numeric
@attribute humidity numeric
@attribute windy {true, false}
@attribute play? {yes, no}

@data
sunny, 85, 85, false, no
sunny, 80, 90, true, no
overcast, 83, 86, false, yes
...
```

Attribute types in Weka

- ARFF supports numeric and nominal attributes
- Interpretation depends on learning scheme
  - Numeric attributes are interpreted as
    - ordinal scales if less-than and greater-than are used
    - ratio scales if distance calculations are performed (normalization/ standardization may be required)
  - Instance-based schemes define distance between nominal values (0 if values are equal, 1 otherwise)
- Integers: nominal, ordinal, or ratio scale?

Nominal vs. ordinal

- Attribute "age" nominal

```
If age = young and astigmatic = no
    and tear production rate = normal
    then recommendation = soft
If age = pre-presbyopic and astigmatic = no
    and tear production rate = normal
    then recommendation = soft
```

- Attribute "age" ordinal

(e.g. "young" < "pre-presbyopic" < "presbyopic")

```
If age ≤ pre-presbyopic and astigmatic = no
   and tear production rate = normal
   then recommendation = soft
```

Missing values

- Frequently indicated by out-of-range entries
  - Types: unknown, unrecorded, irrelevant
  - Reasons:
    - malfunctioning equipment
    - changes in experimental design
    - collation of different datasets
    - measurement not possible
- Missing value may have significance in itself (e.g. missing test in a medical examination)
  - Most schemes assume that is not the case
    ⇨ "missing" may need to be coded as additional value

Missing values - example

- Value may be missing because it is unrecorded or because it is inapplicable
- In medical data, value for **Pregnant?** attribute for **Jane** is missing, while for **Joe** or **Anna** should be considered **Not applicable**
- Some programs can infer missing values

Hospital Check-in Database

Name	Age	Sex	Pregnant?	..
Mary	25	F	N	
Jane	27	F	-	
Joe	30	M	-	
Anna	2	F	-	

Inaccurate values

- Reason: data has not been collected for mining it
- Result: errors and omissions that don't affect original purpose of data (e.g. age of customer)
- Typographical errors in nominal attributes ⇒ values need to be checked for consistency
- Typographical and measurement errors in numeric attributes ⇒ outliers need to be identified
- Errors may be deliberate (e.g. wrong zip codes)
- Other problems: duplicates, stale data

Precision "Illusion"

- Example: gene expression may be reported as X83 = 193.3742, but measurement error may be +/- 20.
- Actual value is in [173, 213] range, so it is appropriate to round the data to 190.
- Don't assume that every reported digit is significant!

Getting to know the data

- Simple visualization tools are very useful
  - Nominal attributes: histograms (Distribution consistent with background knowledge?)
  - Numeric attributes: graphs
    (Any obvious outliers?)
- 2-D and 3-D plots show dependencies
- Need to consult domain experts
- Too much data to inspect? Take a sample!

Summary

- Concept: thing to be learned
- Instance: individual examples of a concept
- Attributes: Measuring aspects of an instance

- *Note: Don't confuse learning "Class" and "Instance" with Java "Class" and "instance"*

Knowledge Representation

Outline: Output - Knowledge representation

- Decision tables
- Decision trees
- Decision rules
- Rules involving relations
- Instance-based representation
  - Prototypes, Clusters

Output: representing structural patterns

- Many different ways of representing patterns
  - Decision trees, rules, instance-based, …
- Also called "knowledge" representation
- Representation determines inference method
- Understanding the output is the key to understanding the underlying learning methods
- Different types of output for different learning problems (e.g. classification, regression, …)

Decision tables

- Simplest way of representing output:
  - Use the same format as input!
- Decision table for the weather problem:

Outlook	Humidity	Play
Sunny	High	No
Sunny	Normal	Yes
Overcast	High	Yes
Overcast	Normal	Yes
Rainy	High	No
Rainy	Normal	No

- Main problem: selecting the right attributes
- Also, not flexible enough

Decision trees

- "Divide-and-conquer" approach produces tree
- Nodes involve testing a particular attribute
- Usually, attribute value is compared to constant
- Other possibilities:
  - Comparing values of two attributes
  - Using a function of one or more attributes
- Leaves assign classification, set of classifications, or probability distribution to instances
- Unknown instance is routed down the tree

Nominal and numeric attributes

- Nominal:
  number of children usually equal to number values
  ⇒ attribute won't get tested more than once
  - Other possibility: division into two subsets
- Numeric:
  test whether value is greater or less than constant
  ⇒ attribute may get tested several times
  - Other possibility: three-way split (or multi-way split)
    - Integer: *less than, equal to, greater than*
    - Real: *below, within, above*

Missing values

- Does absence of value have some significance?
- Yes ⇒ "missing" is a separate value
- No ⇒ "missing" must be treated in a special way
  - Solution A: assign instance to most popular branch
  - Solution B: split instance into pieces
    - Pieces receive weight according to fraction of training instances that go down each branch
    - Classifications from leave nodes are combined using the weights that have percolated to them

Classification rules

- Popular alternative to decision trees
- *Antecedent* (pre-condition): a series of tests (just like the tests at the nodes of a decision tree)
- Tests are usually logically ANDed together (but may also be general logical expressions)
- *Consequent* (conclusion): classes, set of classes, or probability distribution assigned by rule
- Individual rules are often logically ORed together
  - Conflicts arise if different conclusions apply

From trees to rules

- Easy: converting a tree into a set of rules
  - One rule for each leaf:
    - Antecedent contains a condition for every node on the path from the root to the leaf
    - Consequent is class assigned by the leaf
- Produces rules that are unambiguous
  - Doesn't matter in which order they are executed
- But: resulting rules are unnecessarily complex
  - Pruning to remove redundant tests/rules

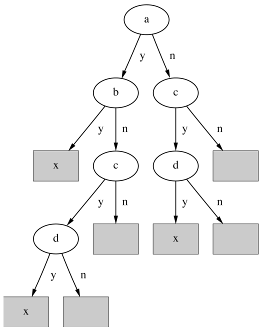
From rules to trees

- More difficult: transforming a rule set into a tree
  - Tree cannot easily express disjunction between rules
- Example: rules which test different attributes

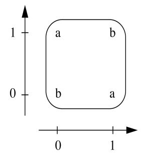
```
If a and b then x
If c and d then x
```

- Symmetry needs to be broken
- Corresponding tree contains identical subtrees (⇒ "replicated subtree problem")

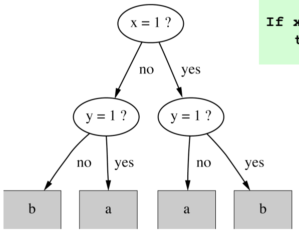
A tree for a simple disjunction



The exclusive-or problem

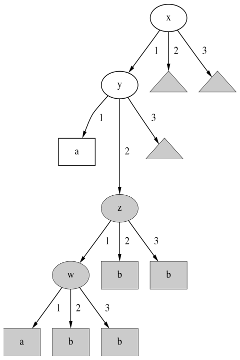


```
If x = 1 and y = 0
   then class = a
If x = 0 and y = 1
   then class = a
If x = 0 and y = 0
   then class = b
If x = 1 and y = 1
   then class = b
```



A tree with a replicated subtree

```
If x = 1 and y = 1
   then class = a
If z = 1 and w = 1
   then class = a
Otherwise class = b
```



"Nuggets" of knowledge

- Are rules independent pieces of knowledge? (It seems easy to add a rule to an existing rule base.)
- Problem: ignores how rules are executed
- Two ways of executing a rule set:
  - Ordered set of rules ("decision list")
    - Order is important for interpretation
  - Unordered set of rules
    - Rules may overlap and lead to different conclusions for the same instance

Interpreting rules

- What if two or more rules conflict?
  - Give no conclusion at all?
  - Go with rule that is most popular on training data?
  - …
- What if no rule applies to a test instance?
  - Give no conclusion at all?
  - Go with class that is most frequent in training data?
  - …

Special case: boolean class

- Assumption: if instance does not belong to class "yes", it belongs to class "no"
- Trick: only learn rules for class "yes" and use default rule for "no"

```
If x = 1 and y = 1 then class = a
If z = 1 and w = 1 then class = a
Otherwise class = b
```

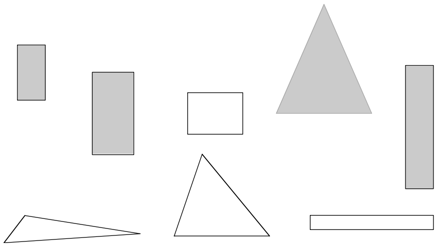
- Order of rules is not important. No conflicts!
- Rule can be written in *disjunctive normal form*

Rules involving relations

- So far: all rules involved comparing an attribute-value to a constant (e.g. temperature < 45)
- These rules are called "propositional" because they have the same expressive power as propositional logic
- What if problem involves relationships between examples (e.g. family tree problem from above)?
  - Can't be expressed with propositional rules
  - More expressive representation required

The shapes problem

- Target concept: *standing up*
- Shaded: *standing*
  Unshaded: *lying*



A propositional solution

Width	Height	Sides	Class
2	4	4	Standing
3	6	4	Standing
4	3	4	Lying
7	8	3	Standing
7	6	3	Lying
2	9	4	Standing
9	1	4	Lying
10	2	3	Lying

```
If width ≥ 3.5 and height < 7.0
   then lying
If height ≥ 3.5 then standing
```

A relational solution

- ❖Comparing attributes with each other

```
If width > height then lying
If height > width then standing
```

- ❖Generalizes better to new data
- ❖Standard relations: =, <, >
- ❖But: learning relational rules is costly
- ❖Simple solution: add extra attributes (e.g. a binary attribute *is width < height?*)

Rules with variables

- Using variables and multiple relations:

```
If height_and_width_of(x,h,w) and h > w
   then standing(x)
```

- The top of a tower of blocks is standing:

```
If height_and_width_of(x,h,w) and h > w
     and is_top_of(x,y)
   then standing(x)
```

- The whole tower is standing:

```
If height_and_width_of(z,h,w) and h > w
     and is_top_of(x,z) and standing(y)
     and is_rest_of(x,y)
   then standing(x)
If empty(x) then standing(x)
```

- Recursive definition!

Inductive logic programming

- Recursive definition can be seen as logic program
- Techniques for learning logic programs stem from the area of "inductive logic programming" (ILP)
- But: recursive definitions are hard to learn
  - Also: few practical problems require recursion
  - Thus: many ILP techniques are restricted to non-recursive definitions to make learning easier

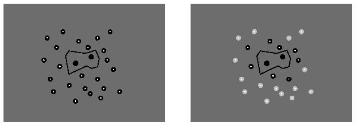
Instance-based representation

- Simplest form of learning: rote learning
  - Training instances are searched for instance that most closely resembles new instance
  - The instances themselves represent the knowledge
  - Also called instance-based learning
- Similarity function defines what's "learned"
- Instance-based learning is *lazy* learning
- Methods: k-nearest-neighbor, …

The distance function

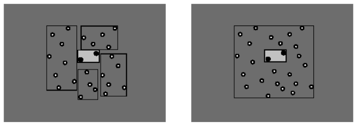
- Simplest case: one numeric attribute
  - Distance is the difference between the two attribute values involved (or a function thereof)
- Several numeric attributes: normally, Euclidean distance is used and attributes are normalized
- Nominal attributes: distance is set to 1 if values are different, 0 if they are equal
- Are all attributes equally important?
  - Weighting the attributes might be necessary

Learning prototypes



- Only those instances involved in a decision need to be stored
- Noisy instances should be filtered out
- Idea: only use *prototypical* examples

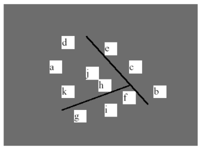
Rectangular generalizations



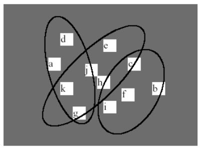
- Nearest-neighbor rule is used outside rectangles
- Rectangles are rules! (But they can be more conservative than "normal" rules.)
- Nested rectangles are rules with exceptions

Representing clusters I

Simple 2-D representation



Venn diagram



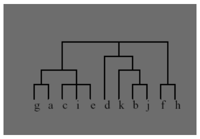
Overlapping clusters

Representing clusters II

Probabilistic assignment

	1	2	3
a	0.4	0.1	0.5
b	0.1	0.8	0.1
c	0.3	0.3	0.4
d	0.1	0.1	0.8
e	0.4	0.2	0.4
f	0.1	0.4	0.5
g	0.7	0.2	0.1
h	0.5	0.4	0.1

Dendrogram



NB: dendron is the Greek word for tree

Summary

- Trees
- Rules
- Relational representation
- Instance-based representation

Algorithms for Classification:

The Basic Methods

Outline

- Simplicity first: 1R

- Naïve Bayes

Classification

- Task: Given a set of pre-classified examples, build a model or *classifier* to classify new cases.
- *Supervised* learning: classes are known for the examples used to build the classifier.
- A classifier can be a set of rules, a decision tree, a neural network, etc.
- Typical applications: credit approval, direct marketing, fraud detection, medical diagnosis, …..

Simplicity first

- Simple algorithms often work very well!
- There are many kinds of simple structure, eg:
  - One attribute does all the work
  - All attributes contribute equally & independently
  - A weighted linear combination might do
  - Instance-based: use a few prototypes
  - Use simple logical rules
- Success of method depends on the domain

Inferring rudimentary rules

- 1R: learns a 1-level decision tree
  - I.e., rules that all test one particular attribute
- Basic version
  - One branch for each value
  - Each branch assigns most frequent class
  - Error rate: proportion of instances that don’t belong to the majority class of their corresponding branch
  - Choose attribute with lowest error rate

(*assumes nominal attributes*)

Pseudo-code for 1R

```
For each attribute,
    For each value of the attribute, make a rule as follows:
        count how often each class appears
        find the most frequent class
        make the rule assign that class to this attribute-value
    Calculate the error rate of the rules
Choose the rules with the smallest error rate
```

- Note: “missing” is treated as a separate attribute value

Evaluating the weather attributes

Outlook	Temp	Humidity	Windy	Play
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Overcast	Hot	High	False	Yes
Rainy	Mild	High	False	Yes
Rainy	Cool	Normal	False	Yes
Rainy	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Sunny	Mild	High	False	No
Sunny	Cool	Normal	False	Yes
Rainy	Mild	Normal	False	Yes
Sunny	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Rainy	Mild	High	True	No

Attribute	Rules	Errors	Total errors
Outlook	Sunny → No	2/5	4/14
	Overcast → Yes	0/4	
	Rainy → Yes	2/5	
Temp	Hot → No*	2/4	5/14
	Mild → Yes	2/6	
	Cool → Yes	1/4	
Humidity	High → No	3/7	4/14
	Normal → Yes	1/7	
Windy	False → Yes	2/8	5/14
	True → No*	3/6	

* indicates a tie

Dealing with numeric attributes

- Discretize numeric attributes
- Divide each attribute’s range into intervals
  - Sort instances according to attribute’s values
  - Place breakpoints where the class changes (the majority class)
  - This minim
- Example: *ten*

Outlook	Temperature	Humidity	Windy	Play
Sunny	85	85	False	No
Sunny	80	90	True	No
Overcast	83	86	False	Yes
Rainy	75	80	False	Yes
…	…	…	…	…

```
64    65    68   69   70    71  72  72    75   75    80   81    83    85
Yes | No  | Yes  Yes  Yes | No  No  Yes | Yes  Yes | No | Yes   Yes | No
```

The problem of overfitting

- This procedure is very sensitive to noise
  - One instance with an incorrect class label will probably produce a separate interval
- Also: *time stamp* attribute will have zero errors
- Simple solution:
  enforce minimum number of instances in majority class per interval

Discretization example

- Example (with min = 3):

```
64     65     68    69    70     71   72   72     75    75     80     81     83     85
Yes  ⊘ No   ⊘ Yes  Yes  Yes  |  No   No   Yes  ⊘  Yes  Yes  |  No  ⊘ Yes    Yes  ⊘ No
```

- Final result for temperature attribute

```
64     65     68    69    70     71   72   72     75     75     80     81     83     85
Yes    No     Yes  Yes  Yes  ⊘  No   No   Yes     Yes  Yes  |  No     Yes    Yes    No
```

With overfitting avoidance

- Resulting rule set:

Attribute	Rules	Errors	Total errors
Outlook	Sunny → No	2/5	4/14
	Overcast → Yes	0/4	
	Rainy → Yes	2/5	
Temperature	≤ 77.5 → Yes	3/10	5/14
	> 77.5 → No*	2/4	
Humidity	≤ 82.5 → Yes	1/7	3/14
	> 82.5 and ≤ 95.5 → No	2/6	
	> 95.5 → Yes	0/1	
Windy	False → Yes	2/8	5/14
	True → No*	3/6	

Discussion of 1R

- 1R was described in a paper by Holte (1993)
  - Contains an experimental evaluation on 16 datasets (using *cross-validation* so that results were representative of performance on future data)
  - Minimum number of instances was set to 6 after some experimentation
  - 1R's simple rules performed not much worse than much more complex decision trees
- Simplicity first pays off!

Very Simple Classification Rules Perform Well on Most Commonly Used Datasets

Robert C. Holte, Computer Science Department, University of Ottawa

Bayesian (Statistical) modeling

- “Opposite” of 1R: use all the attributes
- Two assumptions: Attributes are
  - *equally important*
  - *statistically independent* (given the class value)
    - I.e., knowing the value of one attribute says nothing about the value of another
(if the class is known)
- Independence assumption is almost never correct!
- But … this scheme works well in practice

Probabilities for weather data

Outlook			Temperature			Humidity			Windy			Play	
	Yes	*No*		*Yes*	*No*		*Yes*	*No*		*Yes*	*No*	*Yes*	*No*
Sunny	2	3	Hot	2	2	High	3	4	False	6	2	9	5
Overcast	4	0	Mild	4	2	Normal	6	1	True	3	3		
Rainy	3	2	Cool	3	1								
Sunny	2/9	3/5	Hot	2/9	2/5	High	3/9	4/5	False	6/9	2/5	9/14	5/14
Overcast	4/9	0/5	Mild	4/9	2/5	Normal	6/9	1/5	True	3/9	3/5		
Rainy	3/9	2/5	Cool	3/9	1/5								

Outlook	Temp	Humidity	Windy	Play
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Overcast	Hot	High	False	Yes
Rainy	Mild	High	False	Yes
Rainy	Cool	Normal	False	Yes
Rainy	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Sunny	Mild	High	False	No
Sunny	Cool	Normal	False	Yes
Rainy	Mild	Normal	False	Yes
Sunny	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Rainy	Mild	High	True	No

Probabilities for weather data

Outlook			Temperature			Humidity			Windy			Play	
	Yes	*No*		*Yes*	*No*		*Yes*	*No*		*Yes*	*No*	*Yes*	*No*
Sunny	2	3	Hot	2	2	High	3	4	False	6	2	9	5
Overcast	4	0	Mild	4	2	Normal	6	1	True	3	3		
Rainy	3	2	Cool	3	1								
Sunny	2/9	3/5	Hot	2/9	2/5	High	3/9	4/5	False	6/9	2/5	9/14	5/14
Overcast	4/9	0/5	Mild	4/9	2/5	Normal	6/9	1/5	True	3/9	3/5		
Rainy	3/9	2/5	Cool	3/9	1/5								

- A new day:

Outlook	Temp.	Humidity	Windy	Play
Sunny	Cool	High	True	?

Likelihood of the two classes

For "yes" = 2/9 × 3/9 × 3/9 × 3/9 × 9/14 = 0.0053

For "no" = 3/5 × 1/5 × 4/5 × 3/5 × 5/14 = 0.0206

Conversion into a probability by normalization:

P("yes") = 0.0053 / (0.0053 + 0.0206) = 0.205

P("no") = 0.0206 / (0.0053 + 0.0206) = 0.795

Bayes's rule

- Probability of event *H* given evidence *E* :

$$\Pr[H \mid E] = \frac{\Pr[E \mid H]\Pr[H]}{\Pr[E]}$$

- *A priori* probability of *H* : $\Pr[H]$
  - Probability of event *before* evidence is seen
- *A posteriori* probability of *H* : $\Pr[H \mid E]$
  - Probability of event *after* evidence is seen

from Bayes "Essay towards solving a problem in the doctrine of chances" (1763)

Thomas Bayes

Born: **1702 in London, England**
Died: **1761 in Tunbridge Wells, Kent, England**

Naïve Bayes for classification

- Classification learning: what’s the probability of the class given an instance?
  - Evidence *E* = instance
  - Event *H* = class value for instance
- Naïve assumption: evidence splits into parts (i.e. attributes) that are *independent*

$$\Pr[H \mid E] = \frac{\Pr[E_1 \mid H]\Pr[E_1 \mid H]\ldots\Pr[E_n \mid H]\Pr[H]}{\Pr[E]}$$

Weather data example

Outlook	Temp.	Humidity	Windy	Play
Sunny	Cool	High	True	?

← ***Evidence E***

Probability of class "yes"

$$\begin{aligned}\Pr[yes \mid E] = {} & \Pr[Outlook = Sunny \mid yes] \\ & \times \Pr[Temperature = Cool \mid yes] \\ & \times \Pr[Humidity = High \mid yes] \\ & \times \Pr[Windy = True \mid yes] \\ & \times \frac{\Pr[yes]}{\Pr[E]}\end{aligned}$$

$$= \frac{\frac{2}{9} \times \frac{3}{9} \times \frac{3}{9} \times \frac{3}{9} \times \frac{9}{14}}{\Pr[E]}$$

The “zero-frequency problem”

- What if an attribute value doesn’t occur with every class value?
  (e.g. “Humidity = high” for class “yes”)
  - Probability will be zero!

    $$\Pr[Humidity = High \mid yes] = 0$$

  - *A posteriori* probability will also be zero!
    (No matter how likely the other values are!)

    $$\Pr[yes \mid E] = 0$$

- Remedy: add 1 to the count for every attribute value-class combination (*Laplace estimator)*
- Result: probabilities will never be zero!
  (also: stabilizes probability estimates)

*Modified probability estimates

- In some cases adding a constant different from 1 might be more appropriate
- Example: attribute *outlook* for class *yes*

$$\frac{2+\mu/3}{9+\mu} \qquad \frac{4+\mu/3}{9+\mu} \qquad \frac{3+\mu/3}{9+\mu}$$

Sunny ***Overcast*** ***Rainy***

- Weights don’t need to be equal (but they must sum to 1)

$$\frac{2+\mu p_1}{9+\mu} \qquad \frac{4+\mu p_2}{9+\mu} \qquad \frac{3+\mu p_3}{9+\mu}$$

Missing values

- Training: instance is not included in frequency count for attribute value-class combination
- Classification: attribute will be omitted from calculation
- Example:

Outlook	Temp.	Humidity	Windy	Play
?	Cool	High	True	?

Likelihood of “yes” = 3/9 × 3/9 × 3/9 × 9/14 = 0.0238

Likelihood of “no” = 1/5 × 4/5 × 3/5 × 5/14 = 0.0343

P(“yes”) = 0.0238 / (0.0238 + 0.0343) = 41%

P(“no”) = 0.0343 / (0.0238 + 0.0343) = 59%

Numeric attributes

- Usual assumption: attributes have a *normal* or *Gaussian* probability distribution (given the class)
- The *probability density function* for the normal distribution is defined by two parameters:
  - *Sample mean* μ

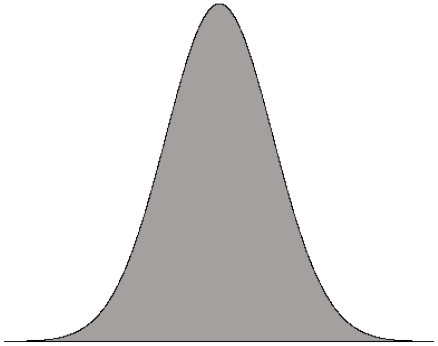
$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i$$

  - *Standard deviation* σ

$$\sigma = \frac{1}{n-1}\sum_{i=1}^{n} (x_i - \mu)^2$$

  - Then the density function *f(x) is*

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



Karl Gauss, 1777-1855
great German mathematician

Statistics for weather data

Outlook			Temperature		Humidity		Windy			Play	
	Yes	*No*	*Yes*	*No*	*Yes*	*No*		*Yes*	*No*	*Yes*	*No*
Sunny	2	3	64, 68,	65, 71,	65, 70,	70, 85,	False	6	2	9	5
Overcast	4	0	69, 70,	72, 80,	70, 75,	90, 91,	True	3	3		
Rainy	3	2	72, …	85, …	80, …	95, …					
Sunny	2/9	3/5	μ =73	μ =75	μ =79	μ =86	False	6/9	2/5	9/14	5/14
Overcast	4/9	0/5	σ =6.2	σ =7.9	σ =10.2	σ =9.7	True	3/9	3/5		
Rainy	3/9	2/5									

- Example density value:

$$f(temperature = 66 \mid yes) = \frac{1}{\sqrt{2\pi}\,6.2} e^{-\frac{(66-73)^2}{2*6.2^2}} = 0.0340$$

Classifying a new day

- A new day:

Outlook	Temp.	Humidity	Windy	Play
Sunny	66	90	true	?

Likelihood of "yes" = 2/9 × 0.0340 × 0.0221 × 3/9 × 9/14 = 0.000036

Likelihood of "no" = 3/5 × 0.0291 × 0.0380 × 3/5 × 5/14 = 0.000136

P("yes") = 0.000036 / (0.000036 + 0. 000136) = 20.9%

P("no") = 0.000136 / (0.000036 + 0. 000136) = 79.1%

- Missing values during training are not included in calculation of mean and standard deviation

*Probability densities

- Relationship between probability and density:

$$\Pr[c-\frac{\varepsilon}{2}<x<c+\frac{\varepsilon}{2}]\approx\varepsilon*f(c)$$

- But: this doesn't change calculation of *a posteriori* probabilities because ε cancels out
- Exact relationship:

$$\Pr[a\le x\le b]=\int_a^b f(t)dt$$

Naïve Bayes: discussion

- Naïve Bayes works surprisingly well (even if independence assumption is clearly violated)
- Why? Because classification doesn't require accurate probability estimates *as long as maximum probability is assigned to correct class*
- However: adding too many redundant attributes will cause problems (e.g. identical attributes)
- Note also: many numeric attributes are not normally distributed (→ *kernel density estimators*)

Naïve Bayes Extensions

- Improvements:
  - select best attributes (e.g. with greedy search)
  - often works as well or better with just a fraction of all attributes
- Bayesian Networks

Summary

- OneR – uses rules based on just one attribute

- Naïve Bayes – use all attributes and Bayes rules to estimate probability of the class given an instance.

- Simple methods frequently work well, but …
  - Complex methods can be better (as we will see)

Classification: Decision Trees

Outline

- Top-Down Decision Tree Construction
- Choosing the Splitting Attribute
- Information Gain and Gain Ratio

DECISION TREE

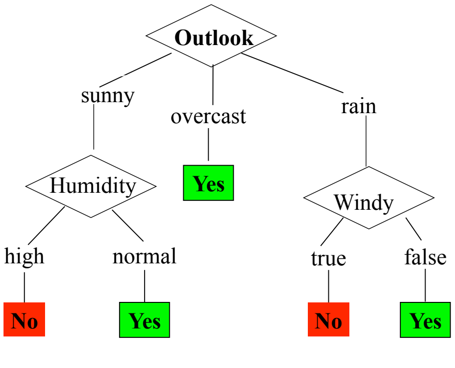
- An internal node is a test on an attribute.
- A branch represents an outcome of the test, e.g., Color=red.
- A leaf node represents a class label or class label distribution.
- At each node, one attribute is chosen to split training examples into distinct classes as much as possible
- A new case is classified by following a matching path to a leaf node.

Weather Data: Play or not Play?

Outlook	Temperature	Humidity	Windy	Play?
sunny	hot	high	false	No
sunny	hot	high	true	No
overcast	hot	high	false	Yes
rain	mild	high	false	Yes
rain	cool	normal	false	Yes
rain	cool	normal	true	No
overcast	cool	normal	true	Yes
sunny	mild	high	false	No
sunny	cool	normal	false	Yes
rain	mild	normal	false	Yes
sunny	mild	normal	true	Yes
overcast	mild	high	true	Yes
overcast	hot	normal	false	Yes
rain	mild	high	true	No

Note:
Outlook is the Forecast,
no relation to Microsoft email program

Example Tree for "Play?"



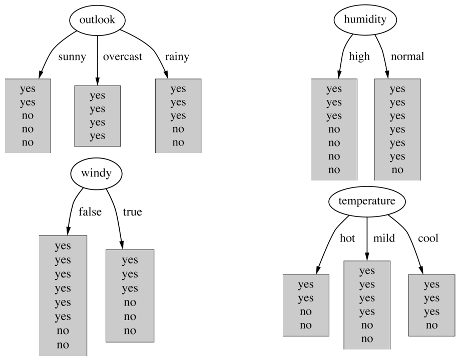
Building Decision Tree [Q93]

- Top-down tree construction
  - At start, all training examples are at the root.
  - Partition the examples recursively by choosing one attribute each time.
- Bottom-up tree pruning
  - Remove subtrees or branches, in a bottom-up manner, to improve the estimated accuracy on new cases.

Choosing the Splitting Attribute

- At each node, available attributes are evaluated on the basis of separating the classes of the training examples. A Goodness function is used for this purpose.
- Typical goodness functions:
  - information gain (ID3/C4.5)
  - information gain ratio
  - gini index

Which attribute to select?



A criterion for attribute selection

- Which is the best attribute?
  - The one which will result in the smallest tree
  - Heuristic: choose the attribute that produces the "purest" nodes
- Popular *impurity criterion*: *information gain*
  - Information gain increases with the average purity of the subsets that an attribute produces
- Strategy: choose attribute that results in greatest information gain

Computing information

- Information is measured in *bits*
  - Given a probability distribution, the info required to predict an event is the distribution's *entropy*
  - Entropy gives the information required in bits (this can involve fractions of bits!)
- Formula for computing the entropy:

$$\text{entropy}(p_1, p_2, \ldots, p_n) = -p_1 \log p_1 - p_2 \log p_2 \ldots - p_n \log p_n$$

*Claude Shannon

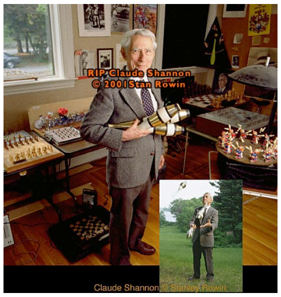
"Father of information theory"

Born: 30 April 1916

Died: 23 February 2001

Claude Shannon, who has died aged 84, perhaps more than anyone laid the groundwork for today's digital revolution. His exposition of information theory, stating that all information could be represented mathematically as a succession of noughts and ones, facilitated the digital manipulation of data without which today's information society would be unthinkable.

Shannon's master's thesis, obtained in 1940 at MIT, demonstrated that problem solving could be achieved by manipulating the symbols 0 and 1 in a process that could be carried out automatically with electrical circuitry. That dissertation has been hailed as one of the most significant master's theses of the 20th century. Eight years later, Shannon published another landmark paper, *A Mathematical Theory of Communication*, generally taken as his most important scientific contribution.



Shannon applied the same radical approach to cryptography research, in which he later became a consultant to the US government.

Many of Shannon's pioneering insights were developed before they could be applied in practical form. He was truly a remarkable man, yet unknown to most of the world.

Example: attribute "Outlook"

- "Outlook" = "Sunny":

$$\text{info}([2,3]) = \text{entropy}(2/5,3/5) = -2/5\log(2/5) - 3/5\log(3/5) = 0.971 \text{ bits}$$

- "Outlook" = "Overcast":

$$\text{info}([4,0]) = \text{entropy}(1,0) = -1\log(1) - 0\log(0) = 0 \text{ bits}$$

*Note: log(0) is not defined, but we evaluate 0*log(0) as zero*

- "Outlook" = "Rainy":

$$\text{info}([3,2]) = \text{entropy}(3/5,2/5) = -3/5\log(3/5) - 2/5\log(2/5) = 0.971 \text{ bits}$$

- Expected information for attribute:

$$\text{info}([3,2],[4,0],[3,2]) = (5/14)\times 0.971 + (4/14)\times 0 + (5/14)\times 0.971$$
$$= 0.693 \text{ bits}$$

Computing the information gain

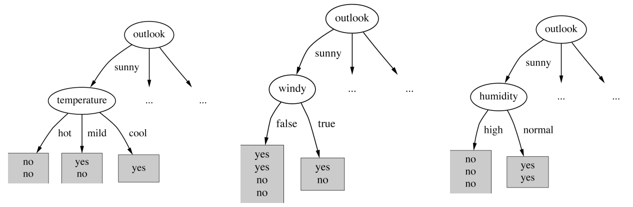
- Information gain:

(information before split) – (information after split)

$$\text{gain}(\text{"Outlook"}) = \text{info}([9,5]) - \text{info}([2,3],[4,0],[3,2]) = 0.940 - 0.693$$
$$= 0.247 \text{ bits}$$

- Information gain for attributes from weather data:
  $\text{gain}(\text{"Outlook"}) = 0.247 \text{ bits}$
  $\text{gain}(\text{"Temperature"}) = 0.029 \text{ bits}$
  $\text{gain}(\text{"Humidity"}) = 0.152 \text{ bits}$
  $\text{gain}(\text{"Windy"}) = 0.048 \text{ bits}$

Continuing to split

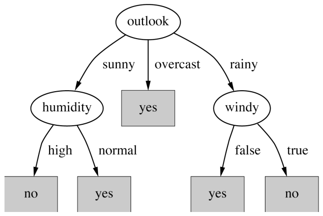


$\text{gain}(\text{"Temperature"}) = 0.571 \text{ bits}$

$\text{gain}(\text{"Humidity"}) = 0.971 \text{ bits}$

$\text{gain}(\text{"Windy"}) = 0.020 \text{ bits}$

The final decision tree



- Note: not all leaves need to be pure; sometimes identical instances have different classes

  ⇒ Splitting stops when data can't be split any further

*Wish list for a purity measure

- Properties we require from a purity measure:
  - When node is pure, measure should be zero
  - When impurity is maximal (i.e. all classes equally likely), measure should be maximal
  - Measure should obey *multistage property* (i.e. decisions can be made in several stages):

$$\text{measure}([2,3,4]) = \text{measure}([2,7]) + (7/9) \times \text{measure}([3,4])$$

- Entropy is a function that satisfies all three properties!

*Properties of the entropy

- The multistage property:

$$\text{entropy}(p,q,r) = \text{entropy}(p,q+r) + (q+r) \times \text{entropy}(\frac{q}{q+r}, \frac{r}{q+r})$$

- Simplification of computation:

$$\text{info}([2,3,4]) = -2/9 \times \log(2/9) - 3/9 \times \log(3/9) - 4/9 \times \log(4/9)$$
$$= [-2\log 2 - 3\log 3 - 4\log 4 + 9\log 9]/9$$

- Note: instead of maximizing info gain we could just minimize information

Highly-branching attributes

- Problematic: attributes with a large number of values (extreme case: ID code)
- Subsets are more likely to be pure if there is a large number of values

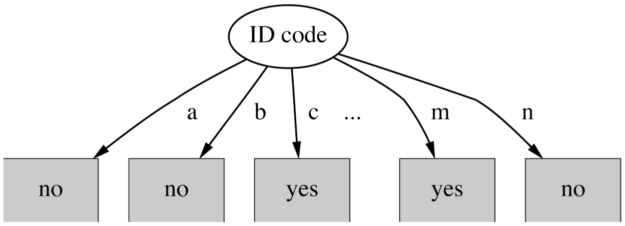
  ⇒Information gain is biased towards choosing attributes with a large number of values

  ⇒This may result in *overfitting* (selection of an attribute that is non-optimal for prediction)

Weather Data with ID code

ID	Outlook	Temperature	Humidity	Windy	Play?
A	sunny	hot	high	false	No
B	sunny	hot	high	true	No
C	overcast	hot	high	false	Yes
D	rain	mild	high	false	Yes
E	rain	cool	normal	false	Yes
F	rain	cool	normal	true	No
G	overcast	cool	normal	true	Yes
H	sunny	mild	high	false	No
I	sunny	cool	normal	false	Yes
J	rain	mild	normal	false	Yes
K	sunny	mild	normal	true	Yes
L	overcast	mild	high	true	Yes
M	overcast	hot	normal	false	Yes
N	rain	mild	high	true	No

Split for ID Code Attribute



Entropy of split = 0 (since each leaf node is “pure”, having only one case.

Information gain is maximal for ID code

Gain ratio

- *Gain ratio*: a modification of the information gain that reduces its bias on high-branch attributes
- Gain ratio should be
  - Large when data is evenly spread
  - Small when all data belong to one branch
- Gain ratio takes number and size of branches into account when choosing an attribute
  - It corrects the information gain by taking the *intrinsic information* of a split into account (i.e. how much info do we need to tell which branch an instance belongs to)

Gain Ratio and Intrinsic Info.

- Intrinsic information: entropy of distribution of instances into branches

$$\mathit{IntrinsicInfo}(S,A) \equiv -\sum \frac{|S_i|}{|S|} \log_2 \frac{|S_i|}{|S|}.$$

- *Gain ratio* (Quinlan’86) normalizes info gain by:

$$\mathit{GainRatio}(S,A) = \frac{\mathit{Gain}(S,A)}{\mathit{IntrinsicInfo}(S,A)}.$$

Computing the gain ratio

- Example: intrinsic information for ID code

$$\text{info}([1,1,\ldots,1]) = 14 \times (-1/14 \times \log 1/14) = 3.807 \text{ bits}$$

- **Importance of attribute decreases as intrinsic information gets larger**
- Example of gain ratio:

$$\text{gain\_ratio}(\text{"Attribute"}) = \frac{\text{gain}(\text{"Attribute"})}{\text{intrinsic\_info}(\text{"Attribute"})}$$

- Example: $\text{gain\_ratio}(\text{"ID\_code"}) = \frac{0.940 \text{ bits}}{3.807 \text{ bits}} = 0.246$

Gain ratios for weather data

Outlook		**Temperature**	
Info:	0.693	Info:	0.911
Gain: 0.940-0.693	0.247	Gain: 0.940-0.911	0.029
Split info: info([5,4,5])	1.577	Split info: info([4,6,4])	1.362
Gain ratio: 0.247/1.577	0.156	Gain ratio: 0.029/1.362	0.021

Humidity		**Windy**	
Info:	0.788	Info:	0.892
Gain: 0.940-0.788	0.152	Gain: 0.940-0.892	0.048
Split info: info([7,7])	1.000	Split info: info([8,6])	0.985
Gain ratio: 0.152/1	0.152	Gain ratio: 0.048/0.985	0.049

More on the gain ratio

- “Outlook” still comes out top
- However: “ID code” has greater gain ratio
  - Standard fix: *ad hoc* test to prevent splitting on that type of attribute
- Problem with gain ratio: it may overcompensate
  - May choose an attribute just because its intrinsic information is very low
  - Standard fix:
    - First, only consider attributes with greater than average information gain
    - Then, compare them on gain ratio

witten&eibe

*CART Splitting Criteria: Gini Index

- If a data set T contains examples from n classes, gini index, gini(T) is defined as

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

where p_j is the relative frequency of class j in T.

gini(T) is minimized if the classes in T are skewed.

*Gini Index

After splitting T into two subsets T1 and T2 with sizes N1 and N2, the gini index of the split data is defined as

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

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

Discussion

- Algorithm for top-down induction of decision trees (“ID3”) was developed by Ross Quinlan
  - Gain ratio just one modification of this basic algorithm
  - Led to development of C4.5, which can deal with numeric attributes, missing values, and noisy data
- There are many other attribute selection criteria! (But almost no difference in accuracy of result.)

Summary

- Top-Down Decision Tree Construction
- Choosing the Splitting Attribute
- Information Gain biased towards attributes with a large number of values
- Gain Ratio takes number and size of branches into account when choosing an attribute

Machine Learning in Real World: C4.5

Outline

- Handling Numeric Attributes
  - Finding Best Split(s)
- Dealing with Missing Values
- Pruning
  - Pre-pruning, Post-pruning, Error Estimates
- From Trees to Rules

Industrial-strength algorithms

- For an algorithm to be useful in a wide range of real-world applications it must:
  - Permit numeric attributes
  - Allow missing values
  - Be robust in the presence of noise
  - Be able to approximate arbitrary concept descriptions (at least in principle)
- Basic schemes need to be extended to fulfill these requirements

C4.5 History

- ID3, CHAID – 1960s
- C4.5 innovations (Quinlan):
  - permit numeric attributes
  - deal sensibly with missing values
  - pruning to deal with for noisy data
- C4.5 - one of best-known and most widely-used learning algorithms
  - Last research version: C4.8, implemented in Weka as J4.8 (Java)
  - Commercial successor: C5.0 (available from Rulequest)

Numeric attributes

- Standard method: binary splits
  - E.g. temp < 45
- Unlike nominal attributes,
  every attribute has many possible split points
- Solution is straightforward extension:
  - Evaluate info gain (or other measure)
    for every possible split point of attribute
  - Choose "best" split point
  - Info gain for best split point is info gain for attribute
- Computationally more demanding

Weather data – nominal values

Outlook	Temperature	Humidity	Windy	Play
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Overcast	Hot	High	False	Yes
Rainy	Mild	Normal	False	Yes
…	…	…	…	…

```
If outlook = sunny and humidity = high then play = no
If outlook = rainy and windy = true then play = no
If outlook = overcast then play = yes
If humidity = normal then play = yes
If none of the above then play = yes
```

Weather data - numeric

Outlook	Temperature	Humidity	Windy	Play
Sunny	85	85	False	No
Sunny	80	90	True	No
Overcast	83	86	False	Yes
Rainy	75	80	False	Yes
…	…	…	…	…

```
If outlook = sunny and humidity > 83 then play = no
If outlook = rainy and windy = true then play = no
If outlook = overcast then play = yes
If humidity < 85 then play = yes
If none of the above then play = yes
```

Example

- Split on temperature attribute:

64	65	68	69	70	71	72	72	75	75	80	81	83	85
Yes	No	Yes	Yes	Yes	No	No	Yes	Yes	Yes	No	Yes	Yes	No

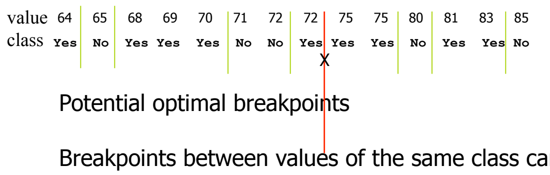
  - E.g. temperature < 71.5: yes/4, no/2
    temperature ≥ 71.5: yes/5, no/3
  - Info([4,2],[5,3])
    = 6/14 info([4,2]) + 8/14 info([5,3])
    = 0.939 bits
- Place split points halfway between values
- Can evaluate all split points in one pass!

Avoid repeated sorting!

- Sort instances by the values of the numeric attribute
  - Time complexity for sorting: $O(n \log n)$
- ***Q. Does this have to be repeated at each node of the tree?***
- A: No! Sort order for children can be derived from sort order for parent
  - Time complexity of derivation: $O(n)$
  - Drawback: need to create and store an array of sorted indices for each numeric attribute

More speeding up

- Entropy only needs to be evaluated between points of different classes (Fayyad & Irani, 1992)



value	64	65	68	69	70	71	72	72	75	75	80	81	83	85
class	Yes	No	Yes	Yes	Yes	No	No	Yes	Yes	Yes	No	Yes	Yes	No

X

Potential optimal breakpoints

Breakpoints between values of the same class cannot be optimal

Binary *vs.* multi-way splits

- Splitting (multi-way) on a nominal attribute exhausts all information in that attribute
  - Nominal attribute is tested (at most) once on any path in the tree
- Not so for binary splits on numeric attributes!
  - Numeric attribute may be tested several times along a path in the tree
- Disadvantage: tree is hard to read
- Remedy:
  - pre-discretize numeric attributes, *or*
  - use multi-way splits instead of binary ones

Missing as a separate value

- Missing value denoted "?" in C4.X
- Simple idea: treat missing as a separate value
- Q: When this is not appropriate?
- A: When values are missing due to different reasons
  - Example 1: gene expression could be missing when it is very high or very low
  - Example 2: field **IsPregnant**=missing for a male patient should be treated differently (no) than for a female patient of age 25 (unknown)

Missing values - advanced

Split instances with missing values into pieces

- A piece going down a branch receives a weight proportional to the popularity of the branch
- weights sum to 1

- Info gain works with fractional instances
  - use sums of weights instead of counts
- During classification, split the instance into pieces in the same way
  - Merge probability distribution using weights

Pruning

- Goal: Prevent overfitting to noise in the data
- Two strategies for "pruning" the decision tree:
  - *Postpruning* - take a fully-grown decision tree and discard unreliable parts
  - *Prepruning* - stop growing a branch when information becomes unreliable
- Postpruning preferred in practice—prepruning can "stop too early"

Prepruning

- Based on statistical significance test
  - Stop growing the tree when there is no *statistically significant* association between any attribute and the class at a particular node
- Most popular test: *chi-squared test*
- ID3 used chi-squared test in addition to information gain
  - Only statistically significant attributes were allowed to be selected by information gain procedure

Early stopping

	a	b	class
1	0	0	0
2	0	1	1
3	1	0	1
4	1	1	0

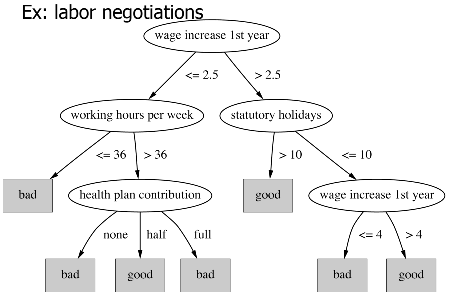
- Pre-pruning may stop the growth process prematurely: *early stopping*
- Classic example: XOR/Parity-problem
  - No *individual* attribute exhibits any significant association to the class
  - Structure is only visible in fully expanded tree
  - Pre-pruning won't expand the root node
- But: XOR-type problems rare in practice
- And: pre-pruning faster than post-pruning

Post-pruning

- First, build full tree
- Then, prune it
  - Fully-grown tree shows all attribute interactions
- Problem: some subtrees might be due to chance effects
- Two pruning operations:
  1. *Subtree replacement*
  2. *Subtree raising*
- Possible strategies:
  - error estimation
  - significance testing
  - MDL principle

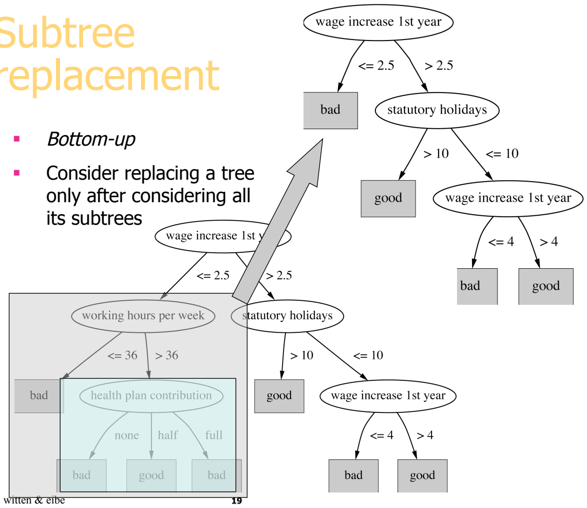
Subtree replacement

- *Bottom-up*
- Consider replacing a tree only after considering all its subtrees
- Ex: labor negotiations



Subtree replacement

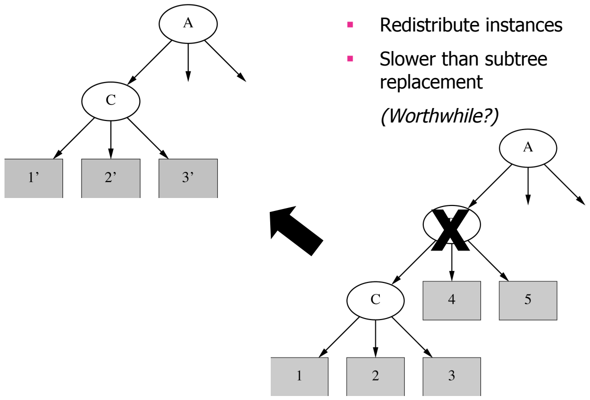
- *Bottom-up*
- Consider replacing a tree only after considering all its subtrees



*Subtree raising

- Delete node
- Redistribute instances
- Slower than subtree replacement

  (Worthwhile?)



Estimating error rates

- Prune only if it reduces the estimated error
- Error on the training data is NOT a useful estimator
  Q: Why it would result in very little pruning?
- Use hold-out set for pruning ("reduced-error pruning")
- C4.5's method
  - Derive confidence interval from training data
  - Use a heuristic limit, derived from this, for pruning
  - Standard Bernoulli-process-based method
  - Shaky statistical assumptions (based on training data)

*Mean and variance

- Mean and variance for a Bernoulli trial: *p, p* (1–*p*)
- Expected success rate *f=S*/*N*
- Mean and variance for *f* : *p, p* (1–*p*)/*N*
- For large enough *N*, *f* follows a Normal distribution
- c% confidence interval [–*z* ≤ *X* ≤ *z*] for random variable with 0 mean is given by:

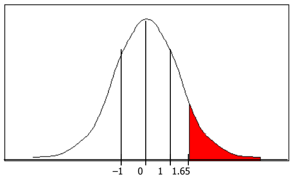
$$\Pr[-z \le X \le z] = c$$

- With a symmetric distribution:

$$\Pr[-z \le X \le z] = 1 - 2 \times \Pr[X \ge z]$$

*Confidence limits

- Confidence limits for the normal distribution with 0 mean and a variance of 1:



Pr[*X* ≥ *z*]	*z*
0.1%	3.09
0.5%	2.58
1%	2.33
5%	1.65
10%	1.28
20%	0.84
25%	0.69
40%	0.25

- Thus:

$$\Pr[-1.65 \le X \le 1.65] = 90\%$$

- To use this we have to reduce our random variable *f* to have 0 mean and unit variance

*Transforming *f*

- Transformed value for *f* : $\frac{f-p}{\sqrt{p(1-p)/N}}$

  (i.e. subtract the mean and divide by the *standard deviation*)

- Resulting equation:

$$\Pr\left[-z \le \frac{f-p}{\sqrt{p(1-p)/N}} \le z\right] = c$$

- Solving for *p*:

$$p = \left(f + \frac{z^2}{2N} \pm z\sqrt{\frac{f}{N} - \frac{f^2}{N} + \frac{z^2}{4N^2}}\right) \Bigg/ \left(1 + \frac{z^2}{N}\right)$$

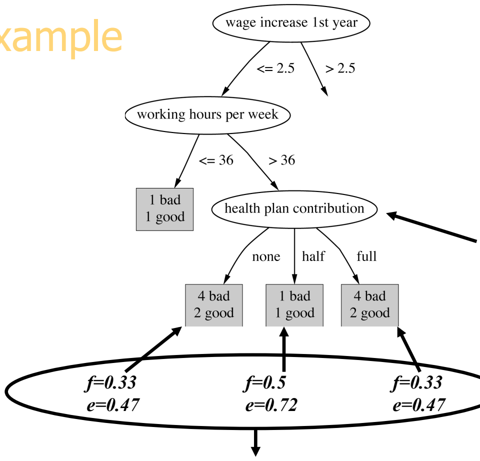
C4.5's method

- Error estimate for subtree is weighted sum of error estimates for all its leaves
- Error estimate for a node (upper bound):

$$e = \left(f + \frac{z^2}{2N} + z\sqrt{\frac{f}{N} - \frac{f^2}{N} + \frac{z^2}{4N^2}} \right) \Bigg/ \left(1 + \frac{z^2}{N} \right)$$

- If *c* = 25% then *z* = 0.69 (from normal distribution)
- *f* is the error on the training data
- *N* is the number of instances covered by the leaf

Example



***f* = 5/14**
***e* = 0.46**
***e* < 0.51**
so prune!

Combined using ratios 6:2:6 gives 0.51

*Complexity of tree induction

- Assume
  - *m* attributes
  - *n* training instances
  - tree depth *O* (log *n*)
- Building a tree *O* (*m n* log *n*)
- Subtree replacement *O* (*n*)
- Subtree raising *O* (*n* (log *n*)^2)
  - Every instance may have to be redistributed at every node between its leaf and the root
  - Cost for redistribution (on average): *O* (log *n*)
- Total cost: *O* (*m n* log *n*) + *O* (*n* (log *n*)^2)

From trees to rules

- Simple way: one rule for each leaf
- C4.5rules: greedily prune conditions from each rule if this reduces its estimated error
  - Can produce duplicate rules
  - Check for this at the end
- Then
  - look at each class in turn
  - consider the rules for that class
  - find a "good" subset (guided by MDL)
- Then rank the subsets to avoid conflicts
- Finally, remove rules (greedily) if this decreases error on the training data

C4.5rules: choices and options

- C4.5rules slow for large and noisy datasets
- Commercial version C5.0rules uses a different technique
  - Much faster and a bit more accurate
- C4.5 has two parameters
  - Confidence value (default 25%): lower values incur heavier pruning
  - Minimum number of instances in the two most popular branches (default 2)

*Classification rules

- Common procedure: *separate-and-conquer*
- Differences:
  - Search method (e.g. greedy, beam search, ...)
  - Test selection criteria (e.g. accuracy, ...)
  - Pruning method (e.g. MDL, hold-out set, ...)
  - Stopping criterion (e.g. minimum accuracy)
  - Post-processing step
- Also: Decision list
  vs. one rule set for each class

*Test selection criteria

- Basic covering algorithm:
  - keep adding conditions to a rule to improve its accuracy
  - Add the condition that improves accuracy the most
- Measure 1: *p*/*t*
  - *t* total instances covered by rule
    p number of these that are positive
  - Produce rules that don't cover *negative* instances, as quickly as possible
  - May produce rules with very small coverage —special cases or noise?
- Measure 2: Information gain *p* (log(*p*/*t*) – log(P/T))
  - *P* and *T* the positive and total numbers before the new condition was added
  - Information gain emphasizes positive rather than negative instances
- These interact with the pruning mechanism used

*Missing values, numeric attributes

- Common treatment of missing values: *for any test, they fail*
  - Algorithm must either
    - use other tests to separate out positive instances
    - leave them uncovered until later in the process
- In some cases it's better to treat "missing" as a separate value
- Numeric attributes are treated just like they are in decision trees

*Pruning rules

- Two main strategies:
  - *Incremental* pruning
  - *Global* pruning
- Other difference: pruning criterion
  - Error on hold-out set (*reduced-error pruning*)
  - Statistical significance
  - MDL principle
- Also: post-pruning vs. pre-pruning

Summary

- Decision Trees
  - splits – binary, multi-way
  - split criteria – entropy, gini, …
  - missing value treatment
  - pruning
  - rule extraction from trees
- No method is always superior – experiment!

Classification Algorithms – Continued

Outline

- **Rules**
- Linear Models (Regression)
- Instance-based (Nearest-neighbor)

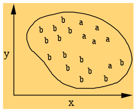
Generating Rules

- Decision tree can be converted into a rule set
- Straightforward conversion:
  - each path to the leaf becomes a rule – makes an overly complex rule set
- More effective conversions are not trivial
  - (e.g. C4.8 tests each node in root-leaf path to see if it can be eliminated without loss in accuracy)

Covering algorithms

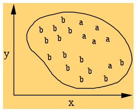
- Strategy for generating a rule set directly: for each class in turn find rule set that covers all instances in it (excluding instances not in the class)
- This approach is called a *covering* approach because at each stage a rule is identified that covers some of the instances

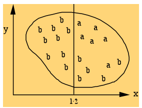
Example: generating a rule



If true then class = a

Example: generating a rule, II

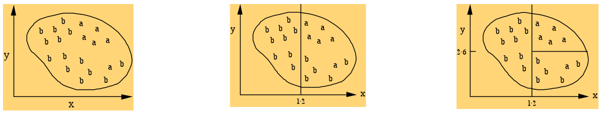




If true then class = a

If x > 1.2 then class = a

Example: generating a rule, III

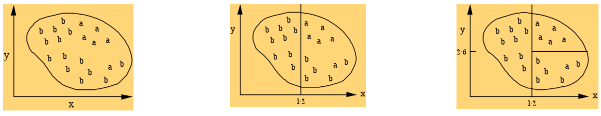


If true then class = a

If x > 1.2 and y > 2.6 then class = a

If x > 1.2 then class = a

Example: generating a rule, IV



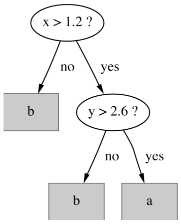
If true then class = a

If x > 1.2 and y > 2.6 then class = a

If x > 1.2 then class = a

- Possible rule set for class “b”:
  If x ≤ 1.2 then class = b
  If x > 1.2 and y ≤ 2.6 then class = b
- More rules could be added for “perfect” rule set

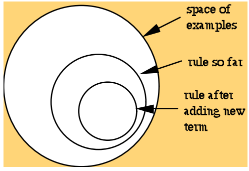
Rules vs. trees



- Corresponding decision tree:
  (produces exactly the same predictions)
- But: rule sets *can* be more clear when decision trees suffer from replicated subtrees
- Also: in multi-class situations, covering algorithm concentrates on one class at a time whereas decision tree learner takes all classes into account

A simple covering algorithm

- Generates a rule by adding tests that maximize rule’s accuracy
- Similar to situation in decision trees: problem of selecting an attribute to split on
  - But: decision tree inducer maximizes overall purity
- Each new test reduces rule’s coverage:



Selecting a test

- Goal: maximize accuracy
  - *t* total number of instances covered by rule
  - *p* positive examples of the class covered by rule
  - *t – p* number of errors made by rule
  - ⇒ Select test that maximizes the ratio *p/t*
- We are finished when *p/t* = 1 or the set of instances can’t be split any further

Example: contact lens data

- Rule we seek:

```
If ?
   then recommendation = hard
```

- Possible tests:

Age = Young	2/8
Age = Pre-presbyopic	1/8
Age = Presbyopic	1/8
Spectacle prescription = Myope	3/12
Spectacle prescription = Hypermetrope	1/12
Astigmatism = no	0/12
Astigmatism = yes	4/12
Tear production rate = Reduced	0/12
Tear production rate = Normal	4/12

Modified rule and resulting data

- Rule with best test added:

```
If astigmatism = yes
   then recommendation = hard
```

- Instances covered by modified rule:

Age	Spectacle prescription	Astigmatism	Tear production rate	Recommended lenses
Young	Myope	Yes	Reduced	None
Young	Myope	Yes	Normal	Hard
Young	Hypermetrope	Yes	Reduced	None
Young	Hypermetrope	Yes	Normal	hard
Pre-presbyopic	Myope	Yes	Reduced	None
Pre-presbyopic	Myope	Yes	Normal	Hard
Pre-presbyopic	Hypermetrope	Yes	Reduced	None
Pre-presbyopic	Hypermetrope	Yes	Normal	None
Presbyopic	Myope	Yes	Reduced	None
Presbyopic	Myope	Yes	Normal	Hard
Presbyopic	Hypermetrope	Yes	Reduced	None
Presbyopic	Hypermetrope	Yes	Normal	None

Further refinement

- Current state:

```
If astigmatism = yes
   and ?
 then recommendation = hard
```

- Possible tests:

Age = Young	2/4
Age = Pre-presbyopic	1/4
Age = Presbyopic	1/4
Spectacle prescription = Myope	3/6
Spectacle prescription = Hypermetrope	1/6
Tear production rate = Reduced	0/6
Tear production rate = Normal	4/6

Modified rule and resulting data

- Rule with best test added:

```
If astigmatism = yes
    and tear production rate = normal
  then recommendation = hard
```

- Instances covered by modified rule:

Age	Spectacle prescription	Astigmatism	Tear production rate	Recommended lenses
Young	Myope	Yes	Normal	Hard
Young	Hypermetrope	Yes	Normal	hard
Pre-presbyopic	Myope	Yes	Normal	Hard
Pre-presbyopic	Hypermetrope	Yes	Normal	None
Presbyopic	Myope	Yes	Normal	Hard
Presbyopic	Hypermetrope	Yes	Normal	None

Further refinement

- Current state:

```
If astigmatism = yes
     and tear production rate = normal
     and ?
   then recommendation = hard
```

- Possible tests:

Age = Young	2/2
Age = Pre-presbyopic	1/2
Age = Presbyopic	1/2
Spectacle prescription = Myope	3/3
Spectacle prescription = Hypermetrope	1/3

- Tie between the first and the fourth test
  - We choose the one with greater coverage

The result

- Final rule:

```
If astigmatism = yes
   and tear production rate = normal
   and spectacle prescription = myope
   then recommendation = hard
```

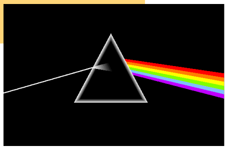
- Second rule for recommending "hard lenses":
(built from instances not covered by first rule)

```
If age = young and astigmatism = yes
   and tear production rate = normal
   then recommendation = hard
```

- These two rules cover all "hard lenses":
  - Process is repeated with other two classes

Pseudo-code for PRISM

```
For each class C
 Initialize E to the instance set
 While E contains instances in class C
  Create a rule R with an empty left-hand side that predicts class C
  Until R is perfect (or there are no more attributes to use) do
   For each attribute A not mentioned in R, and each value v,
    Consider adding the condition A = v to the left-hand side of R
    Select A and v to maximize the accuracy p/t
     (break ties by choosing the condition with the largest p)
   Add A = v to R
  Remove the instances covered by R from E
```



Rules vs. decision lists

- PRISM with outer loop removed generates a decision list for one class
  - Subsequent rules are designed for rules that are not covered by previous rules
  - But: order doesn't matter because all rules predict the same class
- Outer loop considers all classes separately
  - No order dependence implied
- Problems: overlapping rules, default rule required

Separate and conquer

- Methods like PRISM (for dealing with one class) are *separate-and-conquer* algorithms:
  - First, a rule is identified
  - Then, all instances covered by the rule are separated out
  - Finally, the remaining instances are "conquered"
- Difference to divide-and-conquer methods:
  - Subset covered by rule doesn't need to be explored any further

Outline

- Rules
- **Linear Models (Regression)**
- Instance-based (Nearest-neighbor)

Linear models

- Work most naturally with numeric attributes
- Standard technique for numeric prediction: linear regression
  - Outcome is linear combination of attributes

$$x = w_0 + w_1 a_1 + w_2 a_2 + \ldots + w_k a_k$$

- Weights are calculated from the training data
- Predicted value for first training instance $\mathbf{a}^{(1)}$

$$w_0 a_0^{(1)} + w_1 a_1^{(1)} + w_2 a_2^{(1)} + \ldots + w_k a_k^{(1)} = \sum_{j=0}^{k} w_j a_j^{(1)}$$

Minimizing the squared error

- Choose k +1 coefficients to minimize the squared error on the training data
- Squared error: $$\sum_{i=1}^{n} \left(x^{(i)} - \sum_{j=0}^{k} w_j a_j^{(i)} \right)^2$$
- Derive coefficients using standard matrix operations
- Can be done if there are more instances than attributes (roughly speaking)
- Minimizing the *absolute error* is more difficult

Regression for Classification

- *Any* regression technique can be used for classification
  - Training: perform a regression for each class, setting the output to 1 for training instances that belong to class, and 0 for those that don't
  - Prediction: predict class corresponding to model with largest output value (*membership value*)
- For linear regression this is known as *multi-response linear regression*

*Theoretical justification

Observed target value (either 0 or 1)

Model ***Instance*** ***The scheme minimizes this***

$$E_y\{(f(X)-Y)^2 \mid X=x\}$$

True class probability

$$= E_y\{(f(X)-P(Y=1\mid X=x)+P(Y=1\mid X=x)-Y)^2 \mid X=x\}$$

$$= (f(x)-P(Y=1\mid X=x))^2 + 2\times(f(x)-P(Y=1\mid X=x))\times$$
$$E_y\{P(Y=1\mid X=x)-Y \mid X=x\} + E_y\{(P(Y=1\mid X=x)-Y)^2 \mid X=x\}$$

$$= (f(x)-P(Y=1\mid X=x))^2 + 2\times(f(x)-P(Y=1\mid X=x))\times$$
$$(P(Y=1\mid X=x)-E_y\{Y \mid X=x\}) + E_y\{(P(Y=1\mid X=x)-Y)^2 \mid X=x\}$$

$$= \underbrace{(f(x)-P(Y=1\mid X=x))^2}_{\textit{We want to minimize this}} + \underbrace{E_y\{(P(Y=1\mid X=x)-Y)^2 \mid X=x\}}_{\textit{Constant}}$$

*Pairwise regression

- Another way of using regression for classification:
  - A regression function for every *pair* of classes, using only instances from these two classes
  - Assign output of +1 to one member of the pair, –1 to the other
- Prediction is done by voting
  - Class that receives most votes is predicted
  - Alternative: "don't know" if there is no agreement
- More likely to be accurate but more expensive

Logistic regression

- Problem: some assumptions violated when linear regression is applied to classification problems
- *Logistic* regression: alternative to linear regression
  - Designed for classification problems
  - Tries to estimate class probabilities directly
    - Does this using the *maximum likelihood* method
  - Uses this linear model:

$$\log\left(\frac{P}{1-P}\right) = w_0a_0 + w_1a_1 + w_2a_2 + \ldots + w_ka_k$$

P= Class probability

Discussion of linear models

- Not appropriate if data exhibits non-linear dependencies
- But: can serve as building blocks for more complex schemes (i.e. model trees)
- Example: multi-response linear regression defines a *hyperplane* for any two given classes:

$$(w_0^{(1)}-w_0^{(2)})a_0 + (w_1^{(1)}-w_1^{(2)})a_1 + (w_2^{(1)}-w_2^{(2)})a_2 + \ldots + (w_k^{(1)}-w_k^{(2)})a_k > 0$$

Comments on basic methods

- Minsky and Papert (1969) showed that linear classifiers have limitations, e.g. can't learn XOR
  - But: combinations of them can (→ Neural Nets)

Outline

- Rules
- Linear Models (Regression)
- **Instance-based (Nearest-neighbor)**

Instance-based representation

- Simplest form of learning: *rote learning*
  - Training instances are searched for instance that most closely resembles new instance
  - The instances themselves represent the knowledge
  - Also called *instance-based* learning
- Similarity function defines what's "learned"
- Instance-based learning is *lazy* learning
- Methods:
  - *nearest-neighbor*
  - *k-nearest-neighbor*
  - …

The distance function

- Simplest case: one numeric attribute
  - Distance is the difference between the two attribute values involved (or a function thereof)
- Several numeric attributes: normally, Euclidean distance is used and attributes are normalized
- Nominal attributes: distance is set to 1 if values are different, 0 if they are equal
- Are all attributes equally important?
  - Weighting the attributes might be necessary

Instance-based learning

- Distance function defines what's learned
- Most instance-based schemes use *Euclidean distance*:

$$\sqrt{(a_1^{(1)} - a_1^{(2)})^2 + (a_2^{(1)} - a_2^{(2)})^2 + \ldots + (a_k^{(1)} - a_k^{(2)})^2}$$

a$^{(1)}$ and **a**$^{(2)}$: two instances with *k* attributes

- Taking the square root is not required when comparing distances
- Other popular metric: *city-block (Manhattan) metric*
  - Adds differences without squaring them

Normalization and other issues

- Different attributes are measured on different scales ⇒ need to be *normalized*:

$$a_i = \frac{v_i - \min v_i}{\max v_i - \min v_i} \qquad \text{or} \qquad a_i = \frac{v_i - Avg(v_i)}{StDev(v_i)}$$

v_i: the actual value of attribute *i*

- Nominal attributes: distance either 0 or 1
- Common policy for missing values: assumed to be maximally distant (given normalized attributes)

Discussion of 1-NN

- Often very accurate
- … but slow:
  - simple version scans entire training data to derive a prediction
- Assumes all attributes are equally important
  - Remedy: attribute selection or weights
- Possible remedies against noisy instances:
  - Take a majority vote over the *k* nearest neighbors
  - Removing noisy instances from dataset (difficult!)
- Statisticians have used *k*-NN since early 1950s
  - If $n \to \infty$ and $k/n \to 0$, error approaches minimum

Summary

- Simple methods frequently work well
  - robust against noise, errors
- Advanced methods, if properly used, can improve on simple methods
- No method is universally best

Evaluation and Credibility

How much should we believe in what was learned?

Outline

- Introduction
- Classification with Train, Test, and Validation sets
  - Handling Unbalanced Data; Parameter Tuning
- Cross-validation
- Comparing Data Mining Schemes

Introduction

- How predictive is the model we learned?
- Error on the training data is *not* a good indicator of performance on future data
  - ***Q: Why?***
  - A: Because new data will probably not be **exactly** the same as the training data!
- Overfitting – fitting the training data too precisely - usually leads to poor results on new data

Evaluation issues

- Possible evaluation measures:
  - Classification Accuracy
  - Total cost/benefit – when different errors involve different costs
  - Lift and ROC curves
  - Error in numeric predictions
- How reliable are the predicted results ?

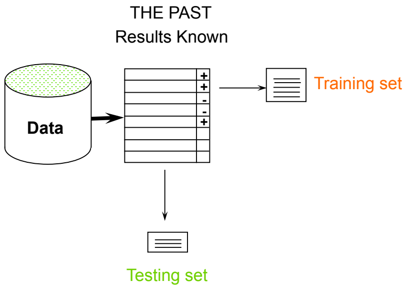
Classifier error rate

- Natural performance measure for classification problems: *error rate*
  - *Success*: instance's class is predicted correctly
  - *Error*: instance's class is predicted incorrectly
  - Error rate: proportion of errors made over the whole set of instances
- *Training set error rate:* is way too optimistic!
  - you can find patterns even in random data

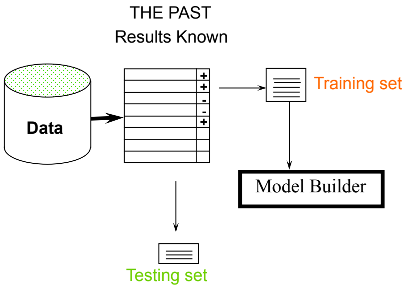
Evaluation on "LARGE" data

- If many (thousands) of examples are available, including several hundred examples from each class, then a simple evaluation is sufficient
  - Randomly split data into training and test sets (usually 2/3 for train, 1/3 for test)
- Build a classifier using the *train* set and evaluate it using the *test* set.

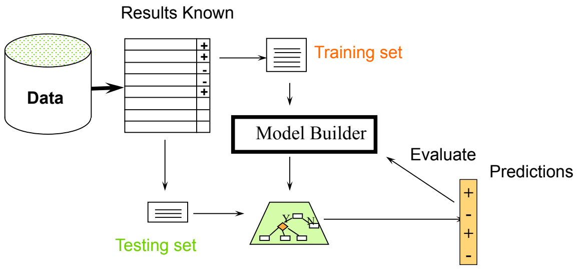
Classification Step 1: Split data into train and test sets



Classification Step 2: Build a model on a training set



Classification Step 3: Evaluate on test set (Re-train?)



Handling unbalanced data

- Sometimes, classes have very unequal frequency
  - Attrition prediction: 97% stay, 3% attrite (in a month)
  - medical diagnosis: 90% healthy, 10% disease
  - eCommerce: 99% don't buy, 1% buy
  - Security: >99.99% of Americans are not terrorists
- Similar situation with multiple classes
- Majority class classifier can be 97% correct, but useless

Balancing unbalanced data

- With two classes, a good approach is to build **BALANCED** train and test sets, and train model on a balanced set
  - randomly select desired number of minority class instances
  - add equal number of randomly selected majority class
- Generalize "balancing" to multiple classes
  - Ensure that each class is represented with approximately equal proportions in train and test

A note on parameter tuning

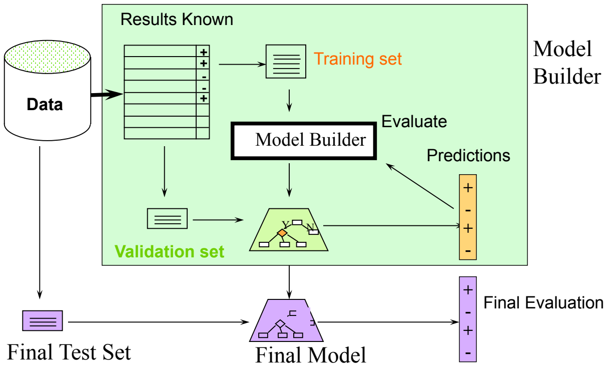
- It is important that the test data is not used *in any way* to create the classifier
- Some learning schemes operate in two stages:
  - Stage 1: builds the basic structure
  - Stage 2: optimizes parameter settings
- The test data can't be used for parameter tuning!
- Proper procedure uses three sets: **training data, validation data, and test data**
  - Validation data is used to optimize parameters

witten & eibe

Making the most of the data

- Once evaluation is complete, *all the data* can be used to build the final classifier
- Generally, the larger the training data the better the classifier (but returns diminish)
- The larger the test data the more accurate the error estimate

Classification: Train, Validation, Test split



*Predicting performance

- Assume the estimated error rate is 25%. How close is this to the true error rate?
  - Depends on the amount of test data
- Prediction is just like tossing a biased (!) coin
  - "Head" is a "success", "tail" is an "error"
- In statistics, a succession of independent events like this is called a Bernoulli process
- Statistical theory provides us with confidence intervals for the true underlying proportion!

*Confidence intervals

- We can say: *p* lies within a certain specified interval with a certain specified confidence
- Example: *S*=750 successes in *N*=1000 trials
  - Estimated success rate: 75%
  - How close is this to true success rate *p*?
    - Answer: with 80% confidence $p \in [73.2,76.7]$
- Another example: *S*=75 and *N*=100
  - Estimated success rate: 75%
  - With 80% confidence $p \in [69.1,80.1]$

*Mean and variance (also Mod 7)

- Mean and variance for a Bernoulli trial: *p, p* (1–*p*)
- Expected success rate *f*=*S*/*N*
- Mean and variance for *f* : *p, p* (1–*p*)/*N*
- For large enough *N*, *f* follows a Normal distribution
- c% confidence interval [–*z* ≤ *X* ≤ *z*] for random variable with 0 mean is given by:

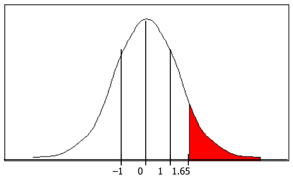
$$\Pr[-z \le X \le z] = c$$

- With a symmetric distribution:

$$\Pr[-z \le X \le z] = 1 - 2 \times \Pr[X \ge z]$$

*Confidence limits

- Confidence limits for the normal distribution with 0 mean and a variance of 1:



$\Pr[X \ge z]$	z
0.1%	3.09
0.5%	2.58
1%	2.33
5%	1.65
10%	1.28
20%	0.84
40%	0.25

- Thus:

$$\Pr[-1.65 \le X \le 1.65] = 90\%$$

- To use this we have to reduce our random variable *f* to have 0 mean and unit variance

*Transforming *f*

- Transformed value for *f* : $\dfrac{f-p}{\sqrt{p(1-p)/N}}$

  (i.e. subtract the mean and divide by the *standard deviation*)

- Resulting equation: $$\Pr\left[-z \le \frac{f-p}{\sqrt{p(1-p)/N}} \le z\right] = c$$

- Solving for *p* :

$$p = \left(f + \frac{z^2}{2N} \pm z\sqrt{\frac{f}{N} - \frac{f^2}{N} + \frac{z^2}{4N^2}}\right) \Big/ \left(1 + \frac{z^2}{N}\right)$$

*Examples

- *f* = 75%, *N* = 1000, *c* = 80% (so that *z* = 1.28): $p \in [0.732, 0.767]$
- *f* = 75%, *N* = 100, *c* = 80% (so that *z* = 1.28): $p \in [0.691, 0.801]$
- Note that normal distribution assumption is only valid for large *N* (i.e. *N* > 100)
- *f* = 75%, *N* = 10, *c* = 80% (so that *z* = 1.28): $p \in [0.549, 0.881]$

  (should be taken with a grain of salt)

Evaluation on “small” data

- The *holdout* method reserves a certain amount for testing and uses the remainder for training
  - Usually: one third for testing, the rest for training
- For small or “unbalanced” datasets, samples might not be representative
  - Few or none instances of some classes
- *Stratified sample: advanced version of balancing the data*
  - Make sure that each class is represented with approximately equal proportions in both subsets

Repeated holdout method

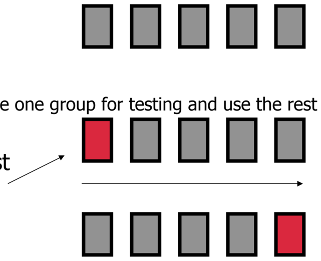
- Holdout estimate can be made more reliable by repeating the process with different subsamples
  - In each iteration, a certain proportion is randomly selected for training (possibly with stratification)
  - The error rates on the different iterations are averaged to yield an overall error rate
- This is called the *repeated holdout* method
- Still not optimum: the different test sets overlap
  - Can we prevent overlapping?

Cross-validation

- *Cross-validation* avoids overlapping test sets
  - First step: data is split into *k* subsets of equal size
  - Second step: each subset in turn is used for testing and the remainder for training
- This is called *k-fold cross-validation*
- Often the subsets are stratified before the cross-validation is performed
- The error estimates are averaged to yield an overall error estimate

Cross-validation example:

- Break up data into groups of the same size
- Hold aside one group for testing and use the rest to build model
- Repeat



More on cross-validation

- Standard method for evaluation: stratified ten-fold cross-validation
- Why ten? Extensive experiments have shown that this is the best choice to get an accurate estimate
- Stratification reduces the estimate's variance
- Even better: repeated stratified cross-validation
  - E.g. ten-fold cross-validation is repeated ten times and results are averaged (reduces the variance)

witten & eibe

Leave-One-Out cross-validation

- Leave-One-Out:
  a particular form of cross-validation:
  - Set number of folds to number of training instances
  - I.e., for *n* training instances, build classifier *n* times
- Makes best use of the data
- Involves no random subsampling
- Very computationally expensive
  - (exception: NN)

Leave-One-Out-CV and stratification

- Disadvantage of Leave-One-Out-CV: stratification is not possible
  - It *guarantees* a non-stratified sample because there is only one instance in the test set!
- Extreme example: random dataset split equally into two classes
  - Best inducer predicts majority class
  - 50% accuracy on fresh data
  - Leave-One-Out-CV estimate is 100% error!

*The bootstrap

- CV uses sampling *without replacement*
  - The same instance, once selected, can not be selected again for a particular training/test set
- The *bootstrap* uses sampling *with replacement* to form the training set
  - Sample a dataset of *n* instances *n* times *with replacement* to form a new dataset of *n* instances
  - Use this data as the training set
  - Use the instances from the original dataset that don't occur in the new training set for testing

*The 0.632 bootstrap

- Also called the *0.632 bootstrap*
  - A particular instance has a probability of 1–1/*n* of *not* being picked
  - Thus its probability of ending up in the test data is:

$$\left(1-\frac{1}{n}\right)^{n} \approx e^{-1} = 0.368$$

  - This means the training data will contain approximately 63.2% of the instances

*Estimating error with the bootstrap

- The error estimate on the test data will be very pessimistic
  - Trained on just ~63% of the instances
- Therefore, combine it with the resubstitution error:

$$err = 0.632 \cdot e_{\text{test instances}} + 0.368 \cdot e_{\text{training instances}}$$

- The resubstitution error gets less weight than the error on the test data
- Repeat process several times with different replacement samples; average the results

*More on the bootstrap

- Probably the best way of estimating performance for very small datasets
- However, it has some problems
  - Consider the random dataset from above
  - A perfect memorizer will achieve
    0% resubstitution error and
    ~50% error on test data
  - Bootstrap estimate for this classifier:
    $$err = 0.632 \cdot 50\% + 0.368 \cdot 0\% = 31.6\%$$
  - True expected error: 50%

Comparing data mining schemes

- Frequent situation: we want to know which one of two learning schemes performs better
- Note: this is domain dependent!
- Obvious way: compare 10-fold CV estimates
- Problem: variance in estimate
- Variance can be reduced using repeated CV
- However, we still don't know whether the results are reliable

Significance tests

- Significance tests tell us how confident we can be that there really is a difference
- *Null hypothesis*: there is no "real" difference
- *Alternative hypothesis*: there is a difference
- A significance test measures how much evidence there is in favor of rejecting the null hypothesis
- Let's say we are using 10 times 10-fold CV
- Then we want to know whether the two means of the 10 CV estimates are significantly different
  - *Student's paired t-test* tells us whether the means of two samples are significantly different

*Paired t-test

- *Student's t-test* tells whether the means of two samples are significantly different
- Take individual samples from the set of all possible cross-validation estimates
- Use a *paired* t-test because the individual samples are paired
  - The same CV is applied twice

William Gosset

Born: 1876 in Canterbury; Died: 1937 in Beaconsfield, England

Obtained a post as a chemist in the Guinness brewery in Dublin in 1899. Invented the t-test to handle small samples for quality control in brewing. Wrote under the name "Student".

*Distribution of the means

- $x_1\ x_2 \ldots x_k$ and $y_1\ y_2 \ldots y_k$ are the 2*k* samples for a *k*-fold CV
- m_x and m_y are the means
- With enough samples, the mean of a set of independent samples is normally distributed
- Estimated variances of the means are σ_x^2/k and σ_y^2/k $\quad \dfrac{m_x - \mu}{\sqrt{\sigma_x^2 / k}}$
- If μ_x and μ_y are the true means then

  are *approximately* normally distributed with mean 0, variance 1 $\quad \dfrac{m_x - \mu_x}{\sqrt{\sigma_x^2 / k}} \quad \dfrac{m_y - \mu_y}{\sqrt{\sigma_y^2 / k}}$

*Student's distribution

- With small samples ($k < 100$) the mean follows *Student's distribution with k*–1 *degrees of freedom*
- Confidence limits:

9 degrees of freedom

Pr[*X* ≥ *z*]	*z*
0.1%	4.30
0.5%	3.25
1%	2.82
5%	1.83
10%	1.38
20%	0.88

normal distribution

Pr[*X* ≥ *z*]	*z*
0.1%	3.09
0.5%	2.58
1%	2.33
5%	1.65
10%	1.28
20%	0.84

*Distribution of the differences

- Let $m_d = m_x - m_y$
- The difference of the means (m_d) also has a Student's distribution with k–1 degrees of freedom
- Let σ_d^2 be the variance of the difference
- The standardized version of m_d is called the *t*-statistic:

$$t = \frac{m_d}{\sqrt{\sigma_d^2 / k}}$$

- We use t to perform the *t*-test

*Performing the test

1. Fix a significance level α
   - If a difference is significant at the α% level, there is a (100-α)% chance that there really is a difference
3. Divide the significance level by two because the test is two-tailed
   - I.e. the true difference can be +ve or – ve
5. Look up the value for z that corresponds to $\alpha/2$
7. If $t \leq -z$ or $t \geq z$ then the difference is significant
   - I.e. the null hypothesis can be rejected

Unpaired observations

- If the CV estimates are from different randomizations, they are no longer paired
- (or maybe we used k -fold CV for one scheme, and j -fold CV for the other one)
- Then we have to use an *un* paired t-test with min(k , j) – 1 degrees of freedom
- The *t*-statistic becomes:

$$t = \frac{m_d}{\sqrt{\sigma_d^2 / k}} \Rightarrow t = \frac{m_x - m_y}{\sqrt{\frac{\sigma_x^2}{k} + \frac{\sigma_y^2}{j}}}$$

*Interpreting the result

- All our cross-validation estimates are based on the same dataset
- Hence the test only tells us whether a *complete k*-fold CV for this dataset would show a difference
  - Complete *k*-fold CV generates all possible partitions of the data into *k* folds and averages the results
- Ideally, should use a different dataset sample for each of the *k*-fold CV estimates used in the test to judge performance across different training sets

*Predicting probabilities

- Performance measure so far: success rate
- Also called *0-1 loss function*:

$$\sum_i \begin{cases} 0 \text{ if prediction is correct} \\ 1 \text{ if prediction is incorrect} \end{cases}$$

- Most classifiers produces class probabilities
- Depending on the application, we might want to check the accuracy of the probability estimates
- 0-1 loss is not the right thing to use in those cases

*Quadratic loss function

- $p_1 \ldots p_k$ are probability estimates for an instance
- c is the index of the instance's actual class
- $a_1 \ldots a_k = 0$, except for a_c which is 1
- *Quadratic loss* is: $\sum_j (p_j - a_j)^2 = \sum_{j \neq c} p_j^2 + (1 - p_c)^2$
- Want to minimize $E\left[\sum_j (p_j - a_j)^2\right]$
- Can show that this is minimized when $p_j = p_j^*$, the true probabilities

*Informational loss function

- The informational loss function is –log(p_c), where c is the index of the instance's actual class
- Number of bits required to communicate the actual class
- Let p_1^* … p_k^* be the true class probabilities
- Then the expected value for the loss function is:

$$-p_1^* \log_2 p_1 - \ldots - p_k^* \log_2 p_k$$

- Justification: minimized when $p_j = p_j^*$
- Difficulty: *zero-frequency problem*

*Discussion

- Which loss function to choose?
  - Both encourage honesty
  - Quadratic loss function takes into account all class probability estimates for an instance
  - Informational loss focuses only on the probability estimate for the actual class
  - Quadratic loss is bounded: *it can never exceed 2* $1 + \sum_j p_j^2$
  - Informational loss can be infinite
- Informational loss is related to *MDL principle* [later]

Evaluation Summary:

- Use Train, Test, Validation sets for "LARGE" data
- Balance "un-balanced" data
- Use Cross-validation for small data
- Don't use test data for parameter tuning - use separate validation data

- Most Important: Avoid Overfitting

Evaluation – next steps

Lift and Costs

Outline

- Lift and Gains charts
- *ROC
- Cost-sensitive learning
- Evaluation for numeric predictions
- MDL principle and Occam's razor

Direct Marketing Paradigm

- Find most likely prospects to contact
- Not everybody needs to be contacted
- Number of targets is usually much smaller than number of prospects

- Typical Applications
  - retailers, catalogues, direct mail (and e-mail)
  - customer acquisition, cross-sell, attrition prediction
  - ...

Direct Marketing Evaluation

- **Accuracy on the entire dataset is not the right measure**
- Approach
  - develop a target model
  - score all prospects and rank them by decreasing score
  - select top P% of prospects for action
- How to decide what is the best selection?

Model-Sorted List

Use a model to assign score to each customer
Sort customers by decreasing score
Expect more targets (hits) near the top of the list

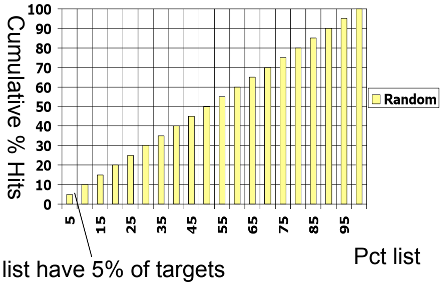
No	Score	Target	CustID	Age	
1	0.97	Y	1746	…	
2	0.95	N	1024	…	
3	0.94	Y	2478	…	
4	0.93	Y	3820	…	
5	0.92	N	4897	…	
…	…		…	…	
99	0.11	N	2734	…	
100	0.06	N	2422		

3 hits in top 5% of the list

If there 15 targets overall, then top 5 has 3/15=20% of targets

CPH (Cumulative Pct Hits)

Definition:
CPH(P,M)
= % of all targets in the first P% of the list scored by model M
CPH frequently called Gains

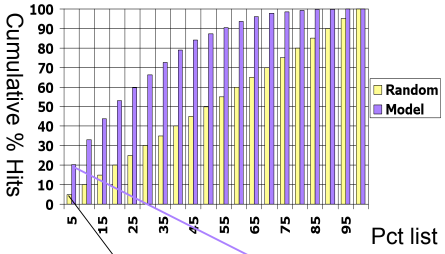


5% of random list have 5% of targets

Q: What is expected value for CPH(P,Random) ?

A: Expected value for CPH(P,Random) = P

CPH: Random List vs Model-ranked list



5% of random list have 5% of targets,

but 5% of model ranked list have 21% of targets CPH(5%,model)=21%.

Lift

Lift(*P*,*M*) = CPH(*P*,*M*) / *P*

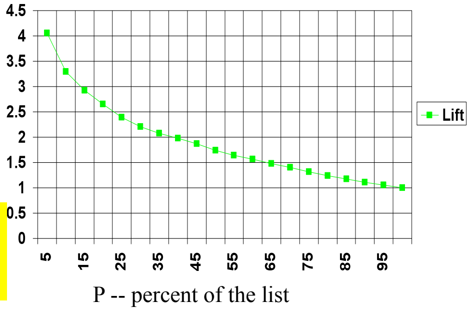
Lift (at 5%)

= 21% / 5%

= 4.2

better than random

Note: Some (including Witten & Eibe) use "Lift" for what we call CPH.



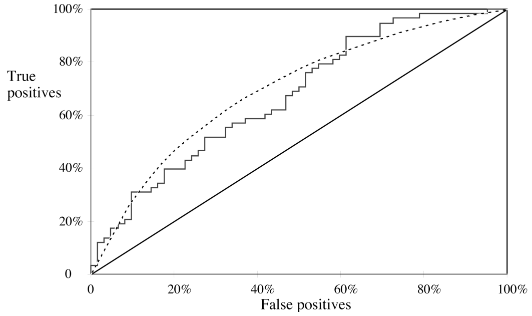
Lift Properties

- ***Q: Lift(P,Random) =***
  - ***A:*** 1 (expected value, can vary)
- ***Q: Lift(100%, M) =***
  - ***A:*** 1 (for any model M)
- ***Q: Can lift be less than 1?***
  - ***A***: yes, if the model is inverted (all the non-targets precede targets in the list)
- Generally, a better model has higher lift

*ROC curves

- *ROC curves* are similar to gains charts
  - Stands for "receiver operating characteristic"
  - Used in signal detection to show tradeoff between hit rate and false alarm rate over noisy channel
- Differences from gains chart:
  - *y* axis shows percentage of true positives in sample *rather than absolute number*
  - *x* axis shows percentage of false positives in sample *rather than sample size*

*A sample ROC curve

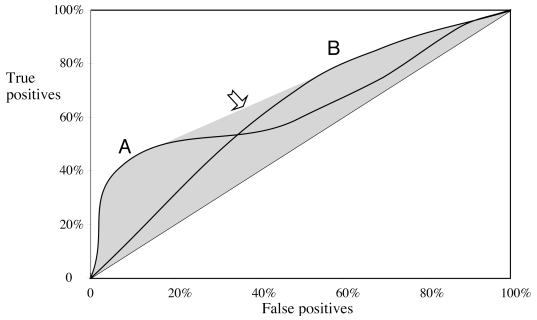


- Jagged curve—one set of test data
- Smooth curve—use cross-validation

*Cross-validation and ROC curves

- Simple method of getting a ROC curve using cross-validation:
  - Collect probabilities for instances in test folds
  - Sort instances according to probabilities
- This method is implemented in WEKA
- However, this is just one possibility
  - The method described in the book generates an ROC curve for each fold and averages them

*ROC curves for two schemes



- For a small, focused sample, use method A
- For a larger one, use method B
- In between, choose between A and B with appropriate probabilities

*The convex hull

- Given two learning schemes we can achieve any point on the convex hull!
- TP and FP rates for scheme 1: t_1 and f_1
- TP and FP rates for scheme 2: t_2 and f_2
- If scheme 1 is used to predict 100×q % of the cases and scheme 2 for the rest, then
  - TP rate for combined scheme: $q \times t_1+(1-q) \times t_2$
  - FP rate for combined scheme: $q \times f_2+(1-q) \times f_2$

Cost Sensitive Learning

- There are two types of errors

		Predicted class	
		Yes	No
Actual class	Yes	TP: True positive	FN: False negative
	No	FP: False positive	TN: True negative

- Machine Learning methods usually minimize FP+FN
- Direct marketing maximizes TP

Different Costs

- In practice, true positive and false negative errors often incur different costs
- Examples:
  - Medical diagnostic tests: does X have leukemia?
  - Loan decisions: approve mortgage for X?
  - Web mining: will X click on this link?
  - Promotional mailing: will X buy the product?
  - …

Cost-sensitive learning

- Most learning schemes do not perform cost-sensitive learning
  - They generate the same classifier no matter what costs are assigned to the different classes
  - Example: standard decision tree learner
- Simple methods for cost-sensitive learning:
  - Re-sampling of instances according to costs
  - Weighting of instances according to costs
- Some schemes are inherently cost-sensitive, e.g. naïve Bayes

*Measures in information retrieval

- Percentage of retrieved documents that are relevant: *precision*=TP/(TP+FP)
- Percentage of relevant documents that are returned: *recall* =TP/(TP+FN)
- Precision/recall curves have hyperbolic shape
- Summary measures: average precision at 20%, 50% and 80% recall (*three-point average recall*)
- *F-measure*=(2×recall×precision)/(recall+precision)

*Summary of measures

	Domain	Plot	Explanation
Lift chart	Marketing	TP Subset size	TP (TP+FP)/(TP+FP+TN+FN)
ROC curve	Communications	TP rate FP rate	TP/(TP+FN) FP/(FP+TN)
Recall-precision curve	Information retrieval	Recall Precision	TP/(TP+FN) TP/(TP+FP)

Evaluating numeric prediction

- Same strategies: independent test set, cross-validation, significance tests, etc.
- Difference: error measures
- Actual target values: $a_1\ a_2 \ldots a_n$
- Predicted target values: $p_1\ p_2 \ldots\ p_n$
- Most popular measure: *mean-squared error*

$$\frac{(p_1 - a_1)^2 + \ldots + (p_n - a_n)^2}{n}$$

  - Easy to manipulate mathematically

Other measures

- The *root mean-squared error* :

$$\sqrt{\frac{(p_1 - a_1)^2 + \ldots + (p_n - a_n)^2}{n}}$$

- The *mean absolute error* is less sensitive to outliers than the mean-squared error:

$$\frac{|p_1 - a_1| + \ldots + |p_n - a_n|}{n}$$

- Sometimes *relative* error values are more appropriate (e.g. 10% for an error of 50 when predicting 500)

Improvement on the mean

- How much does the scheme improve on simply predicting the average?
- The *relative squared error* is ($\bar{a}$ is the average):

$$\frac{(p_1 - a_1)^2 + \ldots + (p_n - a_n)^2}{(\bar{a} - a_1)^2 + \ldots + (\bar{a} - a_n)^2}$$

- The *relative absolute error* is:

$$\frac{|p_1 - a_1| + \ldots + |p_n - a_n|}{|\bar{a} - a_1| + \ldots + |\bar{a} - a_n|}$$

Correlation coefficient

- Measures the *statistical correlation* between the predicted values and the actual values

$$\frac{S_{PA}}{\sqrt{S_P S_A}}$$

$$S_{PA} = \frac{\sum_i (p_i - \bar{p})(a_i - \bar{a})}{n-1} \qquad S_P = \frac{\sum_i (p_i - \bar{p})^2}{n-1} \qquad S_A = \frac{\sum_i (a_i - \bar{a})^2}{n-1}$$

- Scale independent, between –1 and +1
- Good performance leads to large values!

Which measure?

- Best to look at all of them
- Often it doesn't matter
- Example:

	A	B	C	D
Root mean-squared error	67.8	91.7	63.3	57.4
Mean absolute error	41.3	38.5	33.4	29.2
Root rel squared error	42.2%	57.2%	39.4%	35.8%
Relative absolute error	43.1%	40.1%	34.8%	30.4%
Correlation coefficient	0.88	0.88	0.89	0.91

- D best
- C second-best
- A, B arguable

*The MDL principle

- MDL stands for *minimum description length*
- The description length is defined as:

space required to describe a theory

+

space required to describe the theory's mistakes

- In our case the theory is the classifier and the mistakes are the errors on the training data
- Aim: we seek a classifier with minimal DL
- MDL principle is a *model selection criterion*

Model selection criteria

- Model selection criteria attempt to find a good compromise between:
  A. The complexity of a model
  B. Its prediction accuracy on the training data
- Reasoning: a good model is a simple model that achieves high accuracy on the given data
- Also known as *Occam's Razor* : the best theory is the smallest one that describes all the facts

William of Ockham, born in the village of Ockham in Surrey (England) about 1285, was the most influential philosopher of the 14th century and a controversial theologian.

Elegance vs. errors

- Theory 1: very simple, elegant theory that explains the data almost perfectly
- Theory 2: significantly more complex theory that reproduces the data without mistakes
- Theory 1 is probably preferable
- Classical example: Kepler's three laws on planetary motion
  - Less accurate than Copernicus's latest refinement of the Ptolemaic theory of epicycles

*MDL and compression

- MDL principle relates to data compression:
  - The best theory is the one that compresses the data the most
  - I.e. to compress a dataset we generate a model and then store the model and its mistakes
- We need to compute
  (a) size of the model, and
  (b) space needed to encode the errors
- (b) easy: use the informational loss function
- (a) need a method to encode the model

*MDL and Bayes's theorem

- L[T]="length" of the theory
- L[E|T]=training set encoded wrt the theory
- Description length= L[T] + L[E|T]
- Bayes' theorem gives *a posteriori* probability of a theory given the data:

$$\Pr[T \mid E] = \frac{\Pr[E \mid T]\Pr[T]}{\Pr[E]}$$

- Equivalent to:

$$-\log \Pr[T \mid E] = -\log \Pr[E \mid T] - \log \Pr[T] + \underbrace{\log \Pr[E]}_{\textbf{\textit{constant}}}$$

*MDL and MAP

- MAP stands for *maximum a posteriori probability*
- Finding the MAP theory corresponds to finding the MDL theory
- Difficult bit in applying the MAP principle: determining the prior probability Pr[T] of the theory
- Corresponds to difficult part in applying the MDL principle: coding scheme for the theory
- I.e. if we know a priori that a particular theory is more likely we need less bits to encode it

*Discussion of MDL principle

- Advantage: makes full use of the training data when selecting a model
- Disadvantage 1: appropriate coding scheme/prior probabilities for theories are crucial
- Disadvantage 2: no guarantee that the MDL theory is the one which minimizes the expected error
- Note: Occam's Razor is an axiom!
- Epicurus' *principle of multiple explanations*: keep all theories that are consistent with the data

*Bayesian model averaging

- Reflects Epicurus' principle: all theories are used for prediction weighted according to P[T|E]
- Let I be a new instance whose class we must predict
- Let C be the random variable denoting the class
- Then BMA gives the probability of C given
  - I
  - training data E
  - possible theories T_j

$$\Pr[C \mid I, E] = \sum_j \Pr[C \mid I, T_j] \Pr[T_j \mid E]$$

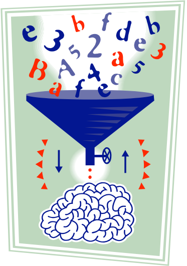
*MDL and clustering

- Description length of theory:
  bits needed to encode the clusters
  - e.g. cluster centers
- Description length of data given theory:
  encode cluster membership and position relative to cluster
  - e.g. distance to cluster center
- Works if coding scheme uses less code space for small numbers than for large ones
- With nominal attributes, must communicate probability distributions for each cluster

Evaluation Summary:

- Avoid Overfitting
- Use Cross-validation for small data
- Don't use test data for parameter tuning - use separate validation data
- Consider costs when appropriate

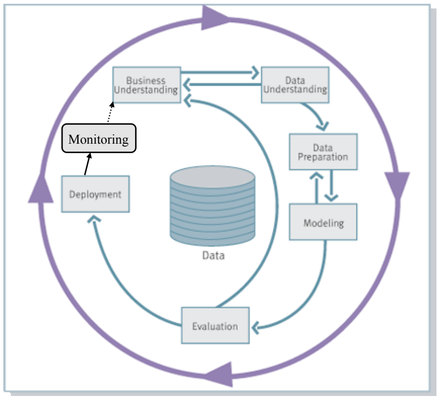
Data Preparation for Knowledge Discovery



Outline: Data Preparation

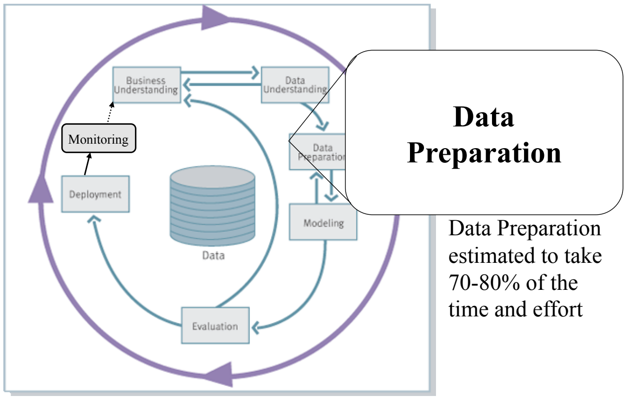
- Data Understanding
- Data Cleaning
  - Metadata
  - Missing Values
  - Unified Date Format
  - Nominal to Numeric
  - Discretization
- Field Selection and "False Predictors"
- Unbalanced Target Distribution

Knowledge Discovery Process flow, according to CRISP-DM



see www.crisp-dm.org for more information

Knowledge Discovery Process, in practice



Data Preparation

Data Preparation estimated to take 70-80% of the time and effort

Data Understanding: Relevance

- What data is available for the task?
- Is this data relevant?
- Is additional relevant data available?
- How much historical data is available?
- Who is the data expert ?

Data Understanding: Quantity

- Number of instances (records)
  - *Rule of thumb: 5,000 or more desired*
  - if less, results are less reliable; use special methods (boosting, …)
- Number of attributes (fields)
  - *Rule of thumb: for each field, 10 or more instances*
  - If more fields, use feature reduction and selection
- Number of targets
  - *Rule of thumb: >100 for each class*
  - if very unbalanced, use stratified sampling

Data Cleaning Steps

- Data acquisition and metadata
- Missing values
- Unified date format
- Converting nominal to numeric
- Discretization of numeric data
- Data validation and statistics

Data Cleaning: Acquisition

- Data can be in DBMS
  - ODBC, JDBC protocols
- Data in a flat file
  - Fixed-column format
  - Delimited format: tab, comma ",", other
  - E.g. C4.5 and Weka "arff" use comma-delimited data
  - Attention: Convert field delimiters inside strings
- Verify the number of fields before and after

Data Cleaning: Example

- **Original data (fixed column format)**

```
000000000130.06.19971979-10-3080145722     #000310 111000301.01.000100000000004
0000000000000.000000000000000.000000000000000.000000000000000.000000000000000.000000000000000.0000
00000000000. 000000000000000.000000000000000.0000000......
000000000000000.000000000000000.000000000000000.000000000000000.000000000000000.000000000000000.00
0000000000000.000000000000000.000000000000000.000000000000000.000000000000000.000000000000000.0000
00000000000.000000000000000.000000000000000.000000000000000.000000000000000.000000000000000.000000
000000000.000000000000000.000000000000000.000000000000000.00 000000000300.00 000000000300.00
```

- **Clean data**

```
0000000001,199706,1979.833,8014,5722   ,   ,#000310  .... ,
111,03,000101,0,04,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0300,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0300,0300.00
```

Data Cleaning: Metadata

- **Field types:**
  - **binary, nominal (categorical), ordinal, numeric, …**
  - **For nominal fields: tables translating codes to full descriptions**
- **Field role:**
  - input : inputs for modeling
  - target : output
  - id/auxiliary : keep, but not use for modeling
  - ignore : don't use for modeling
  - weight : instance weight
  - …
- **Field descriptions**

Data Cleaning: Reformatting

Convert data to a standard format (e.g. arff or csv)

- Missing values
- Unified date format
- Binning of numeric data
- Fix errors and outliers
- Convert nominal fields whose values have order to numeric.
  - ***Q: Why?*** A: to be able to use ">" and "<" comparisons on these fields)

Data Cleaning: Missing Values

- Missing data can appear in several forms:
  - <empty field> "0" "." "999" "NA" …
- Standardize missing value code(s)
- Dealing with missing values:
  - ignore records with missing values
  - treat missing value as a separate value
  - Imputation: fill in with mean or median values

Data Cleaning: Unified Date Format

- We want to transform all dates to the same format internally
- Some systems accept dates in many formats
  - e.g. "Sep 24, 2003" , 9/24/03, 24.09.03, etc
  - dates are transformed internally to a standard value
- Frequently, just the year (YYYY) is sufficient
- For more details, we may need the month, the day, the hour, etc
- Representing date as YYYYMM or YYYYMMDD can be OK, but has problems
- ***Q: What are the problems with YYYYMMDD dates?***
  - A: Ignoring for now the Looming Y10K (year 10,000 crisis …)
  - YYYYMMDD does not preserve intervals:
  - 20040201 - 20040131 /= 20040131 – 20040130
  - This can introduce bias into models

Unified Date Format Options

- To preserve intervals, we can use
  - Unix system date: Number of seconds since 1970
  - Number of days since Jan 1, 1960 (SAS)
- Problem:
  - values are non-obvious
  - don't help intuition and knowledge discovery
  - harder to verify, easier to make an error

KSP Date Format

$$\text{KSP Date} = \text{YYYY} + \frac{\text{days\_starting\_Jan\_1} - 0.5}{365 + \text{1\_if\_leap\_year}}$$

- Preserves intervals (almost)
- The year and quarter are obvious
  - Sep 24, 2003 is 2003 + (267-0.5)/365= 2003.7301 (round to 4 digits)
- Consistent with date starting at noon
- Can be extended to include time

Y2K issues: 2 digit Year

- 2-digit year in old data – legacy of Y2K
- E.g. Q: Year 02 – is it 1902 or 2002 ?
  - A: Depends on context (e.g. child birthday or year of house construction)
  - Typical approach: CUTOFF year, e.g. 30
  - if YY < CUTOFF , then 20YY, else 19YY

Conversion: Nominal to Numeric

- Some tools can deal with nominal values internally
- Other methods (neural nets, regression, nearest neighbor) require only numeric inputs
- To use nominal fields in such methods need to convert them to a numeric value
  - Q: Why not ignore nominal fields altogether?
  - A: They may contain valuable information
- Different strategies for binary, ordered, multi-valued nominal fields

Conversion: Binary to Numeric

- Binary fields
  - E.g. Gender=M, F
- Convert to Field_0_1 with 0, 1 values
  - e.g. Gender = M → Gender_0_1 = 0
  - Gender = F → Gender_0_1 = 1

Conversion: Ordered to Numeric

- Ordered attributes (e.g. Grade) can be converted to numbers preserving *natural* order, e.g.
  - A → 4.0
  - A- → 3.7
  - B+ → 3.3
  - B → 3.0
- Q: Why is it important to preserve *natural* order?
- A: To allow meaningful comparisons, e.g. Grade > 3.5

Conversion: Nominal, Few Values

- Multi-valued, unordered attributes with small (*rule of thumb < 20*) no. of values
  - e.g. Color=Red, Orange, Yellow, …, Violet
  - for each value *v* create a binary "flag" variable C_*v* , which is 1 if Color=*v*, 0 otherwise

ID	Color	…
371	red	
433	yellow	

⇨

ID	C_red	C_orange	C_yellow	…
371	1	0	0	
433	0	0	1	

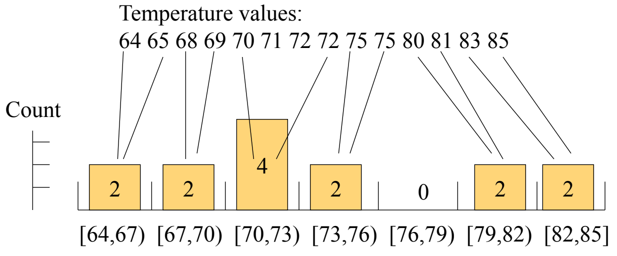
Conversion: Nominal, Many Values

- Examples:
  - US State Code (50 values)
  - Profession Code (7,000 values, but only few frequent)
- Q: How to deal with such fields ?
- A: Ignore ID-like fields whose values are unique for each record
- For other fields, group values "naturally":
  - e.g. 50 US States → 3 or 5 regions
  - Profession – select most frequent ones, group the rest
- Create binary flag-fields for selected values

Data Cleaning: Discretization

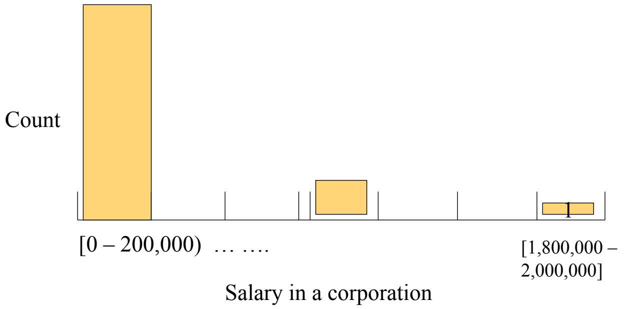
- Some methods require discrete values, e.g. most versions of Naïve Bayes, CHAID
- Discretization is very useful for generating a summary of data
- Also called "binning"

Discretization: Equal-Width



Equal Width, bins Low <= value < High

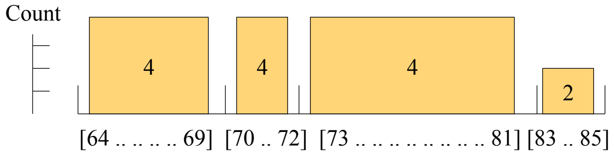
Discretization: Equal-Width may produce clumping



Salary in a corporation

Discretization: Equal-Height

Temperature values:
64 65 68 69 70 71 72 72 75 75 80 81 83 85



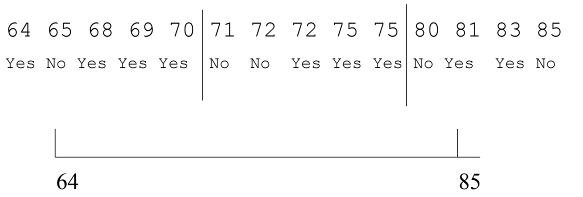
Equal Height = 4, except for the last bin

Discretization: Equal-height advantages

- Generally preferred because avoids clumping
- In practice, "almost-equal" height binning is used which avoids clumping and gives more intuitive breakpoints
- Additional considerations:
  - don't split frequent values across bins
  - create separate bins for special values (e.g. 0)
  - readable breakpoints (e.g. round breakpoints)

Discretization: Class Dependent

Eibe – min of 3 values per bucket



```
64  65  68  69  70 | 71  72  72  75  75 | 80  81  83  85
Yes No  Yes Yes Yes| No  No  Yes Yes Yes| No  Yes Yes No
```

Discretization considerations

- Equal Width is simplest, good for many classes
  - can fail miserably for unequal distributions
- Equal Height gives better results
- Class-dependent can be better for classification
  - Note: decision trees build discretization on the fly
  - Naïve Bayes requires initial discretization
- Many other methods exist …

Outliers and Errors

- Outliers are values thought to be out of range.
- Approaches:
  - do nothing
  - enforce upper and lower bounds
  - let binning handle the problem

Examine Data Statistics

```
*****************      Field 9:  MILES_ACCUMULATED

Total entries = 865636 (23809 different values).  Contains non-numeric values.    Missing
data indicated by "" (and possibly others).

Numeric items = 165161,  high = 418187.000,  low = -95050.000
       mean = 4194.557,  std = 10505.109,  skew =    7.000

Most frequent entries:

       Value   Total
           :          700474 ( 80.9%)
          0:          32748 (  3.8%)
          1:          416 (  0.0%)
          2:          337 (  0.0%)
         10:          321 (  0.0%)
          8:          284 (  0.0%)
          5:          269 (  0.0%)
          6:          267 (  0.0%)
         12:          262 (  0.0%)
          7:          246 (  0.0%)
          4:          237 (  0.0%)
```

Data Cleaning: Field Selection

First: Remove fields with no or little variability

- Examine the number of distinct field values
  - *Rule of thumb: remove a field where almost all values are the same (e.g. null), except possibly in minp % or less of all records.*
  - *minp* could be 0.5% or more generally less than 5% of the number of targets of the smallest class

False Predictors or Information "Leakers"

- False predictors are fields correlated to target behavior, which describe events that happen at the same time or *after* the target behavior
- If databases don't have the event dates, a false predictor will appear as a good predictor
- Example: Service cancellation date is a leaker when predicting attriters.
- Q: Give another example of a false predictor
- A: e.g. student final grade, for the task of predicting whether the student passed the course

False Predictors: Find "suspects"

- Build an initial decision-tree model
- Consider very strongly predictive fields as "suspects"
  - strongly predictive – if a field by itself provides close to 100% accuracy, at the top or a branch below
- Verify "suspects" using domain knowledge or with a domain expert
- Remove false predictors and build an initial model

(Almost) Automated False Predictor Detection

- For each field
  - Build 1-field decision trees for each field
  - (or compute correlation with the target field)
- Rank all suspects by 1-field prediction accuracy (or correlation)
- Remove suspects whose accuracy is close to 100% (Note: the threshold is domain dependent)
- Verify top "suspects" with domain expert

Selecting Most Relevant Fields

- If there are too many fields, select a subset that is most relevant.
- Can select top N fields using 1-field predictive accuracy as computed earlier.
- What is good N?
  - Rule of thumb -- keep top 50 fields

Field Reduction Improves Classification

- most learning algorithms look for non-linear combinations of fields -- can easily find many spurious combinations given small # of records and large # of fields
- Classification accuracy improves if we first reduce number of fields
- Multi-class heuristic: select equal # of fields from each class

Derived Variables

- Better to have a fair modeling method and good variables, than to have the best modeling method and poor variables.
- Insurance Example: People are eligible for pension withdrawal at age 59 ½. Create it as a separate Boolean variable!
- *Advanced methods exists for automatically examining variable combinations, but it is very computationally expensive!

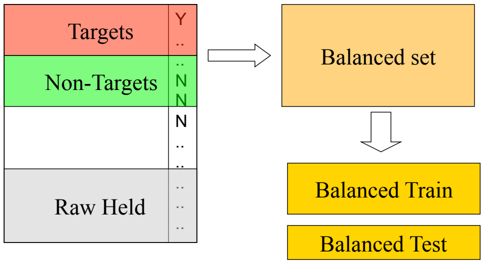
Unbalanced Target Distribution

- Sometimes, classes have very unequal frequency
  - Attrition prediction: 97% stay, 3% attrite (in a month)
  - medical diagnosis: 90% healthy, 10% disease
  - eCommerce: 99% don’t buy, 1% buy
  - Security: >99.99% of Americans are not terrorists
- Similar situation with multiple classes
- Majority class classifier can be 97% correct, but useless

Handling Unbalanced Data

- With two classes: let positive targets be a minority
- Separate raw held-aside set (e.g. 30% of data) and raw train
  - put aside raw held-aside and don’t use it till the final model
- Select remaining positive targets (e.g. 70% of all targets) from raw train
- Join with equal number of negative targets from raw train, and randomly sort it.
- Separate randomized balanced set into balanced train and balanced test

Building Balanced Train Sets



Targets

Y

..

Non-Targets

..

N

N

N

..

..

Raw Held

..

..

..

Balanced set

Balanced Train

Balanced Test

Learning with Unbalanced Data

- Build models on balanced train/test sets
- Estimate the final results (lift curve) on the raw held set

- Can generalize “balancing” to multiple classes
  - stratified sampling
  - Ensure that each class is represented with approximately equal proportions in train and test

Data Preparation Key Ideas

- Use meta-data
- Inspect data for anomalies and errors
- Eliminate “false positives”
- Develop small, reusable software components
- Plan for verification - verify the results after each step

Summary

Good data preparation is key to producing valid and reliable models

Clustering

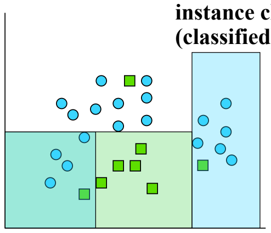
Outline

- Introduction
- K-means clustering
- Hierarchical clustering: COBWEB

Classification vs. Clustering

Classification: Supervised learning:

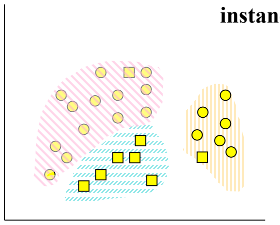
Learns a method for predicting the instance class from pre-labeled (classified) instances



Clustering

Unsupervised learning:

Finds "natural" grouping of instances given un-labeled data



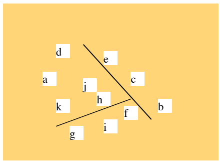
Clustering Methods

- Many different method and algorithms:
  - For numeric and/or symbolic data
  - Deterministic vs. probabilistic
  - Exclusive vs. overlapping
  - Hierarchical vs. flat
  - Top-down vs. bottom-up

Clusters: exclusive vs. overlapping

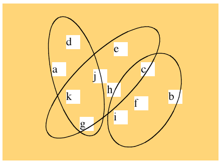
Simple 2-D representation

Non-overlapping



Venn diagram

Overlapping



Clustering Evaluation

- Manual inspection
- Benchmarking on existing labels
- Cluster quality measures
  - distance measures
  - high similarity within a cluster, low across clusters

The distance function

- Simplest case: one numeric attribute A
  - Distance(X,Y) = A(X) – A(Y)
- Several numeric attributes:
  - Distance(X,Y) = Euclidean distance between X,Y
- Nominal attributes: distance is set to 1 if values are different, 0 if they are equal
- Are all attributes equally important?
  - Weighting the attributes might be necessary

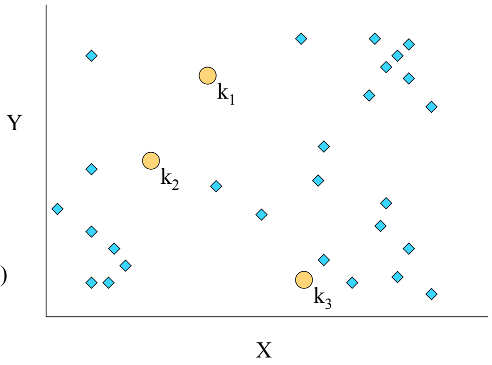
Simple Clustering: K-means

Works with numeric data only

1) Pick a number (K) of cluster centers (at random)
2) Assign every item to its nearest cluster center (e.g. using Euclidean distance)
3) Move each cluster center to the mean of its assigned items
4) Repeat steps 2,3 until convergence (change in cluster assignments less than a threshold)

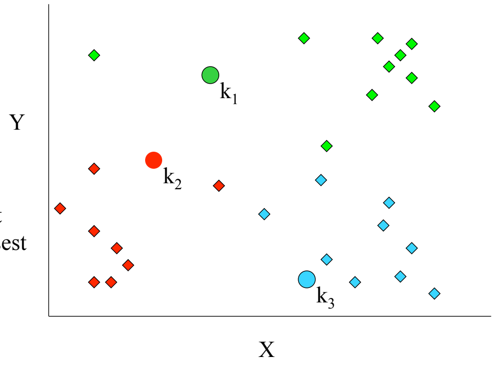
K-means example, step 1

Pick 3 initial cluster centers (randomly)



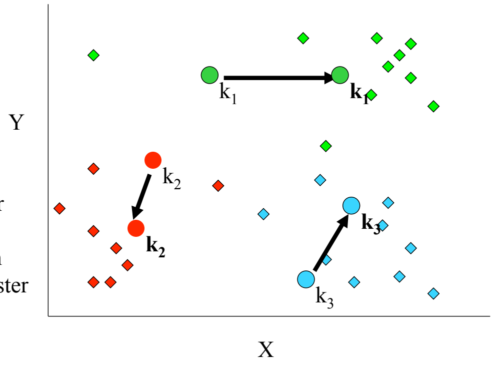
K-means example, step 2

Assign each point to the closest cluster center



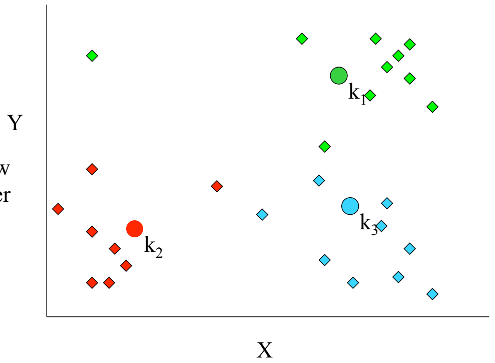
K-means example, step 3

Move each cluster center to the mean of each cluster



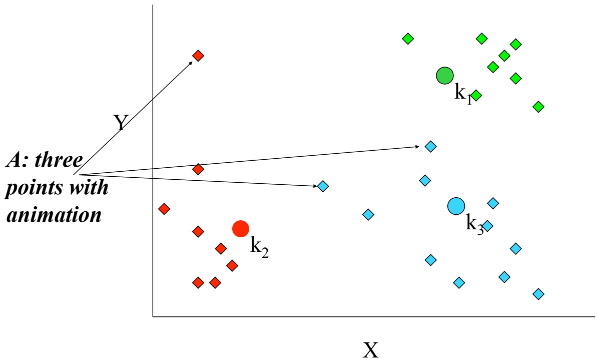
K-means example, step 4

Reassign points closest to a different new cluster center



Q: Which points are reassigned?

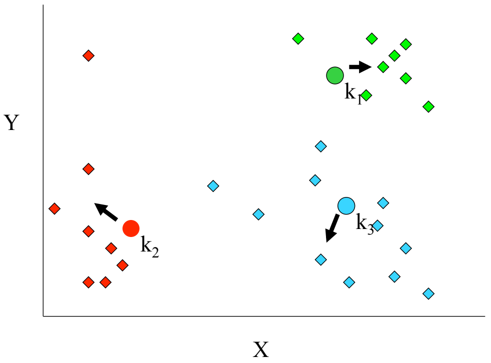
K-means example, step 4 …



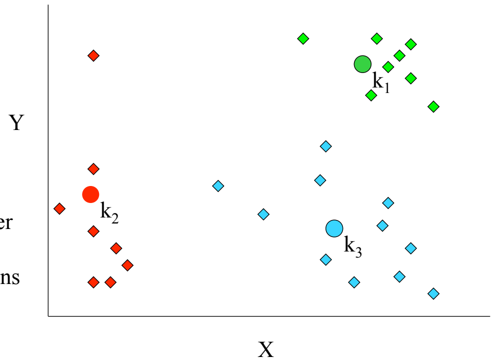
A: three points with animation

K-means example, step 4b

re-compute cluster means



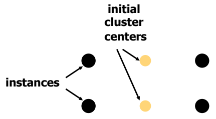
K-means example, step 5



move cluster centers to cluster means

Discussion

- Result can vary significantly depending on initial choice of seeds
- Can get trapped in local minimum
  - Example:



- To increase chance of finding global optimum: restart with different random seeds

K-means clustering summary

Advantages

- Simple, understandable
- items automatically assigned to clusters

Disadvantages

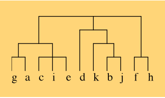
- Must pick number of clusters before hand
- All items forced into a cluster
- Too sensitive to outliers

K-means variations

- **K-medoids** – instead of mean, use medians of each cluster
  - Mean of 1, 3, 5, 7, 9 is **5**
  - Mean of 1, 3, 5, 7, 1009 is **205**
  - Median of 1, 3, 5, 7, 1009 is **5**
  - Median advantage: not affected by extreme values
- For large databases, use sampling

*Hierarchical clustering

- Bottom up
  - Start with single-instance clusters
  - At each step, join the two closest clusters
  - Design decision: distance between clusters
    - E.g. two closest instances in clusters vs. distance between means
- Top down
  - Start with one universal cluster
  - Find two clusters
  - Proceed recursively on each subset
  - Can be very fast
- Both methods produce a *dendrogram*

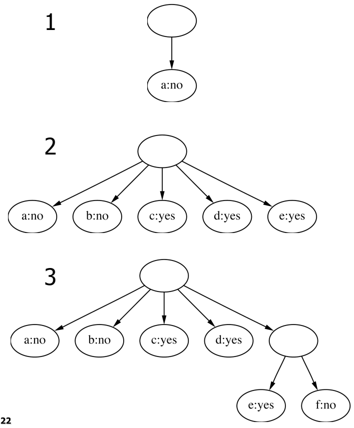


*Incremental clustering

- Heuristic approach (COBWEB/CLASSIT)
- Form a hierarchy of clusters incrementally
- Start:
  - tree consists of empty root node
- Then:
  - add instances one by one
  - update tree appropriately at each stage
  - to update, find the right leaf for an instance
  - May involve restructuring the tree
- Base update decisions on *category utility*

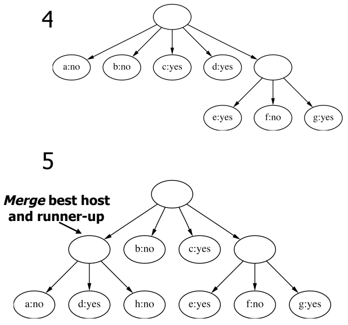
*Clustering weather data

ID	Outlook	Temp.	Humidity	Windy
A	Sunny	Hot	High	False
B	Sunny	Hot	High	True
C	Overcast	Hot	High	False
D	Rainy	Mild	High	False
E	Rainy	Cool	Normal	False
F	Rainy	Cool	Normal	True
G	Overcast	Cool	Normal	True
H	Sunny	Mild	High	False
I	Sunny	Cool	Normal	False
J	Rainy	Mild	Normal	False
K	Sunny	Mild	Normal	True
L	Overcast	Mild	High	True
M	Overcast	Hot	Normal	False
N	Rainy	Mild	High	True



*Clustering weather data

ID	Outlook	Temp.	Humidity	Windy
A	Sunny	Hot	High	False
B	Sunny	Hot	High	True
C	Overcast	Hot	High	False
D	Rainy	Mild	High	False
E	Rainy	Cool	Normal	False
F	Rainy	Cool	Normal	True
G	Overcast	Cool	Normal	True
H	Sunny	Mild	High	False
I	Sunny	Cool	Normal	False
J	Rainy	Mild	Normal	False
K	Sunny	Mild	Normal	True
L	Overcast	Mild	High	True
M	Overcast	Hot	Normal	False
N	Rainy	Mild	High	True

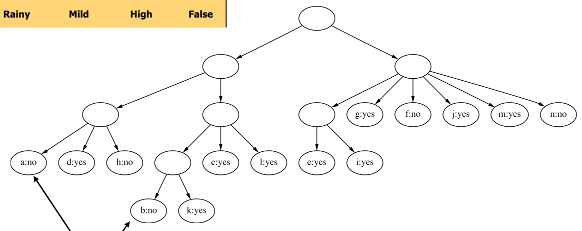


Merge **best host and runner-up**

Consider *splitting* the best host if merging doesn't help

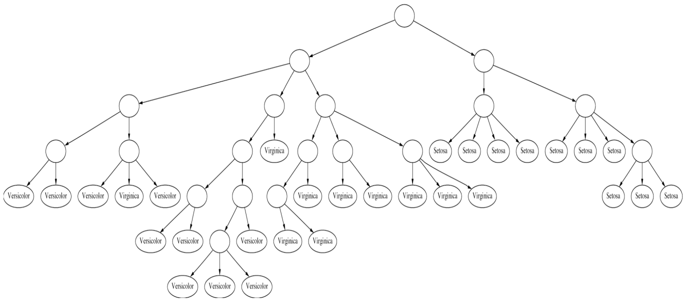
*Final hierarchy

ID	Outlook	Temp.	Humidity	Windy
A	Sunny	Hot	High	False
B	Sunny	Hot	High	True
C	Overcast	Hot	High	False
D	Rainy	Mild	High	False

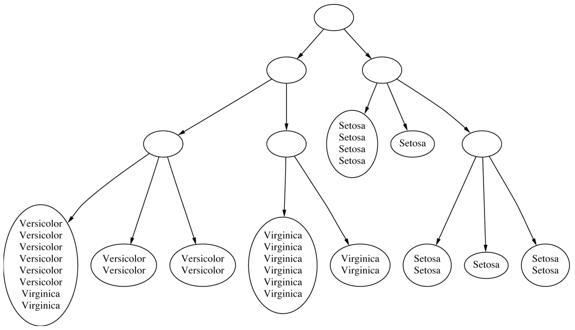


Oops! *a* and *b* are actually very similar

*Example: the iris data (subset)



*Clustering with cutoff



*Category utility

- Category utility: quadratic loss function defined on conditional probabilities:

$$CU(C_1, C_2, \ldots, C_k) = \frac{\sum_l \Pr[C_l] \underset{\uparrow}{\sum_i} \underset{\nwarrow}{\sum_j} (\Pr[a_i = v_{ij} \mid C_l]^2 - \Pr[a_i = v_{ij}]^2)}{k}$$

- Every instance in different category ⇒ numerator becomes

$$\underset{\substack{\uparrow \\ \textbf{number of attributes}}}{m} - \Pr[a_i = v_{ij}]^2 \longleftarrow \textbf{\textit{maximum}}$$

*Overfitting-avoidance heuristic

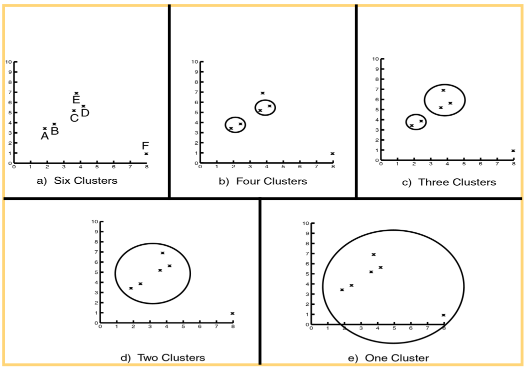
- If every instance gets put into a different category the numerator becomes (maximal):

$$n - \sum_i \sum_j \Pr[a_i = v_{ij}]^2 \longleftarrow \boxed{\textbf{Maximum value of CU}}$$

Where *n* is number of all possible attribute values.

- So without *k* in the denominator of the CU-formula, every cluster would consist of one instance!

Levels of Clustering



a) Six Clusters

b) Four Clusters

c) Three Clusters

d) Two Clusters

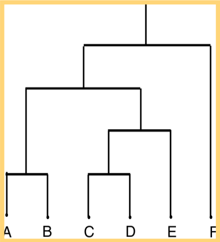
e) One Cluster

Hierarchical Clustering

- Clusters are created in levels actually creating sets of clusters at each level.
- ***Agglomerative***
  - Initially each item in its own cluster
  - Iteratively clusters are merged together
  - Bottom Up
- ***Divisive***
  - Initially all items in one cluster
  - Large clusters are successively divided
  - Top Down

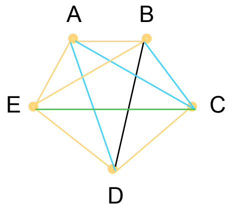
Dendrogram

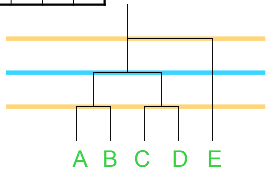
- ***Dendrogram:*** a tree data structure which illustrates hierarchical clustering techniques.
- Each level shows clusters for that level.
  - Leaf – individual clusters
  - Root – one cluster
- A cluster at level i is the union of its children clusters at level i+1.



Agglomerative Example

	A	B	C	D	E
A	0	1	2	2	3
B	1	0	2	4	3
C	2	2	0	1	5
D	2	4	1	0	3
E	3	3	5	3	0



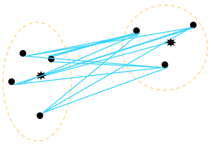


Threshold of

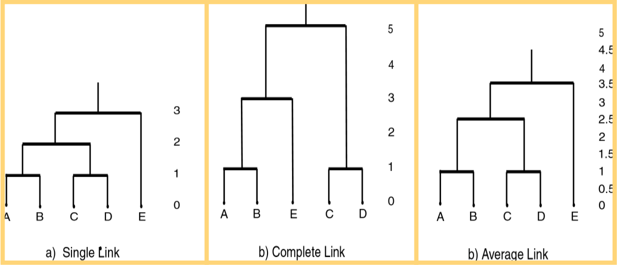
1 2 3 4 5

Distance Between Clusters

- ***Single Link***: smallest distance between points
- ***Complete Link:*** largest distance between points
- ***Average Link:*** average distance between points
- ***Centroid:*** distance between centroids



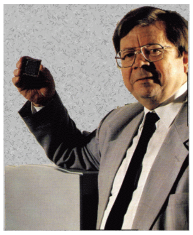
Single Link Clustering



Other Clustering Approaches

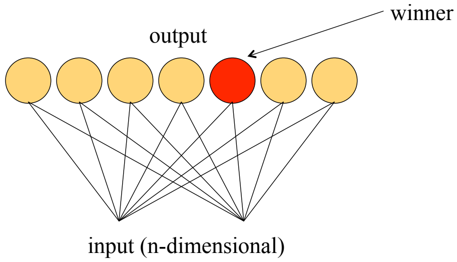
- EM – probability based clustering
- Bayesian clustering
- SOM – self-organizing maps
- …

Self-Organizing Map



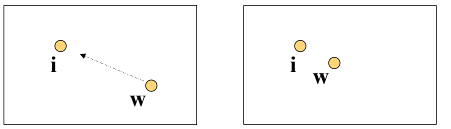
Self Organizing Map

- Unsupervised learning
- Competitive learning



Self Organizing Map

- Determine the winner (the neuron of which the weight vector has the smallest distance to the input vector)
- Move the weight vector **w** of the winning neuron towards the input **i**

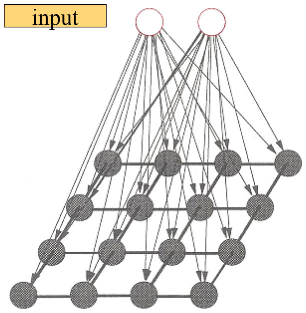


Before learning *After learning*

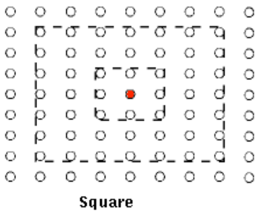
Self Organizing Map

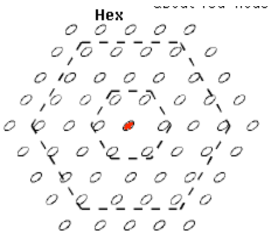
- Impose a topological order onto the competitive neurons (e.g., rectangular map)
- Let neighbors of the winner share the "prize" (The "postcode lottery" principle)
- After learning, neurons with similar weights tend to cluster on the map

Self Organizing Map



Self Organizing Map

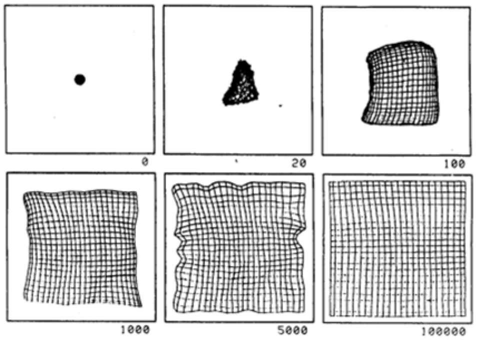




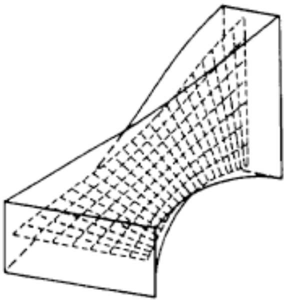
Self Organizing Map

- Input: uniformly randomly distributed points
- Output: Map of 20^2 neurons
- Training
  - Starting with a large learning rate and neighborhood size, both are gradually decreased to facilitate convergence

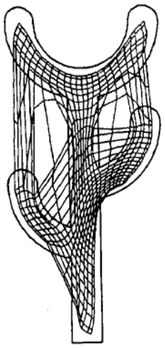
Self Organizing Map

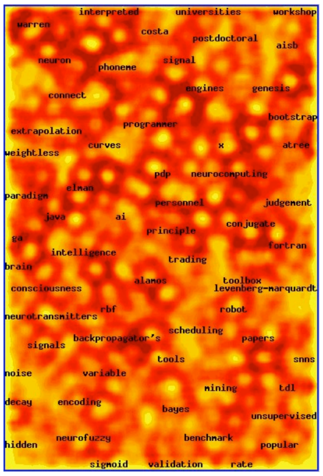


Self Organizing Map

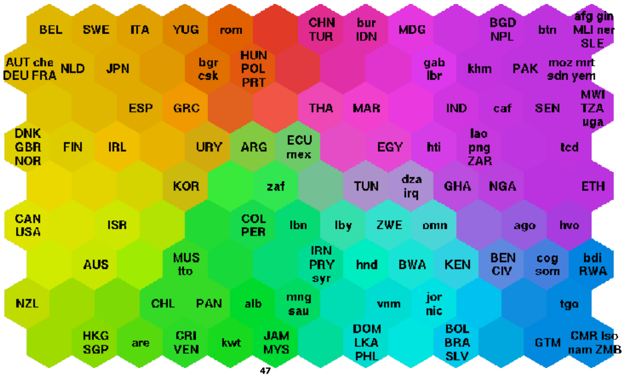


Self Organizing Map

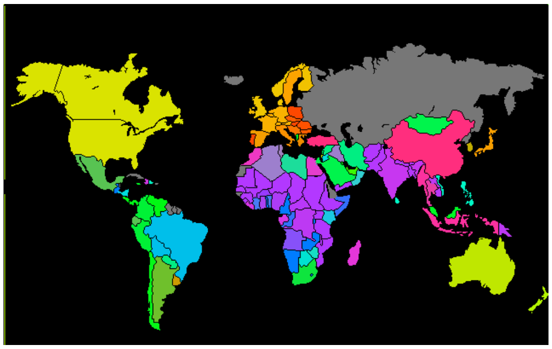




Self Organizing Map



Self Organizing Map



Discussion

- Can interpret clusters by using supervised learning
  - learn a classifier based on clusters
- Decrease dependence between attributes?
  - pre-processing step
  - E.g. use *principal component analysis*
- Can be used to fill in missing values
- Key advantage of probabilistic clustering:
  - Can estimate likelihood of data
  - Use it to compare different models objectively

Examples of Clustering Applications

- **Marketing:** discover customer groups and use them for targeted marketing and re-organization
- **Astronomy:** find groups of similar stars and galaxies
- **Earth-quake studies:** Observed earth quake epicenters should be clustered along continent faults
- **Genomics:** finding groups of gene with similar expressions
- …

Clustering Summary

- unsupervised
- many approaches
  - K-means – simple, sometimes useful
    - K-medoids is less sensitive to outliers
  - Hierarchical clustering – works for symbolic attributes
- Evaluation is a problem

Associations and Frequent Item Analysis

Outline

- Transactions
- Frequent itemsets
- Subset Property
- Association rules
- Applications

Transactions Example

TID	Produce
1	MILK, BREAD, EGGS
2	BREAD, SUGAR
3	BREAD, CEREAL
4	MILK, BREAD, SUGAR
5	MILK, CEREAL
6	BREAD, CEREAL
7	MILK, CEREAL
8	MILK, BREAD, CEREAL, EGGS
9	MILK, BREAD, CEREAL

Transaction database: Example

TID	Products
1	A, B, E
2	B, D
3	B, C
4	A, B, D
5	A, C
6	B, C
7	A, C
8	A, B, C, E
9	A, B, C

ITEMS:

Instances = Transactions

A = milk
B= bread
C= cereal
D= sugar
E= eggs

Transaction database: Example

Attributes converted to binary flags

TID	Products
1	A, B, E
2	B, D
3	B, C
4	A, B, D
5	A, C
6	B, C
7	A, C
8	A, B, C, E
9	A, B, C

TID	A	B	C	D	E
1	1	1	0	0	1
2	0	1	0	1	0
3	0	1	1	0	0
4	1	1	0	1	0
5	1	0	1	0	0
6	0	1	1	0	0
7	1	0	1	0	0
8	1	1	1	0	1
9	1	1	1	0	0

Definitions

- Item: *attribute*=*value* pair or simply *value*
  - usually attributes are converted to binary *flags* for each value, e.g. **product="A"** is written as **"A"**
- Itemset I : a subset of possible items
  - Example: I = {A,B,E} (order unimportant)
- Transaction: (TID, itemset)
  - TID is transaction ID

Support and Frequent Itemsets

- Support of an itemset
  - sup(*I*) = no. of transactions *t* that support (i.e. contain) *I*
- In example database:
  - sup ({A,B,E}) = 2, sup ({B,C}) = 4
- Frequent itemset *I* is one with at least the minimum support count
  - sup(*I*) >= *minsup*

SUBSET PROPERTY

- **Every subset of a frequent set is frequent!**
- Q: Why is it so?
- A: Example: Suppose {A,B} is frequent. Since each occurrence of A,B includes both A and B, then both A and B must also be frequent
- Similar argument for larger itemsets
- Almost all association rule algorithms are based on this subset property

Association Rules

- Association rule *R* : *Itemset1 => Itemset2*
  - *Itemset1, 2* are disjoint and *Itemset2* is non-empty
  - meaning: if transaction includes *Itemset1* then it also has *Itemset2*
- Examples
  - A,B => E,C
  - A => B,C

From Frequent Itemsets to Association Rules

- *Q: Given frequent set {A,B,E}, what are possible association rules?*
  - A => B, E
  - A, B => E
  - A, E => B
  - B => A, E
  - B, E => A
  - E => A, B
  - __ => A,B,E (empty rule), or true => A,B,E

Classification vs Association Rules

Classification Rules

- Focus on one target field
- Specify class in all cases
- Measures: Accuracy

Association Rules

- Many target fields
- Applicable in some cases
- Measures: Support, Confidence, Lift

Rule Support and Confidence

- Suppose *R : I => J* is an association rule
  - sup (R) = sup (I ∪ J) is the *support count*
    - support of itemset I ∪ J (I or J)
  - conf (R) = sup(J) / sup(R) is the *confidence* of R
    - fraction of transactions with I ∪ J that have J
- Association rules with minimum support and count are sometimes called "***strong***" rules

Association Rules Example:

- ***Q: Given frequent set {A,B,E}, what association rules have minsup = 2 and minconf= 50% ?***

A, B => E : conf=2/4 = 50%

A, E => B : conf=2/2 = 100%

B, E => A : conf=2/2 = 100%

E => A, B : conf=2/2 = 100%

Don't qualify

A =>B, E : conf=2/6 =33%< 50%

B => A, E : conf=2/7 = 28% < 50%

__ => A,B,E : conf: 2/9 = 22% < 50%

TID	List of items
1	**A, B, E**
2	B, D
3	B, C
4	A, B, D
5	A, C
6	B, C
7	A, C
8	**A, B, C, E**
9	A, B, C

Find Strong Association Rules

- A rule has the parameters *minsup* and *minconf*:
  - sup(R) >= *minsup* and conf (R) >= *minconf*
- Problem:
  - Find all association rules with given *minsup* and *minconf*
- First, find all frequent itemsets

Finding Frequent Itemsets

- Start by finding one-item sets (easy)
- *Q: How?*
- A: Simply count the frequencies of all items

Finding itemsets: next level

- Apriori algorithm (Agrawal & Srikant)
- Idea: use one-item sets to generate two-item sets, two-item sets to generate three-item sets, …
  - If (A B) is a frequent item set, then (A) and (B) have to be frequent item sets as well!
  - In general: if X is frequent *k*-item set, then all (*k*-1)-item subsets of X are also frequent

  ⇒Compute *k*-item set by merging (*k*-1)-item sets

An example

- Given: five three-item sets

  `(A B C), (A B D), (A C D), (A C E), (B C D)`
- Lexicographic order improves efficiency
- Candidate four-item sets:

  `(A B C D)` **Q: OK?**

A: yes, because all 3-item subsets are frequent

`(A C D E)` **Q: OK?**

A: No, because (C D E) is not frequent

Generating Association Rules

- Two stage process:
  - Determine frequent itemsets e.g. with the Apriori algorithm.
  - For each frequent item set *I*
    - for each subset *J* of *I*
      - determine all association rules of the form: *I-J => J*
- Main idea used in both stages : subset property

Example: Generating Rules from an Itemset

- Frequent itemset from golf data:

`Humidity = Normal, Windy = False, Play = Yes (4)`

- Seven potential rules:

If Humidity = Normal and Windy = False then Play = Yes	4/4
If Humidity = Normal and Play = Yes then Windy = False	4/6
If Windy = False and Play = Yes then Humidity = Normal	4/6
If Humidity = Normal then Windy = False and Play = Yes	4/7
If Windy = False then Humidity = Normal and Play = Yes	4/8
If Play = Yes then Humidity = Normal and Windy = False	4/9
If True then Humidity = Normal and Windy = False and Play = Yes	4/12

Rules for the weather data

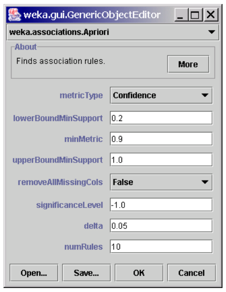
- Rules with support > 1 and confidence = 100%:

	Association rule		Sup.	Conf.
1	Humidity=Normal Windy=False	⇒Play=Yes	4	100%
2	Temperature=Cool	⇒Humidity=Normal	4	100%
3	Outlook=Overcast	⇒Play=Yes	4	100%
4	Temperature=Cold Play=Yes	⇒Humidity=Normal	3	100%
...	...	...	...	...
58	Outlook=Sunny Temperature=Hot	⇒Humidity=High	2	100%

- In total: 3 rules with support four, 5 with support three, and 50 with support two

Weka associations

File: weather.nominal.arff
MinSupport: 0.2

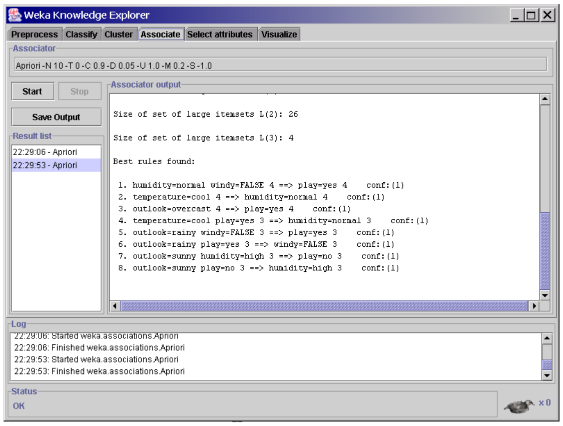


weka.associations.Apriori

About
Finds association rules.

metricType	Confidence
lowerBoundMinSupport	0.2
minMetric	0.9
upperBoundMinSupport	1.0
removeAllMissingCols	False
significanceLevel	-1.0
delta	0.05
numRules	10

Weka associations: output



Associator
Apriori -N 10 -T 0 -C 0.9 -D 0.05 -U 1.0 -M 0.2 -S -1.0

Result list
22:29:06 - Apriori
22:29:53 - Apriori

Associator output

```
Size of set of large itemsets L(2): 26

Size of set of large itemsets L(3): 4

Best rules found:

 1. humidity=normal windy=FALSE 4 ==> play=yes 4    conf:(1)
 2. temperature=cool 4 ==> humidity=normal 4    conf:(1)
 3. outlook=overcast 4 ==> play=yes 4    conf:(1)
 4. temperature=cool play=yes 3 ==> humidity=normal 3    conf:(1)
 5. outlook=rainy windy=FALSE 3 ==> play=yes 3    conf:(1)
 6. outlook=rainy play=yes 3 ==> windy=FALSE 3    conf:(1)
 7. outlook=sunny humidity=high 3 ==> play=no 3    conf:(1)
 8. outlook=sunny play=no 3 ==> humidity=high 3    conf:(1)
```

Log

```
22:29:06: Started weka.associations.Apriori
22:29:06: Finished weka.associations.Apriori
22:29:53: Started weka.associations.Apriori
22:29:53: Finished weka.associations.Apriori
```

Status
OK

Filtering Association Rules

- Problem: any large dataset can lead to very large number of association rules, even with reasonable Min Confidence and Support
- Confidence by itself is not sufficient
  - e.g. if all transactions include Z, then
  - any rule I => Z will have confidence 100%.
- Other measures to filter rules

Association Rule LIFT

- The *lift* of an association rule *I => J* is defined as:
  - lift = P(J|I) / P(J)
  - Note, P(I) = (support of I) / (no. of transactions)
  - ratio of confidence to expected confidence
- Interpretation:
  - if lift > 1, then I and J are positively correlated

    lift < 1, then I are J are negatively correlated.

    lift = 1, then I and J are independent.

Other issues

- ARFF format very inefficient for typical *market basket data*
  - Attributes represent items in a basket and most items are usually missing
- Interestingness of associations
  - find unusual associations: Milk usually goes with bread, but soy milk does not.

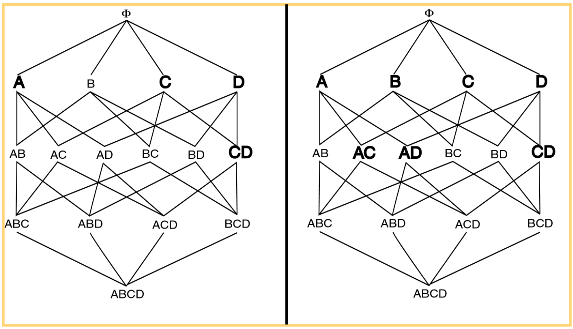
Beyond Binary Data

- Hierarchies
  - drink → milk → low-fat milk → Stop&Shop low-fat milk …
  - find associations on any level
- Sequences over time
- …

Sampling

- Large databases
- Sample the database and apply Apriori to the sample.
- ***Potentially Large Itemsets (PL):*** Large itemsets from sample
- ***Negative Border (BD^{-}):***
  - Generalization of Apriori-Gen applied to itemsets of varying sizes.
  - Minimal set of itemsets which are not in PL, but whose subsets are all in PL.

Negative Border Example



PL

PL $\cup BD^{-}(PL)$

Sampling Algorithm

1. D_s = sample of Database D;
2. PL = Large itemsets in D_s using smalls;
3. C = PL $\cup$ $BD^{-}(PL)$;
4. Count C in Database using s;
5. ML = large itemsets in $BD^{-}(PL)$;
6. If ML = ∅ then done
7.   else C = repeated application of $BD^{-;}$
8.   Count C in Database;

Sampling Example

- Find AR assuming s = 20%
- D_s = { t_1,t_2}
- Smalls = 10%
- PL = {{Bread}, {Jelly}, {PeanutButter}, {Bread,Jelly}, {Bread,PeanutButter}, {Jelly, PeanutButter}, {Bread,Jelly,PeanutButter}}
- BD^{-}(PL)={{Beer},{Milk}}
- ML = {{Beer}, {Milk}}
- Repeated application of BD^{-} generates all remaining itemsets

Sampling Adv/Disadv

- ***Advantages:***
  - Reduces number of database scans to one in the best case and two in worst.
  - Scales better.
- ***Disadvantages:***
  - Potentially large number of candidates in second pass

Partitioning

- Divide database into partitions $D^1,D^2,\ldots,D^p$
- Apply Apriori to each partition
- Any large itemset must be large in at least one partition.

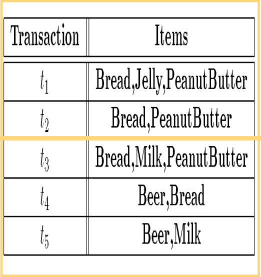
Partitioning Algorithm

```
Divide D into partitions $D^1,D^2,\ldots,D^p$;
For I = 1 to p do
    $L^i$ = Apriori($D^i$);
$C = L^1 \cup \ldots \cup L^p$;
Count C on D to generate L;
```

Partitioning Example

D^1

D^2



Transaction	Items
t_1	Bread,Jelly,PeanutButter
t_2	Bread,PeanutButter
t_3	Bread,Milk,PeanutButter
t_4	Beer,Bread
t_5	Beer,Milk

S=10%

L^1 ={{Bread}, {Jelly}, {PeanutButter}, {Bread,Jelly}, {Bread,PeanutButter}, {Jelly, PeanutButter}, {Bread,Jelly,PeanutButter}}

L^2 ={{Bread}, {Milk}, {PeanutButter}, {Bread,Milk}, {Bread,PeanutButter}, {Milk, PeanutButter}, {Bread,Milk,PeanutButter}, {Beer}, {Beer,Bread}, {Beer,Milk}}

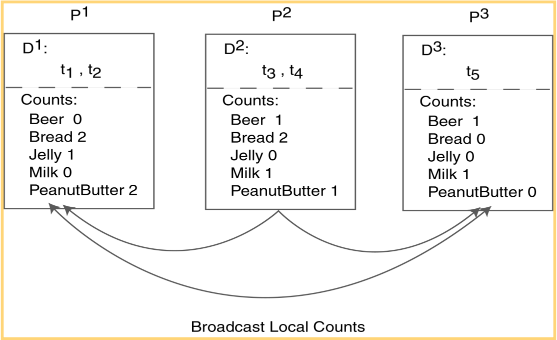
Partitioning Adv/Disadv

- ***Advantages:***
  - Adapts to available main memory
  - Easily parallelized
  - Maximum number of database scans is two.
- ***Disadvantages:***
  - May have many candidates during second scan.

Count Distribution Algorithm(CDA)

```
Place data partition at each site.
In Parallel at each site do
    $C_1$ = Itemsets of size one in I;
    Count $C_1$;
    Broadcast counts to all sites;
    Determine global large itemsets of size 1, $L_1$;
    i = 1;
    Repeat
        i = i + 1;
        $C_i$ = Apriori-Gen($L_{i-1}$);
        Count $C_i$;
        Broadcast counts to all sites;
        Determine global large itemsets of size i, $L_i$;
    until no more large itemsets found;
```

CDA Example



P^1

D^1:

t_1 , t_2

Counts:
Beer 0
Bread 2
Jelly 1
Milk 0
PeanutButter 2

P^2

D^2:

t_3 , t_4

Counts:
Beer 1
Bread 2
Jelly 0
Milk 1
PeanutButter 1

P^3

D^3:

t_5

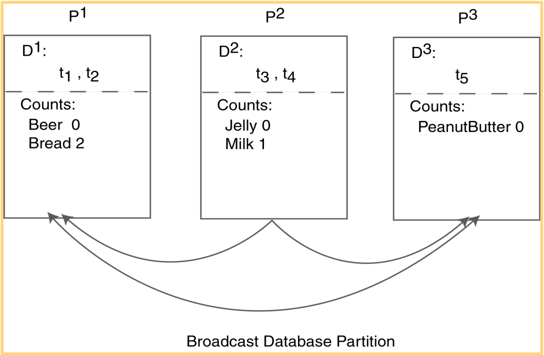
Counts:
Beer 1
Bread 0
Jelly 0
Milk 1
PeanutButter 0

Broadcast Local Counts

Data Distribution Algorithm(DDA)

```
Place data partition at each site.
In Parallel at each site do
    Determine local candidates of size 1 to count;
    Broadcast local transactions to other sites;
    Count local candidates of size 1 on all data;
    Determine large itemsets of size 1 for local
     candidates;
    Broadcast large itemsets to all sites;
    Determine $L_1$;
    i = 1;
    Repeat
        i = i + 1;
        $C_i$ = Apriori-Gen($L_{i-1}$);
        Determine local candidates of size i to count;
        Count, broadcast, and find  $L_i$;
    until no more large itemsets found;
```

DDA Example



P^1

D^1:

t_1 , t_2

Counts:
Beer 0
Bread 2

P^2

D^2:

t_3 , t_4

Counts:
Jelly 0
Milk 1

P^3

D^3:

t_5

Counts:
PeanutButter 0

Broadcast Database Partition

Applications

- Market basket analysis
  - Store layout, client offers
- …

Application Difficulties

- Wal-Mart knows that customers who buy Barbie dolls have a 60% likelihood of buying one of three types of candy bars.
- What does Wal-Mart do with information like that? 'I don't have a clue,' says Wal-Mart's chief of merchandising, Lee Scott
- See - KDnuggets 98:01 for many ideas www.kdnuggets.com/news/98/n01.html
- Diapers and beer urban legend

Summary

- Frequent itemsets
- Association rules
- Subset property
- Apriori algorithm
- Application difficulties

Controversial Issues

- Data mining (or simple analysis) on people may come with a profile that would raise controversial issues of
  - Discrimination
  - Privacy
  - Security
- Examples:
  - Should males between 18 and 35 from countries that produced terrorists be singled out for search before flight?
  - Can people be denied mortgage based on age, sex, race?
  - Women live longer. Should they pay less for life insurance?

Data Mining and Discrimination

- Can discrimination be based on features like sex, age, national origin?
- In some areas (e.g. mortgages, employment), some features cannot be used for decision making
- In other areas, these features are needed to assess the risk factors
  - E.g. people of African descent are more susceptible to sickle cell anemia

Data Mining and Privacy

- Can information collected for one purpose be used for mining data for another purpose
  - In Europe, generally no, without explicit consent
  - In US, generally yes
- Companies routinely collect information about customers and use it for marketing, etc.
- People may be willing to give up some of their privacy in exchange for some benefits

Data Mining with Privacy

- Data Mining looks for patterns, not people!
- Technical solutions can limit privacy invasion
  - Replacing sensitive personal data with anon. ID
  - Give randomized outputs
    - return salary + random()
    - …

Data Mining and Security Controversy in the News

- TIA: Terrorism (formerly Total) Information Awareness Program –
  - DARPA program closed by Congress, Sep 2003
  - some functions transferred to intelligence agencies
- CAPPS II – screen all airline passengers
  - controversial
- …
- Invasion of Privacy or Defensive Shield?

Criticism of analytic approach to Threat Detection:

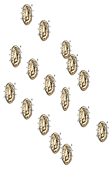
Data Mining will

- invade privacy
- generate millions of false positives

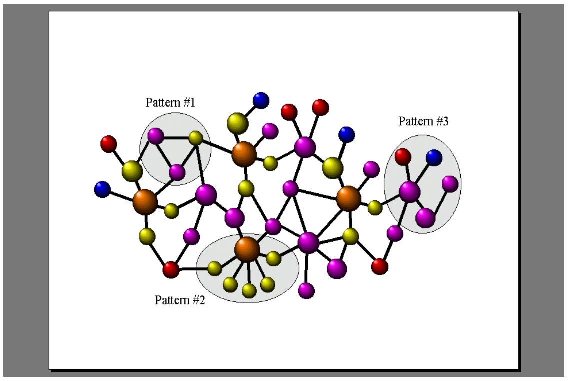
But can it be effective?

Is criticism sound ?

- Criticism: Databases have 5% errors, so analyzing 100 million suspects will generate 5 million false positives
- Reality: Analytical models correlate many items of information to reduce false positives.
- Example: Identify one biased coin from 1,000.
  - After one throw of each coin, we cannot
  - After 30 throws, one biased coin will stand out with high probability.
  - Can identify 19 biased coins out of 100 million with sufficient number of throws



Another Approach: Link Analysis



Can Find Unusual Patterns in the Network Structure

Analytic technology can be effective

- Combining multiple models and link analysis can reduce false positives
- Today there are millions of false positives with manual analysis
- Data mining is just one additional tool to help analysts
- Analytic technology has the potential to reduce the current high rate of false positives

Data Mining and Society

- No easy answers to controversial questions
- Society and policy-makers need to make an educated choice
  - Benefits and efficiency of data mining programs vs. cost and erosion of privacy

Data Mining Future Directions

- Currently, most data mining is on flat tables
- Richer data sources
  - text, links, web, images, multimedia, knowledge bases
- Advanced methods
  - Link mining, Stream mining, …
- Applications
  - Web, Bioinformatics, Customer modeling, …

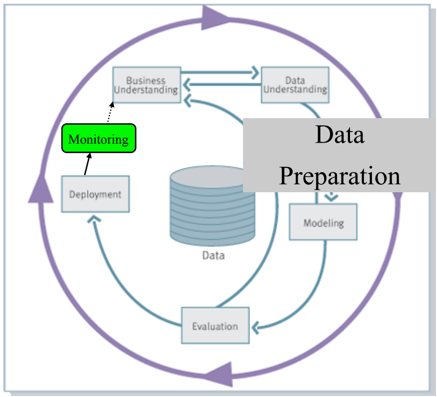
Challenges for Data Mining

- Technical
  - tera-bytes and peta-bytes
  - complex, multi-media, structured data
  - integration with domain knowledge
- Business
  - finding good application areas
- Societal
  - Privacy issues

Data Mining Central Quest

Find true patterns and avoid *overfitting* (false patterns due to randomness)

Knowledge Discovery Process



Start with Business (Problem) Understanding

Data Preparation usually takes the most effort

Knowledge Discovery is an Iterative Process

Key Ideas

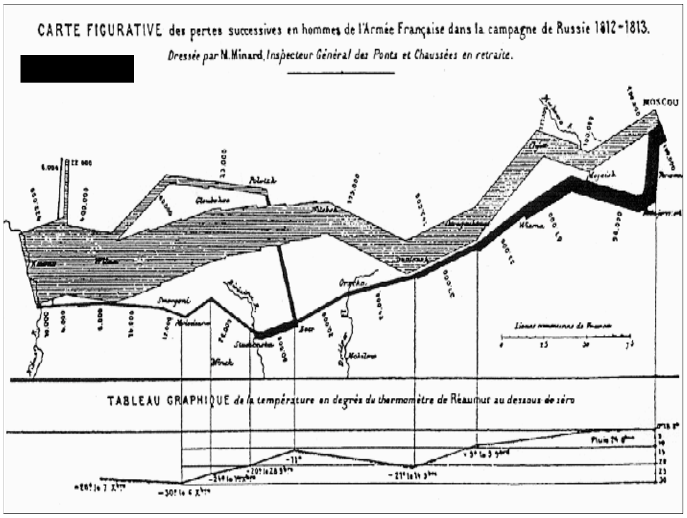
- Avoid Overfitting!
- Data Preparation
  - catch false predictors
  - evaluation: train, validate, test subset
- Classification: C4.5, Bayes, …
- Evaluation: Lift, ROC, …
- Clustering, Association, Other tasks
- Knowledge Discovery is a Process

Visualization and Data Mining

Napoleon Invasion of Russia, 1812

Napoleon

ARCTIC OCEAN
Providenya
Anadyr'
Bering Sea
Kaliningrad
Vyborg
Murmansk
Saint Petersburg
Petrozavodsk
Arkhangel'sk
MOSCOW
Nizhniy Novgorod
Kazan'
Yekaterinburg
Chelyabinsk
Rostov
Novorossiysk
Volgograd
Tuapse
Sochi
Astrakhan'
Omsk
Novosibirsk
Krasnoyarsk
Irkutsk
Noril'sk
Yakutsk
Magadan
Petropavlovsk-Kamchatskiy
NORTH PACIFIC OCEAN
Khabarovsk
Kholmsk
Nevel'sk
Nakhodka
Vostochnyy
Vladivostok
SIBERIA
URAL MTS.
UKRAINE
KAZAKHSTAN
CHINA
JAPAN
MONG.
IRAN
GER.
POLAND
BELARUS
FIN.
SWE.
NOR.
N. KOR.

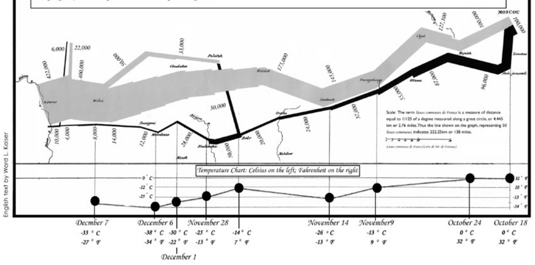


[redacted]

STMpage80 9/12/01 7:43 AM Page 1

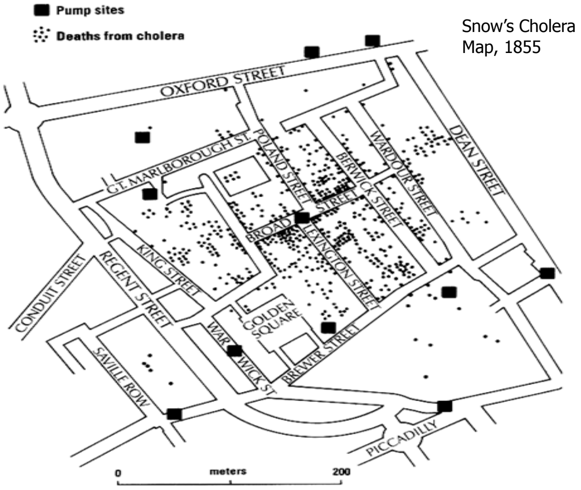
Map representing the losses over time of French army troops during the Russian campaign, 1812-1813.
Constructed by Charles Joseph Minard, Inspector General of Public Works retired.
Paris, 20 November 1869

The number of men present at any given time is represented by the width of the grey line; one mm. indicates ten thousand men. Figures are also written besides the lines. Grey designates men moving into Russia ; black , for those leaving. Sources for the data are the works of messrs. Thiers, Segur, Fezensac, Chambray and the unpublished diary of Jacob. who became an Army Pharmacist on 28 October. In order to visualize the army's losses more clearly, I have drawn this as if the units under prince Jerome and Marshall Davoust (temporarily seperated from the main body to go to Minsk and Mikilow, which then joined up with the main army again), had stayed with the army throughout.



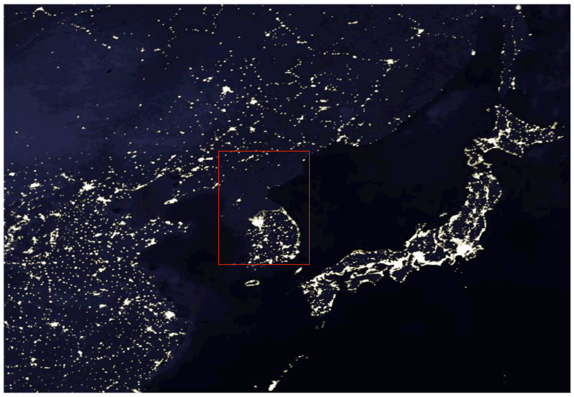
Editor s note: dates & temperatures are only referenced for the retreat from Moscow

Figure 58. ***Minard s map of Napoleon s Russian campaign.***
This graphic has been translated from French to English and modified to most effectively display the temperature data.

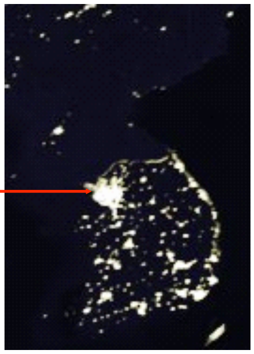


Snow's Cholera
Map, 1855

Asia at night



South and North Korea at night



North Korea
Notice how dark
it is

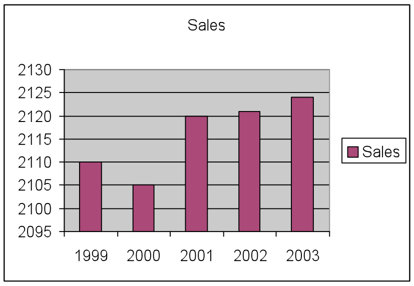
Seoul,
South Korea

Visualization Role

- Support interactive exploration
- Help in result presentation
- Disadvantage: requires human eyes
- Can be misleading

Bad Visualization: Spreadsheet with misleading Y -axis

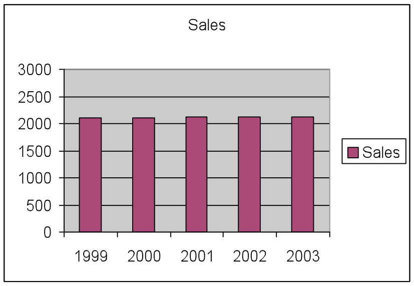
Year	Sales
1999	2110
2000	2105
2001	2120
2002	2121
2003	2124



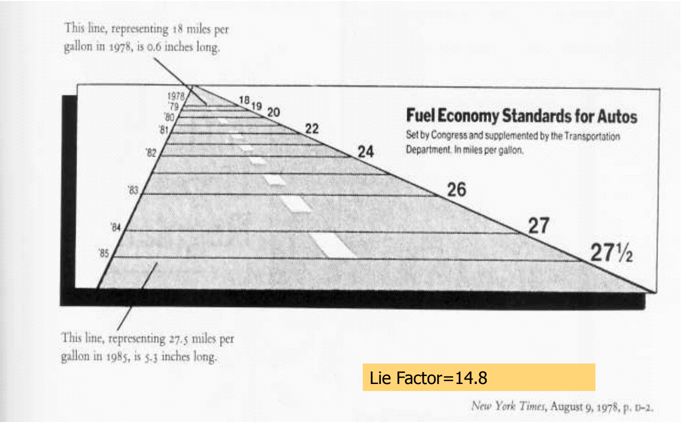
Y-Axis scale gives **WRONG** impression of big change

Better Visualization

Year	Sales
1999	2110
2000	2105
2001	2120
2002	2121
2003	2124



Axis from 0 to 2000 scale gives correct impression of small change



This line, representing 18 miles per gallon in 1978, is 0.6 inches long.

Fuel Economy Standards for Autos

Set by Congress and supplemented by the Transportation Department. In miles per gallon.

This line, representing 27.5 miles per gallon in 1985, is 5.3 inches long.

Lie Factor=14.8

New York Times, August 9, 1978, p. D-2.

(E.R. Tufte, "The Visual Display of Quantitative Information", 2nd edition)

Lie Factor

$$Lie\ Factor = \frac{size\ of\ ef\ f\ ect\ shown\ in\ graphic}{size\ of\ ef\ f\ ect\ in\ data} =$$

$$= \frac{\frac{(5.3 - 0.6)}{0.6}}{\frac{(27.5 - 18.0)}{18}} = \frac{7.833}{0.528} = 14.8$$

Tufte requirement: 0.95<Lie Factor<1.05

(E.R. Tufte, "The Visual Display of Quantitative Information", 2nd edition)

Tufte's Principles of Graphical Excellence

- Give the viewer
  - the greatest number of ideas
  - in the shortest time
  - with the least ink in the smallest space.
- Tell the truth about the data!

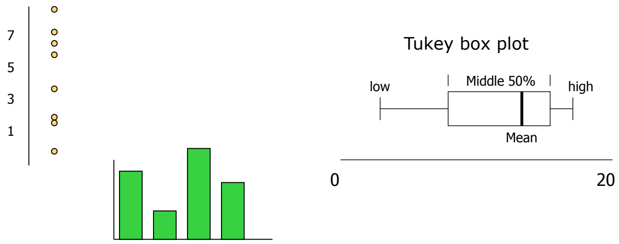
(E.R. Tufte, "The Visual Display of Quantitative Information", 2nd edition)

Visualization Methods

- Visualizing in 1-D, 2-D and 3-D
  - well-known visualization methods
- Visualizing more dimensions
  - Parallel Coordinates
  - Other ideas

1-D (Univariate) Data

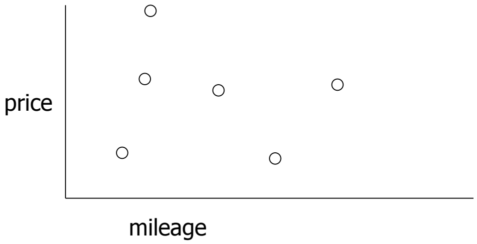
- Representations



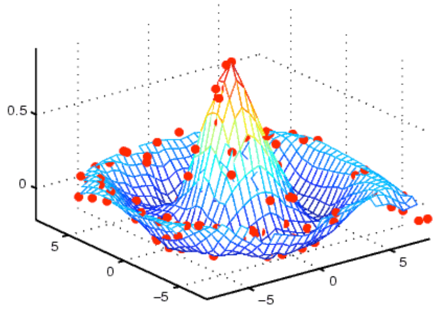
Histogram

2-D (Bivariate) Data

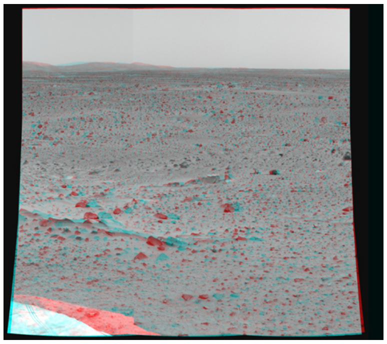
- Scatter plot, …



3-D Data (projection)



3-D image (requires 3-D blue and red glasses)



Taken by Mars Rover Spirit, Jan 2004

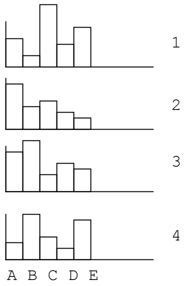
Visualizing in 4+ Dimensions

- Scatterplots
- Parallel Coordinates
- Chernoff faces
- …

Multiple Views

Give each variable its own display

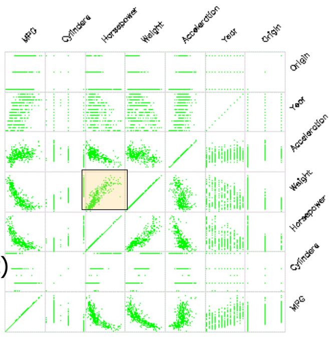
	A	B	C	D	E
1	4	1	8	3	5
2	6	3	4	2	1
3	5	7	2	4	3
4	2	6	3	1	5



Problem: does not show correlations

Scatterplot Matrix

Represent each possible pair of variables in their own 2-D scatterplot (car data)

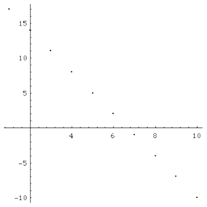


Q*: *Useful for what?
A: linear correlations
(e.g. horsepower & weight)

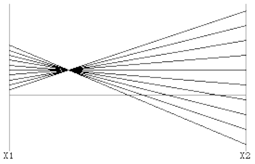
Q: Misses what?
A: multivariate effects

Parallel Coordinates

- Encode variables along a horizontal row
- Vertical line specifies values



Dataset in a Cartesian coordinates



Same dataset in parallel coordinates

Invented by
Alfred Inselberg
while at IBM, 1985

Example: Visualizing Iris Data

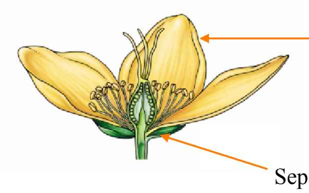
Iris setosa

sepal length	sepal width	petal length	petal width
5.1	3.5	1.4	0.2
4.9	3	1.4	0.2
…	…	…	…
5.9	3	5.1	1.8

Iris versicolor

Iris virginica

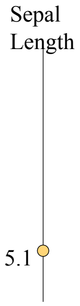
Flower Parts



Petal, a non-reproductive part of the flower

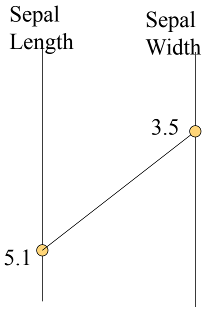
Sepal, a non-reproductive part of the flower

Parallel Coordinates



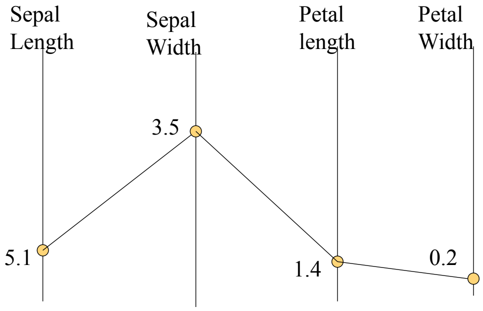
sepal length	sepal width	petal length	petal width
5.1	3.5	1.4	0.2

Parallel Coordinates: 2 D



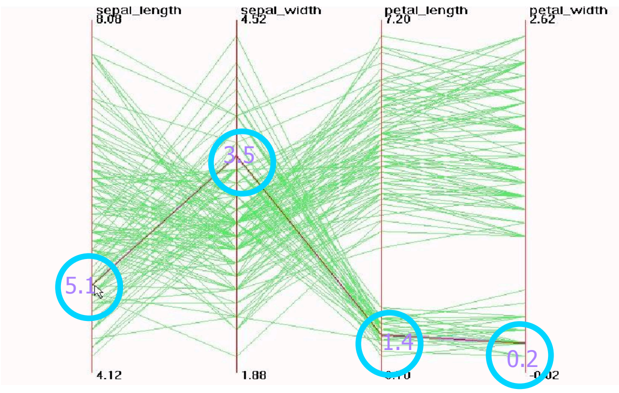
sepal length	sepal width	petal length	petal width
5.1	3.5	1.4	0.2

Parallel Coordinates: 4 D



sepal length	sepal width	petal length	petal width
5.1	3.5	1.4	0.2

Parallel Visualization of Iris data



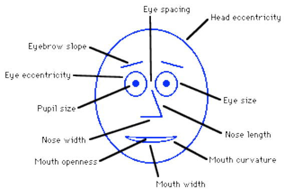
Parallel Visualization Summary

- Each data point is a line
- Similar points correspond to similar lines
- Lines crossing over correspond to negatively correlated attributes
- Interactive exploration and clustering

- Problems: order of axes, limit to ~20 dimensions

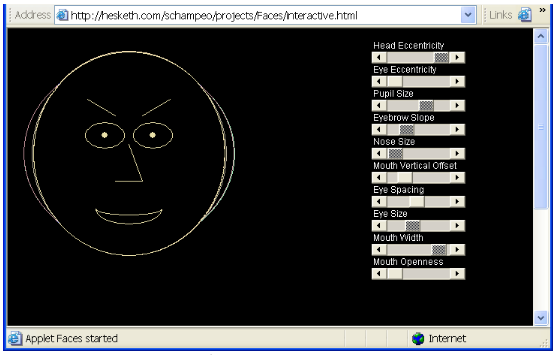
Chernoff Faces

Encode different variables’ values in characteristics of human face



Cute applets: http://www.cs.uchicago.edu/~wiseman/chernoff/
http://hesketh.com/schampeo/projects/Faces/chernoff.html

Interactive Face

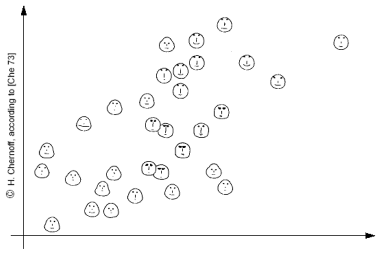


http://hesketh.com/schampeo/projects/Faces/interactive.html

Applet Faces started

Chernoff faces, example

Chernoff-Faces [Che 73, Tuf 83]



Visualization Summary

- Many methods
- Visualization is possible in more than 3-D
- Aim for graphical excellence

Extending linear classification

- Linear classifiers can't model nonlinear class boundaries
- Simple trick:
  - Map attributes into new space consisting of combinations of attribute values
  - E.g.: all products of *n* factors that can be constructed from the attributes
- Example with two attributes and $n = 3$:

$$x = w_1 a_1^3 + w_2 a_1^2 a_2 + w_3 a_1 a_2^2 + w_4 a_2^3$$

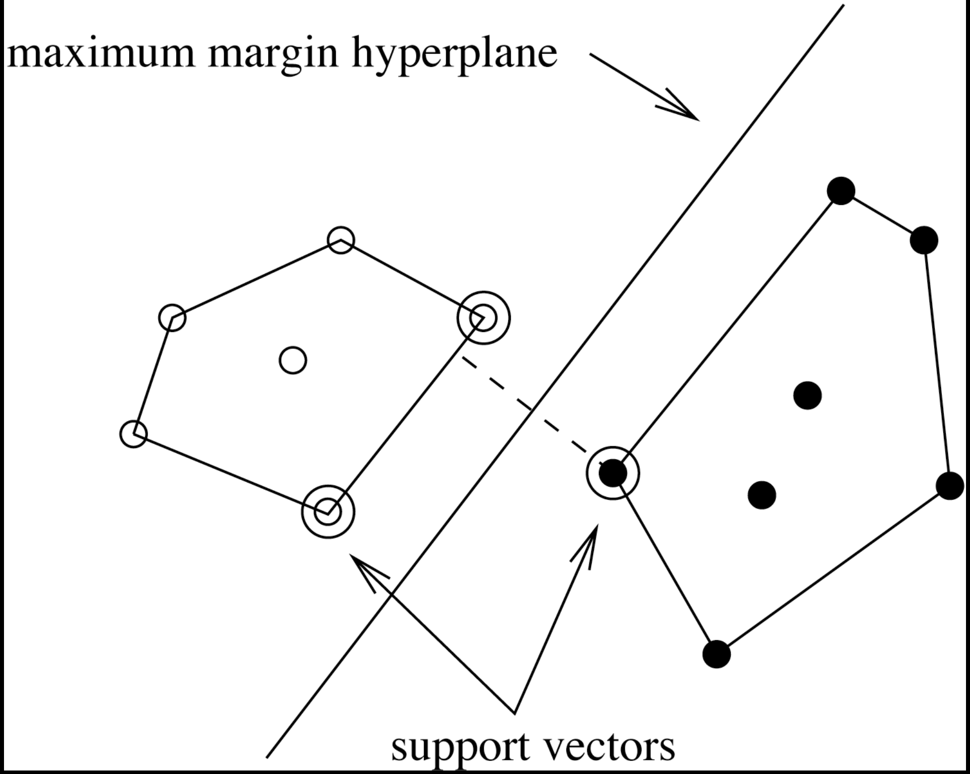
Problems with this approach

- 1st problem: speed
  - 10 attributes, and $n = 5 \Longrightarrow$ 2000 coefficients
  - Use linear regression with attribute selection
  - Run time is cubic in number of attributes
- 2nd problem: overfitting
  - Number of coefficients is large relative to the number of training instances
  - *Curse of dimensionality* kicks in

Support vector machines

- *Support vector machines* are algorithms for learning linear classifiers
- Resilient to overfitting because they learn a particular linear decision boundary:
  - The *maximum margin hyperplane*
- Fast in the nonlinear case
  - Use a mathematical trick to avoid creating "pseudo-attributes"
  - The nonlinear space is created implicitly

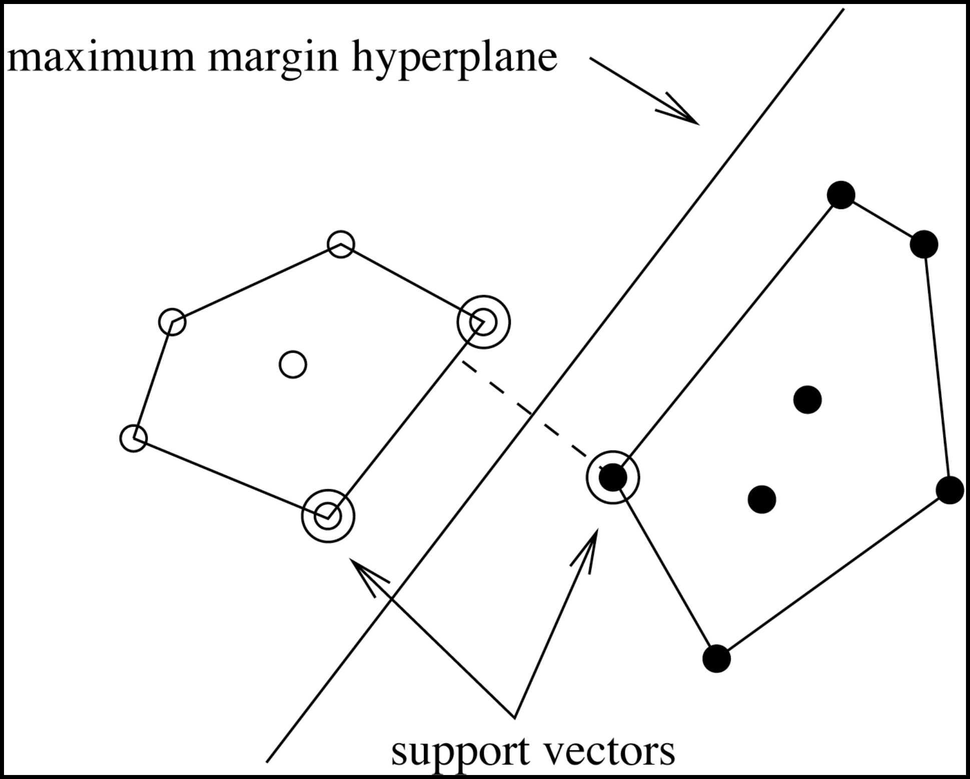
The maximum margin hyperplane



- The instances closest to the maximum margin hyperplane are called *support vectors*

Support vectors

- The support vectors define the maximum margin hyperplane
  - **All other instances can be deleted without changing its position and orientation**



- This means the hyperplane can be written as

$$x = w_0 + w_1 a_1 + w_2 a_2$$

$$x = b + \sum_{\text{i is supp. vector}} \alpha_i y_i \vec{a}(i) \cdot \vec{a}$$

Finding support vectors

$$x = b + \sum_{\text{i is supp. vector}} \alpha_i y_i \vec{a}(i) \cdot \vec{a}$$

- Support vector: training instance for which $\alpha_i > 0$
- Determine α_i and b ?—
  A *constrained quadratic optimization* problem
  - Off-the-shelf tools for solving these problems
  - However, special-purpose algorithms are faster
  - Example: Platt's *sequential minimal optimization* algorithm (implemented in WEKA)
- Note: all this assumes separable data!

Nonlinear SVMs

- “Pseudo attributes” represent attribute combinations
- Overfitting not a problem because the maximum margin hyperplane is stable
  - There are usually few support vectors relative to the size of the training set
- Computation time still an issue
  - Each time the dot product is computed, all the “pseudo attributes” must be included

A mathematical trick

- Avoid computing the “pseudo attributes”
- Compute the dot product before doing the nonlinear mapping
- Example:

$$x = b + \sum_{\text{i is supp. vector}} \alpha_i y_i (\vec{a}(i) \cdot \vec{a})^n$$

- Corresponds to a map into the instance space spanned by all products of *n* attributes

Other kernel functions

- Mapping is called a “kernel function”
- Polynomial kernel

$$x=b+\sum_{\text{i is supp. vector}}\alpha_i y_i(\vec{a}(i)\cdot\vec{a})^n$$

- We can use others:

$$x=b+\sum_{\text{i is supp. vector}}\alpha_i y_i K(\vec{a}(i)\cdot\vec{a})$$

- Only requirement:

$$K(\vec{x}_i,\vec{x}_j)=\phi(\vec{x}_i)\cdot\phi(\vec{x}_j)$$

- Examples:

$$K(\vec{x}_i,\vec{x}_j)=(\vec{x}_i\cdot\vec{x}_j+1)^d$$

$$K(\vec{x}_i,\vec{x}_j)=\exp(\frac{-(\vec{x}_i-\vec{x}_j)^2}{2\sigma^2})$$

$$K(\vec{x}_i,\vec{x}_j)=\tanh(\beta\vec{x}_i\cdot\vec{x}_j+b)$$ *

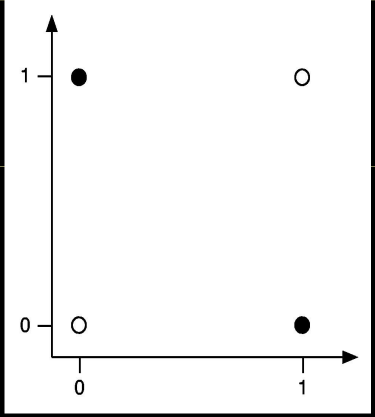
Applications

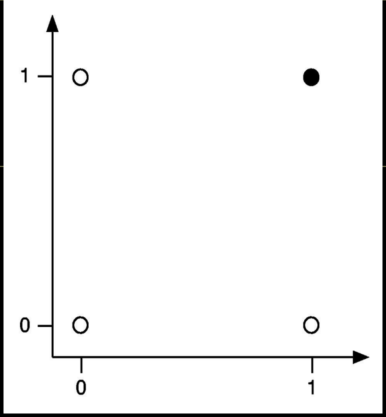
- Machine vision: e.g face identification
  - Outperforms alternative approaches (1.5% error)
- Handwritten digit recognition: USPS data
  - Comparable to best alternative (0.8% error)
- Bioinformatics: e.g. prediction of protein secondary structure
- Text classifiation
- Can modify SVM technique for numeric prediction problems

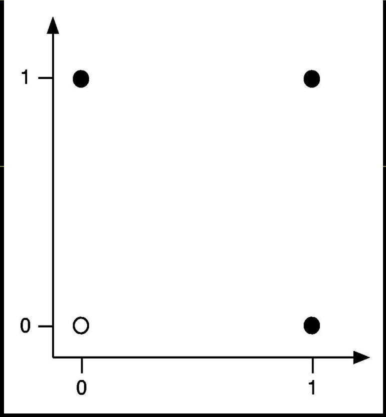
Multilayer perceptrons

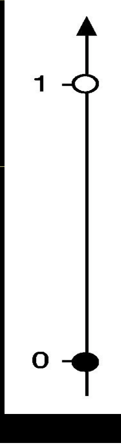
- Using kernels is only one way to build nonlinear classifier based on perceptrons
- Can create network of perceptrons to approximate arbitrary target concepts
- *Multilayer perceptron* is an example of an artificial neural network
  - Consists of: input layer, hidden layer(s), and output layer
- Structure of MLP is usually found by experimentation
- Parameters can be found using *backpropagation*

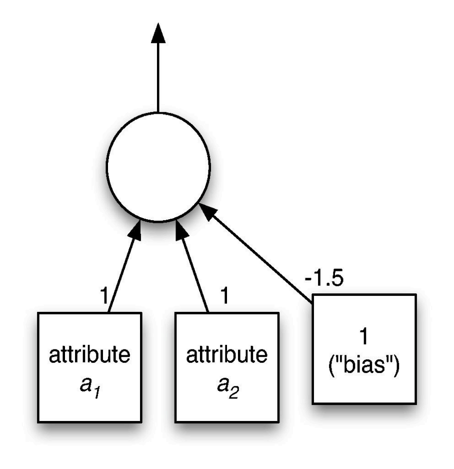
Examples

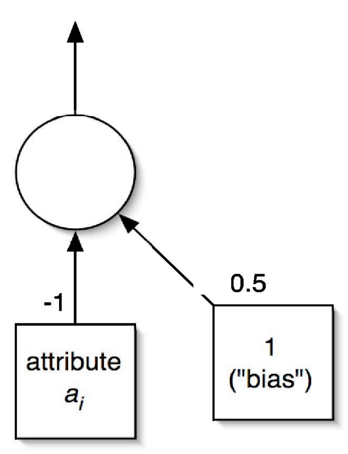

1
0
0
1

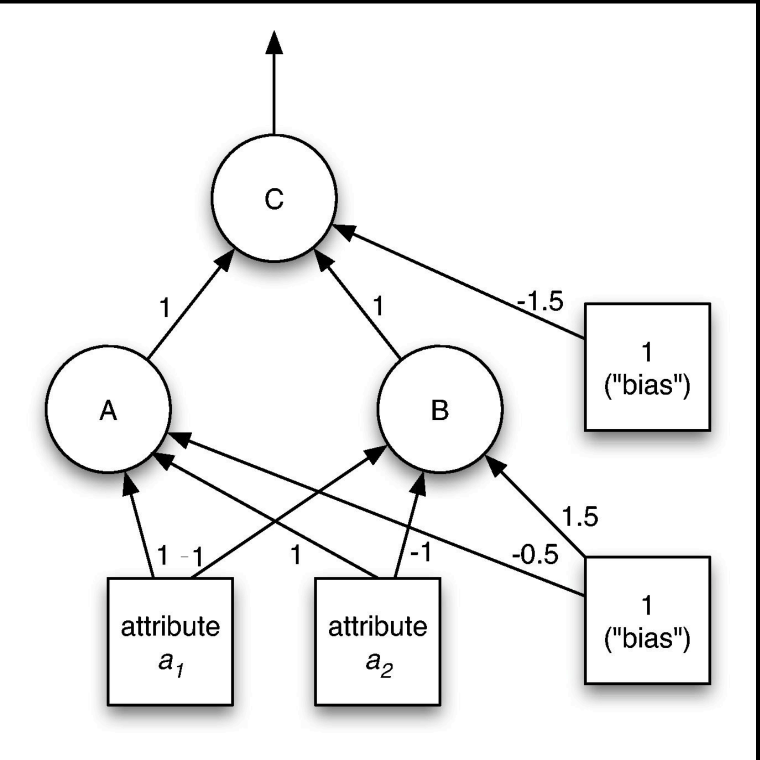

1
0
0
1

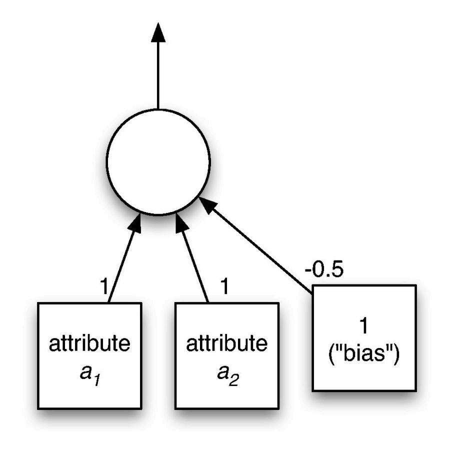

1
0
0
1


1
0


1
1
-1.5
attribute a_1
attribute a_2
1 ("bias")


-1
0.5
attribute a_i
1 ("bias")


C
1
1
-1.5
A
B
1 ("bias")
1.5
1 -1
1
-1
-0.5
attribute a_1
attribute a_2
1 ("bias")


1
1
-0.5
attribute a_1
attribute a_2
1 ("bias")

Backpropagation

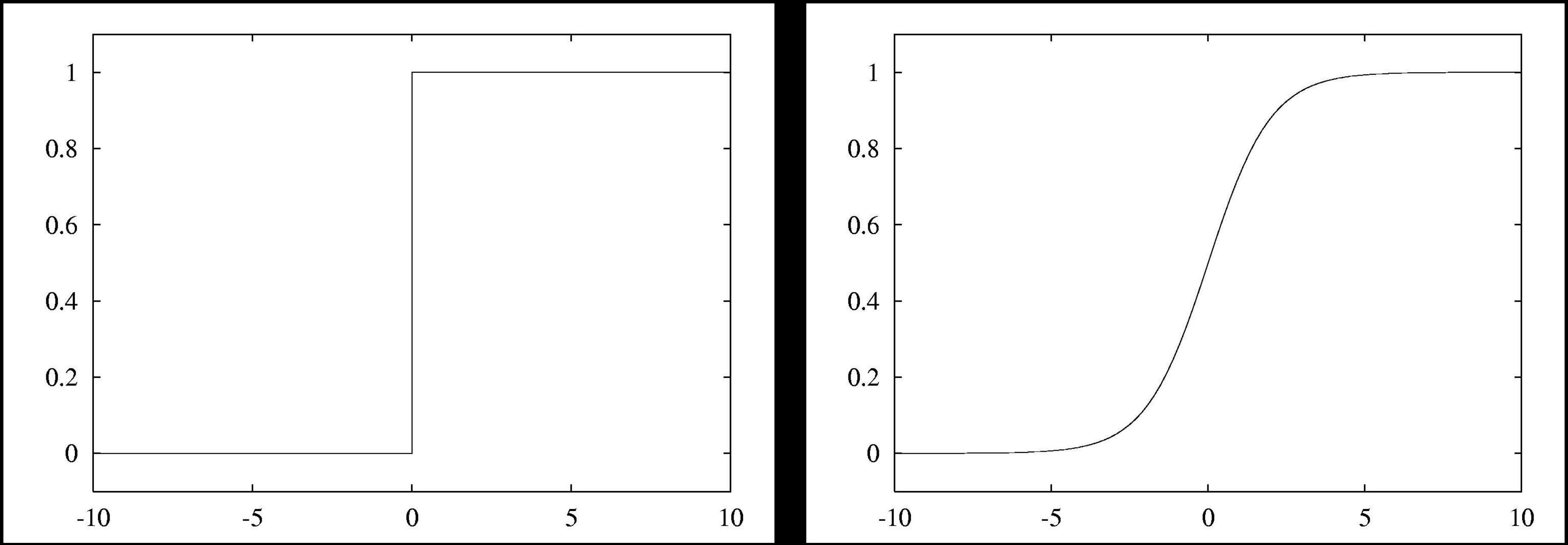
- How to learn weights given network structure?
  - Cannot simply use perceptron learning rule because we have hidden layer(s)
  - Function we are trying to minimize: error
  - Can use a general function minimization technique called *gradient descent*
    - Need differentiable *activation function*: use *sigmoid function* instead of threshold function

$$f(x)=\frac{1}{1+\exp(-x)}$$

    - Need differentiable error function: can't use zero-one loss, but can use squared error

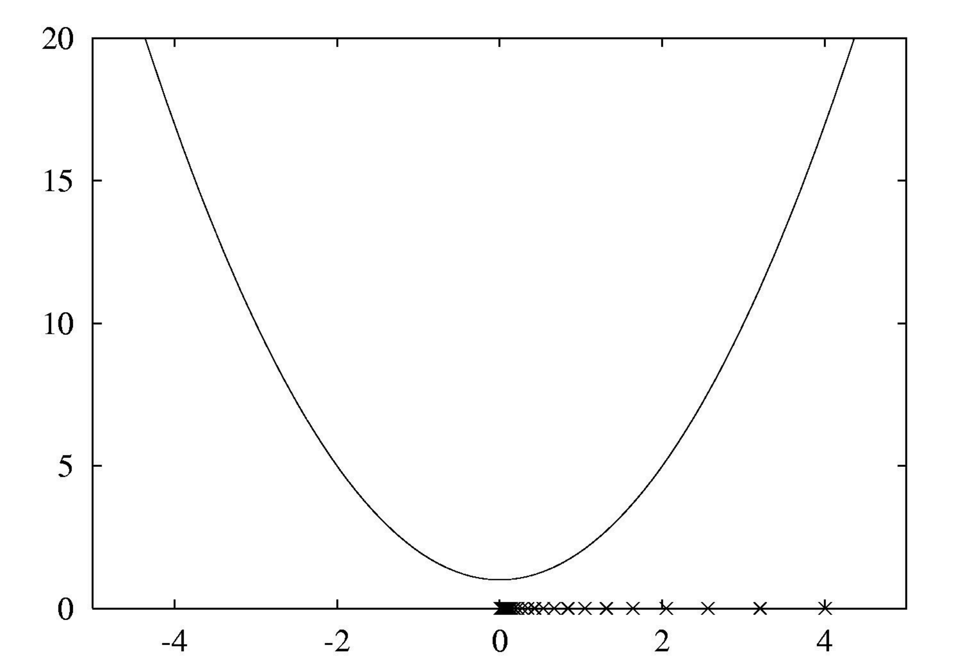
$$E=\frac{1}{2}(y-f(x))^2$$

The two activation functions



Gradient descent example

- Function: x^2+1
- Derivative: $2x$
- Learning rate: 0.1
- Start value: 4



Can only find a local minimum!

Minimizing the error I

- Need to find partial derivative of error function for each parameter (i.e. weight)

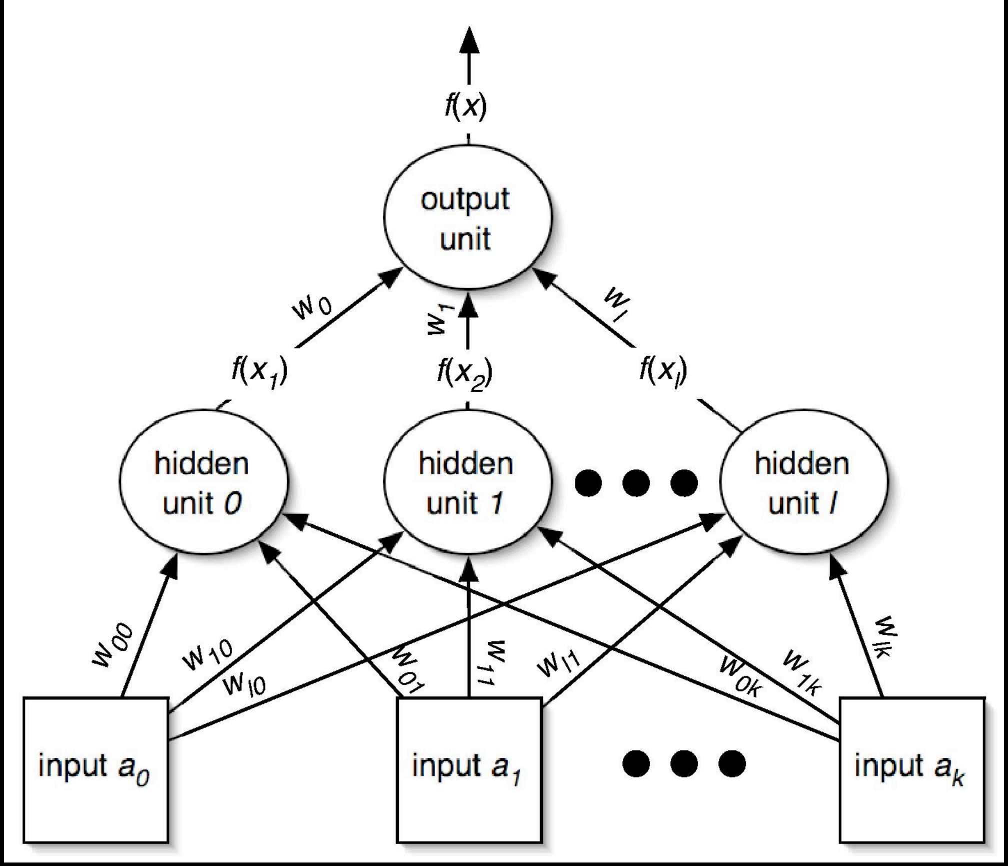
$$\frac{dE}{dw_i}=(y-f(x))\frac{df(x)}{dw_i}$$

$$\frac{df(x)}{dx}=f(x)(1-f(x))$$

$$x=\sum_i w_i f(x_i)$$

$$\frac{df(x)}{dw_i}=f'(x)f(x_i)$$

$$\frac{dE}{dw_i}=(y-f(x))f'(x)f(x_i)$$



Minimizing the error II

- What about the weights for the connections from the input to the hidden layer?

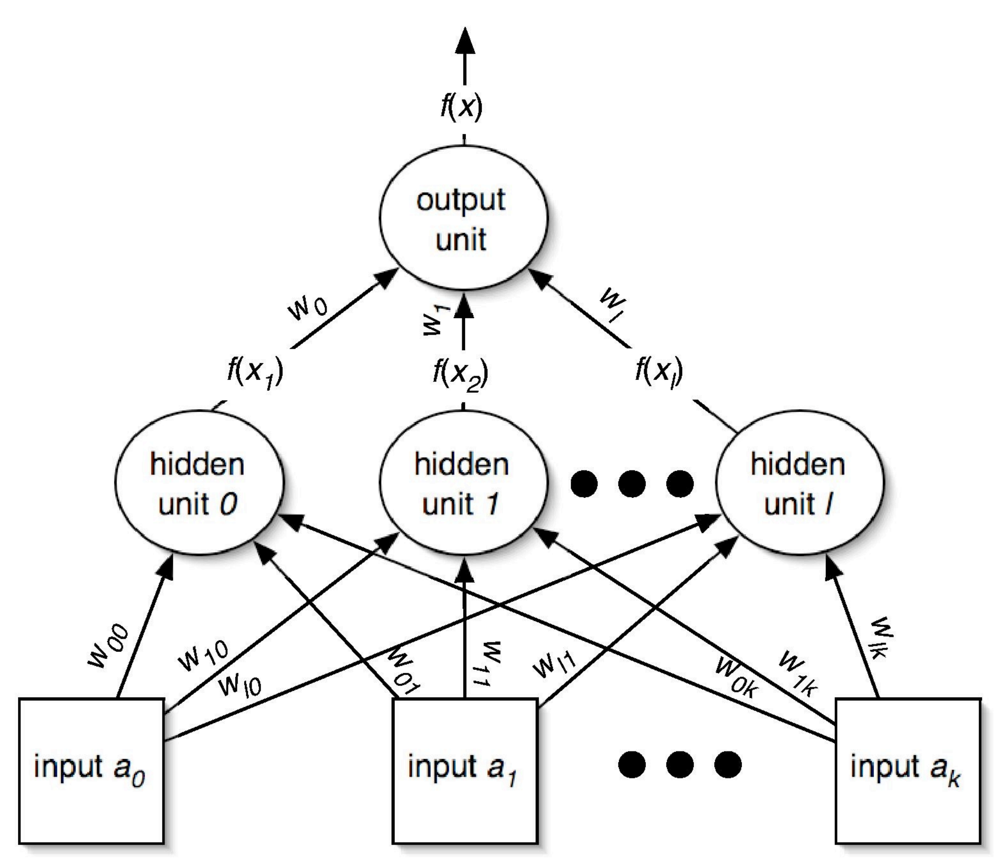
$$\frac{dE}{dw_{ij}} = \frac{dE}{dx}\frac{dx}{dw_{ij}} = (y - f(x))\, f'(x) \frac{dx}{dw_{ij}}$$

$$x = \sum_i w_i f(x_i)$$

$$\frac{dx}{dw_{ij}} = w_i \frac{df(x_i)}{dw_{ij}}$$

$$\frac{df(x_i)}{dw_{ij}} = f'(x_i) \frac{dx_i}{dw_{ij}} = f'(x_i)\, a_i$$

$$\frac{dE}{dw_{ij}} = (y - f(x))\, f'(x)\, w_i f'(x_i)\, a_i$$



Remarks

- Same process works for multiple hidden layers and multiple output units (eg. for multiple classes)
- Can update weights after all training instances have been processed or incrementally:
    - *batch learning* vs. *stochastic backpropagation*
    - Weights are initialized to small random values
- How to avoid overfitting?
    - *Early stopping*: use validation set to check when to stop
    - *Weight decay*: add penalty term to error function
- How to speed up learning?
    - *Momentum*: re-use proportion of old weight change
    - Use optimization method that employs 2nd derivative

Radial basis function networks

- Another type of *feedforward network* with two layers (plus the input layer)
- Hidden units represent points in instance space and activation depends on distance
  - To this end, distance is converted into similarity: Gaussian activation function
    - Width may be different for each hidden unit
  - Points of equal activation form hypersphere (or hyperellipsoid) as opposed to hyperplane
- Output layer same as in MLP

Learning RBF networks

- Parameters: centers and widths of the RBFs + weights in output layer
- Can learn two sets of parameters independently and still get accurate models
    - Eg.: clusters from *k*-means can be used to form basis functions
    - Linear model can be used based on fixed RBFs
    - Makes learning RBFs very efficient
- Disadvantage: no built-in attribute weighting based on relevance
- RBF networks are related to RBF SVMs

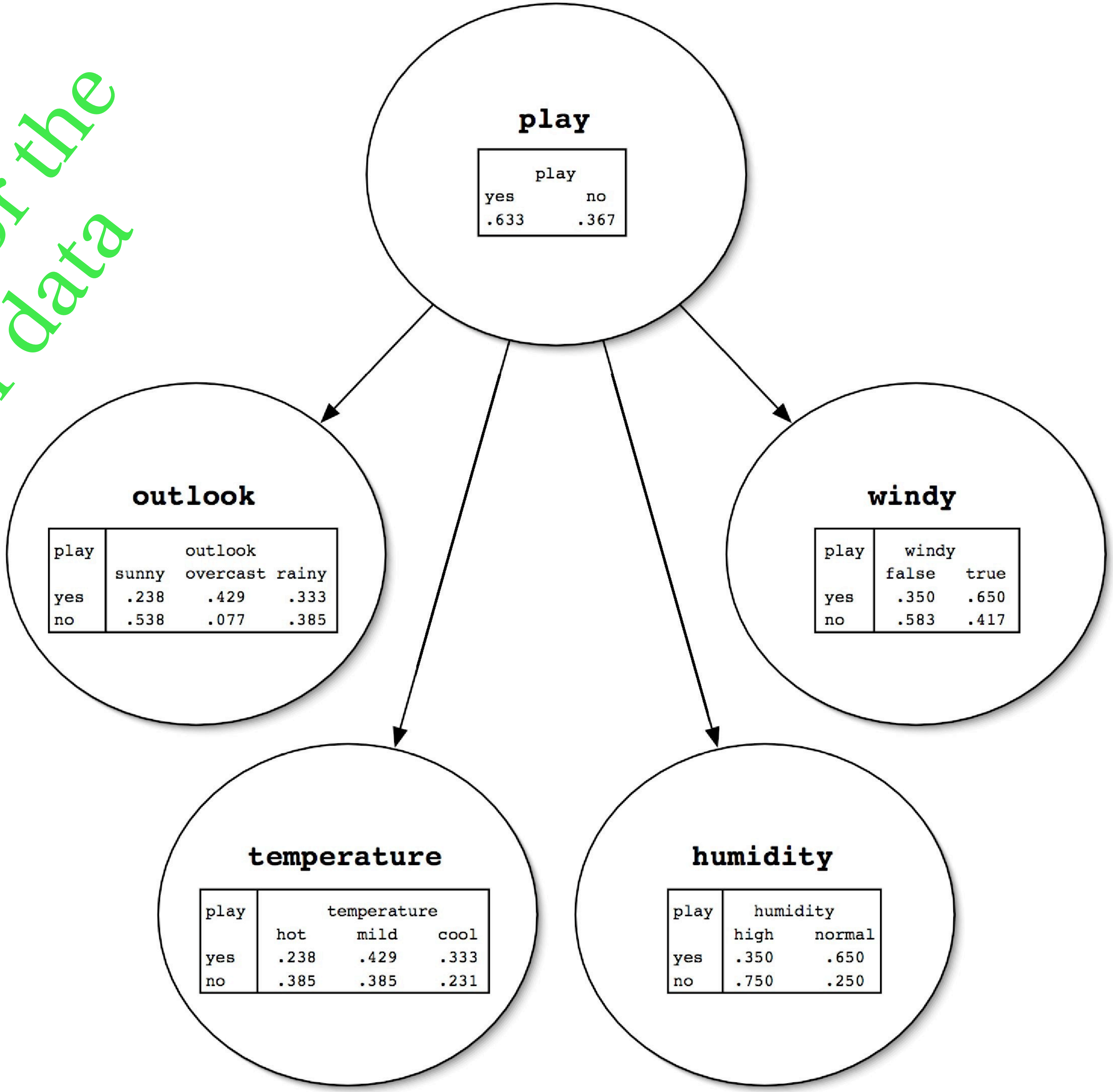
From naïve Bayes to Bayesian Networks

- Naïve Bayes assumes:
  attributes conditionally independent given the class
- Doesn't hold in practice but classification accuracy often high
- However: sometimes performance much worse than e.g. decision tree
- Can we eliminate the assumption?

Enter Bayesian networks

- Graphical models that can represent any probability distribution
- Graphical representation: directed acyclic graph, one node for each attribute
- Overall probability distribution factorized into component distributions
- Graph’s nodes hold component distributions (conditional distributions)

Network for the weather data



play

play	
yes	no
.633	.367

outlook

play	outlook		
	sunny	overcast	rainy
yes	.238	.429	.333
no	.538	.077	.385

windy

play	windy	
	false	true
yes	.350	.650
no	.583	.417

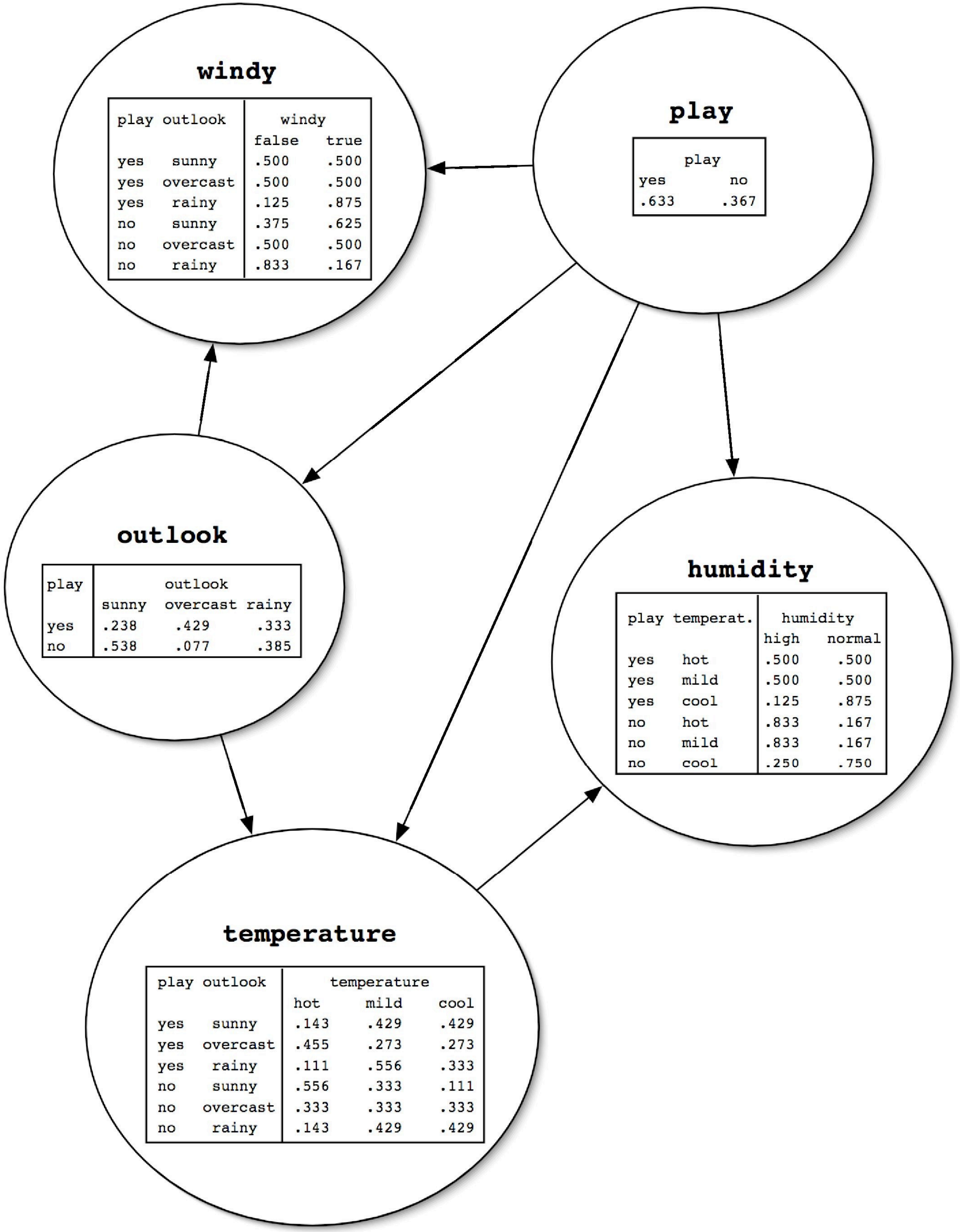
temperature

play	temperature		
	hot	mild	cool
yes	.238	.429	.333
no	.385	.385	.231

humidity

play	humidity	
	high	normal
yes	.350	.650
no	.750	.250

Network for the weather data



windy

play	outlook	windy	
		false	true
yes	sunny	.500	.500
yes	overcast	.500	.500
yes	rainy	.125	.875
no	sunny	.375	.625
no	overcast	.500	.500
no	rainy	.833	.167

play

play	
yes	no
.633	.367

outlook

play	outlook		
	sunny	overcast	rainy
yes	.238	.429	.333
no	.538	.077	.385

humidity

play	temperat.	humidity	
		high	normal
yes	hot	.500	.500
yes	mild	.500	.500
yes	cool	.125	.875
no	hot	.833	.167
no	mild	.833	.167
no	cool	.250	.750

temperature

play	outlook	temperature		
		hot	mild	cool
yes	sunny	.143	.429	.429
yes	overcast	.455	.273	.273
yes	rainy	.111	.556	.333
no	sunny	.556	.333	.111
no	overcast	.333	.333	.333
no	rainy	.143	.429	.429

Computing the class probabilities

- Two steps: computing a product of probabilities for each class and normalization
  - For each class value
    - Take all attribute values and class value
    - Look up corresponding entries in conditional probability distribution tables
    - Take the product of all probabilities
  - Divide the product for each class by the sum of the products (normalization)

Why can we do this? (Part I)

- Single assumption: values of a node’s parents completely determine probability distribution for current node

$$Pr[\text{node}|\text{ancestors}]=Pr[\text{node}|\text{parents}]$$

Means that node/attribute is conditionally independent of other ancestors given parents

Why can we do this? (Part II)

- Chain rule from probability theory:

$$Pr[a_{1,} a_{2,} \ldots , a_n] = \prod_{i=1}^{n} Pr[a_i | a_{i-1} , \ldots , a_1]$$

Because of our assumption from the previous slide:

$$Pr[a_{1,} a_{2,} \ldots , a_n] = \prod_{i=1}^{n} Pr[a_i | a_{i-1} , \ldots , a_1] =$$
$$\prod_{i=1}^{n} Pr[a_i | a_i 'sparents]$$

Learning Bayes nets

- Basic components of algorithms for learning Bayes nets:
  - Method for evaluating the goodness of a given network
    - Measure based on probability of training data given the network (or the logarithm thereof)
  - Method for searching through space of possible networks
    - Amounts to searching through sets of edges because nodes are fixed

Just apply a learner? NO!

- Scheme/parameter selection

  treat selection process as part of the learning process

- Modifying the input:
  - Data engineering to make learning possible or easier
- Modifying the output
  - Combining models to improve performance

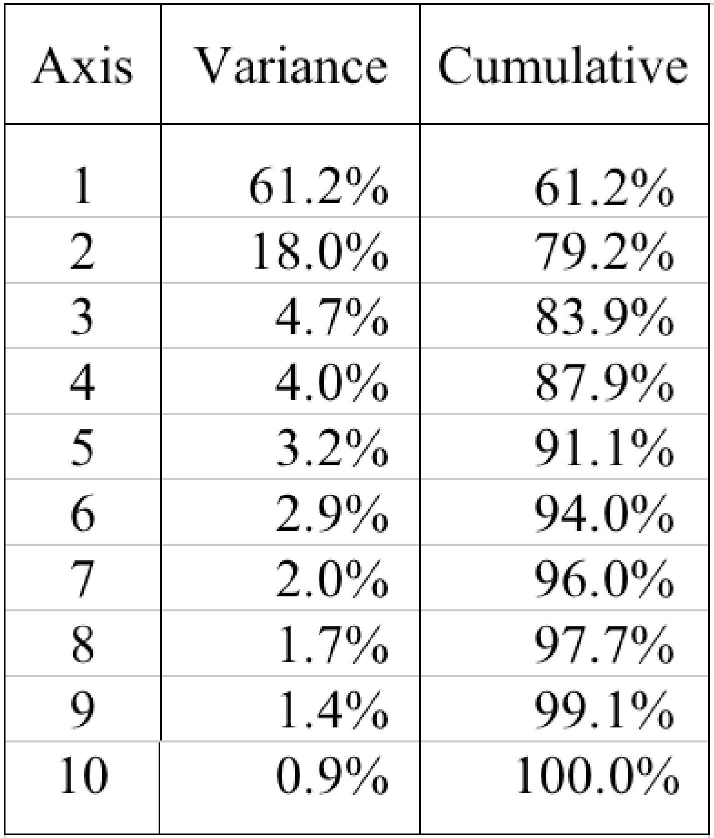
Data transformations

- Simple transformations can often make a large difference in performance
- Example transformations (not necessarily for performance improvement):
  - Difference of two date attributes
  - Ratio of two numeric (ratio-scale) attributes
  - Concatenating the values of nominal attributes
  - Encoding cluster membership
  - Adding noise to data
  - Removing data randomly or selectively
  - Obfuscating the data

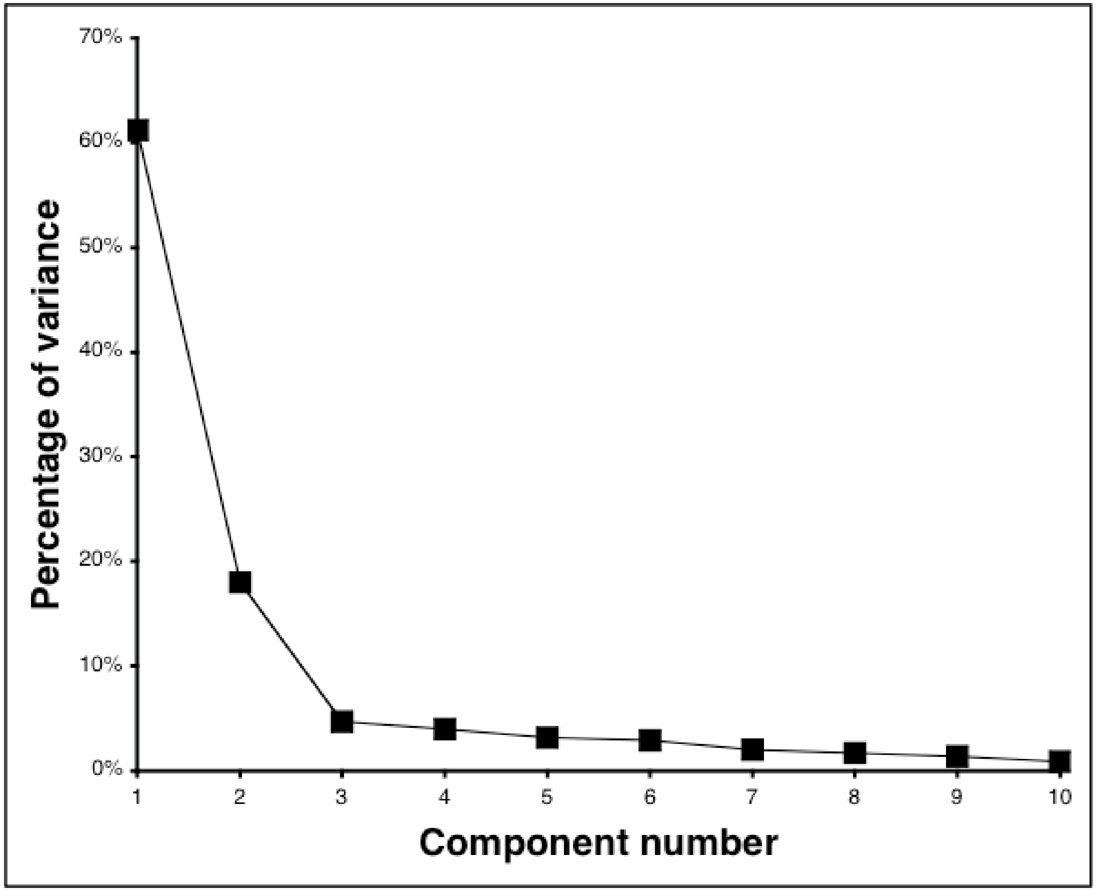
Principal component analysis

- Method for identifying the important “directions” in the data
- Can rotate data into (reduced) coordinate system that is given by those directions
- Algorithm:
    1. Find direction (axis) of greatest variance
    2. Find direction of greatest variance that is perpendicular to previous direction and repeat
- Implementation: find eigenvectors of covariance matrix by diagonalization
    - Eigenvectors (sorted by eigenvalues) are the directions

Example: 10-dimensional data



Axis	Variance	Cumulative
1	61.2%	61.2%
2	18.0%	79.2%
3	4.7%	83.9%
4	4.0%	87.9%
5	3.2%	91.1%
6	2.9%	94.0%
7	2.0%	96.0%
8	1.7%	97.7%
9	1.4%	99.1%
10	0.9%	100.0%



- Can transform data into space given by components
- Data is normally standardized for PCA
- Could also apply this recursively in tree learner

Combining multiple models

- Basic idea:
  build different "experts", let them vote
- Advantage:
  - often improves predictive performance
- Disadvantage:
  - usually produces output that is very hard to analyze
  - but: there are approaches that aim to produce a single comprehensible structure

Bagging

- Combining predictions by voting/averaging
  - Simplest way
  - Each model receives equal weight
- "Idealized" version:
  - Sample several training sets of size *n* (instead of just having one training set of size *n*)
  - Build a classifier for each training set
  - Combine the classifiers' predictions
- Learning scheme is *unstable* ⇒ almost always improves performance
  - Small change in training data can make big change in model (e.g. decision trees)

Bias-variance decomposition

- Used to analyze how much selection of any *specific* training set affects performance
- Assume infinitely many classifiers, built from different training sets of size *n*
- For any learning scheme,
  - *Bias* = expected error of the combined classifier on new data
  - *Variance*= expected error due to the particular training set used
- Total expected error ≈ bias + variance

More on bagging

- Bagging works because it reduces *variance* by voting/averaging
  - Note: in some pathological hypothetical situations the overall error might increase
  - Usually, the more classifiers the better
- Problem: we only have one dataset!
- Solution: generate new ones of size *n* by sampling from it *with replacement*
- Can help a lot if data is noisy
- Can also be applied to numeric prediction
  - Aside: bias-variance decomposition originally only known for numeric prediction

Bagging classifiers

Model generation

```
Let $n$ be the number of instances in the training data
For each of $t$ iterations:
  Sample $n$ instances from training set
    (with replacement)
  Apply learning algorithm to the sample
  Store resulting model
```

Classification

```
For each of the $t$ models:
  Predict class of instance using model
Return class that is predicted most often
```

Randomization

- Can randomize learning algorithm instead of input
- Some algorithms already have a random component: eg. initial weights in neural net
- Most algorithms can be randomized, eg. greedy algorithms:
  - Pick from the *N* best options at random instead of always picking the best options
  - Eg.: attribute selection in decision trees
- More generally applicable than bagging: e.g. random subsets in nearest-neighbor scheme
- Can be combined with bagging

Boosting

- Also uses voting/averaging
- Weights models according to performance
- Iterative: new models are influenced by performance of previously built ones
    - Encourage new model to become an “expert” for instances misclassified by earlier models
    - Intuitive justification: models should be experts that complement each other
- Several variants

AdaBoost.M1

Model generation

```
Assign equal weight to each training instance
For $t$ iterations:
   Apply learning algorithm to weighted dataset,
     store resulting model
   Compute model's error $e$ on weighted dataset
   If $e = 0$ or $e \geq 0.5$:
     Terminate model generation
   For each instance in dataset:
     If classified correctly by model:
        Multiply instance's weight by $e/(1-e)$
   Normalize weight of all instances
```

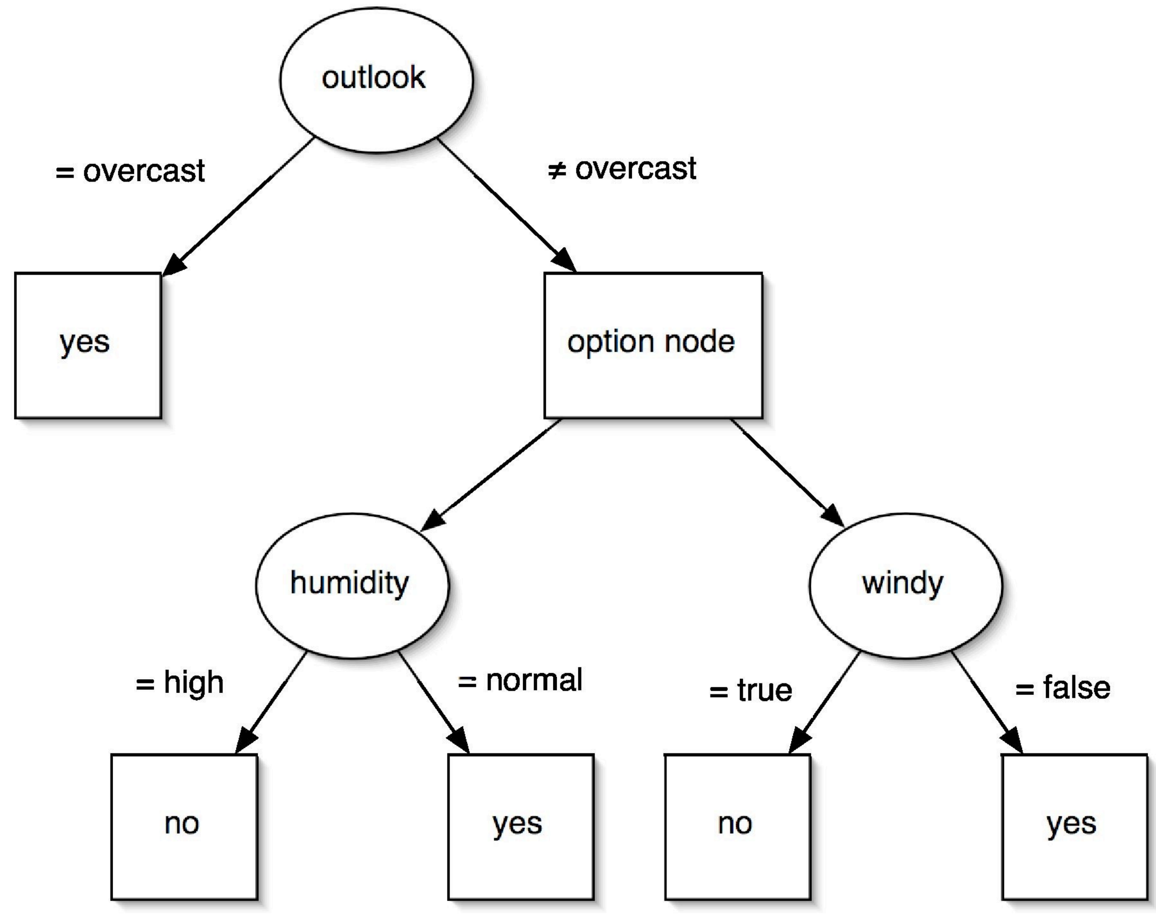
Classification

```
Assign weight = 0 to all classes
For each of the $t$ (or less) models:
   For the class this model predicts
     add $-\log e/(1-e)$ to this class's weight
Return class with highest weight
```

More on boosting I

- Boosting needs weights … but
- Can adapt learning algorithm ... or
- Can apply boosting *without* weights
  - resample with probability determined by weights
  - disadvantage: not all instances are used
  - advantage: if error > 0.5, can resample again
- Stems from *computational learning theory*
- Theoretical result:
  - training error decreases exponentially
- Also:
  - works if base classifiers are not too complex, and
  - their error doesn't become too large too quickly

Example



- Can be learned by modifying tree learner:
  - Create option node if there are several equally promising splits (within user-specified interval)
  - When pruning, error at option node is average error of options

Stacking

- To combine predictions of base learners, don’t vote, use *meta learner*
  - Base learners: *level-0 models*
  - Meta learner: *level-1 model*
  - Predictions of base learners are input to meta learner
- Base learners are usually different schemes
- Can’t use predictions on training data to generate data for level-1 model!
  - Instead use cross-validation-like scheme
- Hard to analyze theoretically: “black magic”

More on stacking

- If base learners can output probabilities, use those as input to meta learner instead
- Which algorithm to use for meta learner?
    - In principle, any learning scheme
    - Prefer “relatively global, smooth” model
        - Base learners do most of the work
        - Reduces risk of overfitting
- Stacking can be applied to numeric prediction too