

Managing Big Data

Preliminaries: About Algorithms and Data

Manolis Tzagarakis
Assistant Professor
Department of Economics
University of Patras

tzagara@upatras.gr
2610 969845
google:tzagara
Facebook: tzagara
SkypeID: tzagara
QuakeLive: DeusEx
CoD: CoDFather

Asymptotic analysis of algorithms

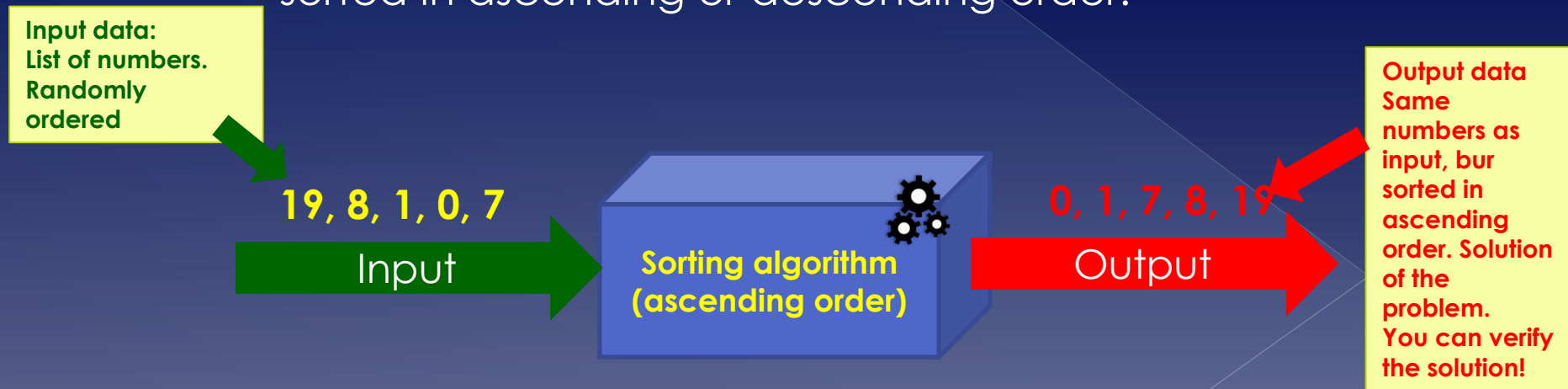
- ⦿ Computers are machines that through programs can solve many problems including
 - > Sorting (ascending/descending) of numbers, strings, dates etc.
 - > Finding the roots of polynomials like $x^7+3x^4+2=0$ e.g. by applying the Newton-Raphson method
 - > Solving system of linear equations e.g. by using the Gauss-Jordan elimination
 - > Calculate functions such as Sine, Cosine, Logarithms etc using e.g. Taylor and McLaurin Series
 - > ...and many more (but not all ☹)

Asymptotic analysis of algorithms

- ◎ Steps to solve a problem using computers
 - > **Write** the appropriate program
 - E.g. To find the roots of some polynomial, write a program that implements the Newton-Raphson method
 - > **Give** the program the appropriate input i.e. required data
 - > **Execute** the program on the computer with the data provided
 - > After some time, **the program will emit (on screen/disk) the solution** to the problem (output)
 - E.g. in case of finding the roots of a polynomials, it will show the roots. You can verify the solution, i.e. roots
 - > **You can always verify the solution**

Asymptotic analysis of algorithms

- ◉ In our context «Program = Algorithm»
 - > What's an algorithm? = a well defined order of commands and operations (steps) that when executed by machines called computers and given the appropriate input can solve a particular problem on it's output.
 - «**Solve a problem**»: the algorithm on it's output will produce data which are the solution to the problem (as it has been defined).
 - Example: Algorithm which sorts numbers: it's a set of commands/operations that when given as input a random list of numbers will produce as output the same numbers sorted in ascending or descending order.



Asymptotic analysis of algorithms

- ⦿ There **exist algorithms** that solve many, many problems:
 - > Calculate the average of a set of numbers
 - > Calculate sine, cosine, logarithm of a number
 - > Find min, max, median, stdev etc from a set of numbers
 - > Find the roots of a polynomial
 - > Find solution(s) of a system of linear equations
 - > Find a particular character in a string
 - > Find webpages that contain the word “economics”
 - > ...and many, many more
- ⦿ Problems for which there exist an algorithm that solve them (correctly) are called **“Decidable”** 😊

Asymptotic analysis of algorithms

- ◉ But not all! Unfortunately there are some problems for which there exist NO algorithm that can solve them. E.g.
 - > **“Halting problem”**: whether or not an algorithm with some input will terminate or not (i.e. crash). Such algorithm (currently) does not exist.
 - > **“Matrix mortality problem”**: Given a finite set of $n \times n$ matrices, is there a way of multiplying them (with or without repetition) that will result in the zero matrix? Such algorithm (currently) does not exist.
- ◉ Problems for which there are no algorithms that can solve them are called **“Undecidable”** ☹️

Asymptotic analysis of algorithms

Example of algorithm?

Simple algorithm that finds the greatest number in an array of numbers.

Input: Array with 10 random numbers

Output: The greatest number (max) contained in the input numbers

Order of execution

Step 1: Read 10 numbers from user (keyboard) and insert these numbers in an array T such that T[1] contains the first number given and T[10] the last number given

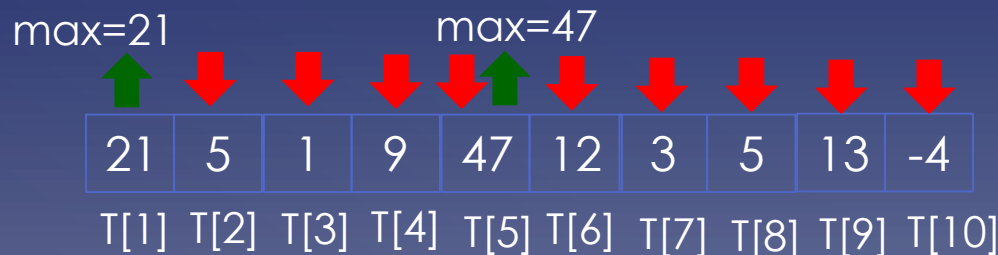
Step 2: Assume that the greatest number is the number in the first position of the array i.e. T[1]. Store this number in a variable called max: max=T[1].

Step 3: For each position of array T, starting from 2 up until position 10 of the array T do the following:

Step 4: Compare number in current array position with max. If current position of array has number greater than max then:

Step 5: Store in variable max the number in current position of array

Step 6: Output on screen the variable max. This will be the greatest number in array T.



↓ = Compare with max

↑ = Replace max.
Found new max

Asymptotic analysis of

- algorithms
- Some algorithm in a real programming language (Python)

Simple algorithm that finds the greatest number in an array of numbers.

Input: Array with 10 random numbers

Output: The greatest number (max) contained in the input numbers

#Array T will hold the numbers

```
T=[]
```

Step 1

#Ask for 10 numbers from user.

```
for i in range (0, 10):
```

Step 2

```
    number = int (input("Give me the number: ") )
```

Step 3

```
    T.append(number)
```

Step 4

#Ok. Got 10 numbers. Do your thing now.

#Set as max the first number. In python, array indices start at 0, not 1!

```
currMax = T[0]
```

Step 5

```
i=1
```

Step 6

```
while (i < len(T)):
```

Step 7

```
    #Is current position T[i] greater than currMax?
```

```
    if ( T[i] >= currMax ):
```

Step 8

```
        #Yes, it is! Found a new max value
```

```
        currMax = T[i]
```

Step 9

```
        #advance to next element in array
```

```
        i = i + 1
```

Step 10

```
print("The result is: ", currMax)
```

Step 11

Order of
execution



Asymptotic analysis of algorithms

- ⦿ An algorithm can be represented in many ways (form does not matter)
 - > Pseudocode (first example)
 - > Programming language (second example)
- ⦿ ...as long as
 - > Specifies concrete steps
 - > Takes well defined input
 - > Each step can be executed by the machine (computer)
 - > Solves (correctly) the problem at the output

Asymptotic analysis of algorithms

- ⦿ For a given problem there might be two or more different algorithms that solve the problem
 - > Different? They execute different set of steps
- ⦿ For example we can come up with two different algorithms for finding the greatest number in a list
 - > Earlier example

Asymptotic analysis of algorithms

Two different algorithms solving the “find greatest number” problem

First algorithm (see previously)

Step 1: Set as max the first element of the list

Step 2: Compare other elements with current max

Step 3: If current max is smaller than current element, set current element as max

Step 4: If all elements of list have been compared, output max. This will be the greatest number. Output it and terminate.

Second algorithm

Step 1: Compare first element with all other elements of table to see if this is greater than all other elements

Step 2: If it is, then output first element as greatest and terminate.

Step 3: If first element is not the greatest, compare second element to all other elements to see if this is greater than all other elements

Step 4: If it is, then output second element as greatest and terminate.

...

Step 19: Display last element of list as the largest number (no need to compare).

Both algorithms solve the problem (correctly)!

Asymptotic analysis of algorithms

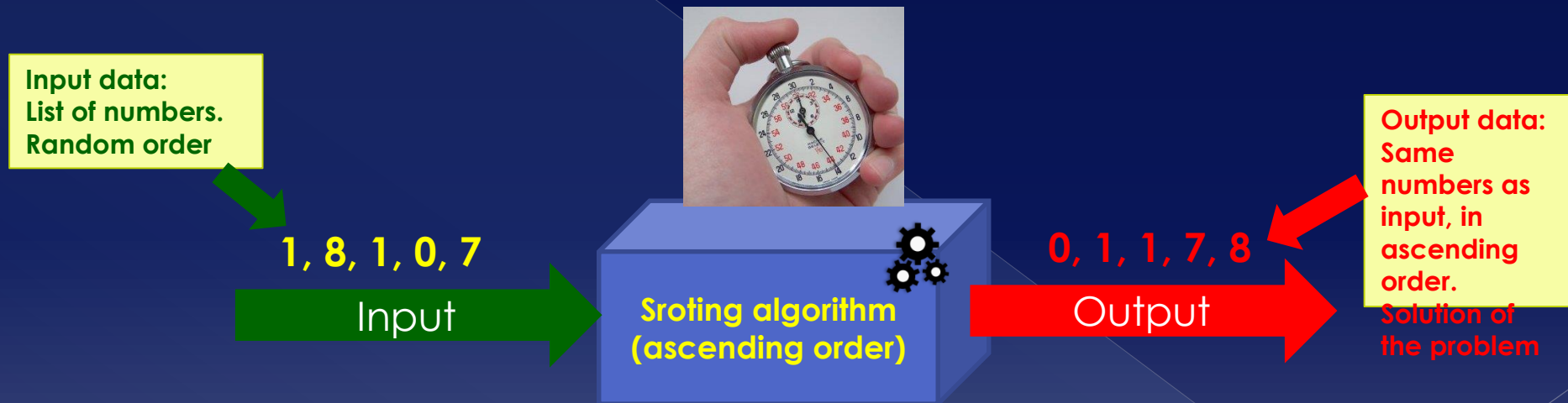
- Many different algorithms that solve the same problem
 - > E.g. Sorting algorithms (numbers, strings, dates etc):
 - **BubbleSort**
 - **QuickSort**
 - **HeapSort**
 - **MergeSort**

Asymptotic analysis of algorithms

- ◉ Since there might be different algorithms that solve the same problem, **which algorithm is better?**
 - > **What does “better” mean?** Here it means “**faster**” i.e. which algorithm solves a given problem in the shortest time (= Execution time of algorithm)
 - Many different interpretations of “better”: in terms of execution time, space used etc.
 - > Of course we are interested in solving the problem quickly
 - An algorithm A solving a problem in less time than algorithm B, is faster and hence in our context is better.
- ◉ To answer such questions, we need to answer another important and related question:
 - > **How to measure the execution time of algorithms? What is the best metric to assess the time needed by an algorithm to solve a problem?**
 - One approach: use real time (measured in min, sec, msec, ns etc)
 - . Measuring the real time elapsed: when the algorithm starts executing until it produces the solution of the problem on the output and terminates

Asymptotic analysis of algorithms

- Is time (measured in min, sec, msec etc) a good metric for assessing the execution speed of algorithms?



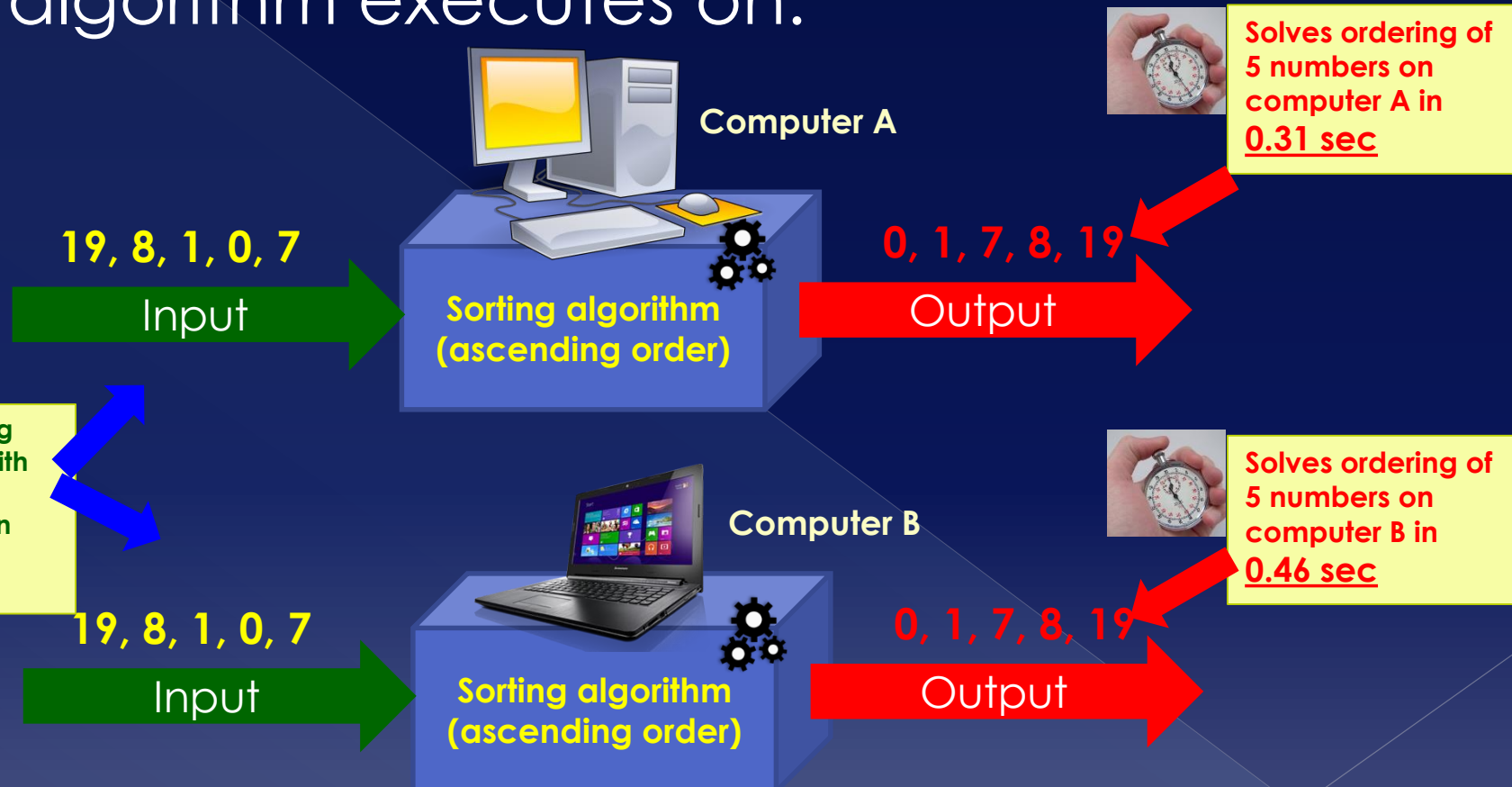
- It's one (valid) approach. Used today in many practical situations.

Asymptotic analysis of algorithms

- ◉ But, using the (real) time as a metric to assess running time of algorithms, **is not the best idea**. This, for two reasons:
 - > Measured that way, running time **depends on the computer/machine** the algorithm runs on
 - > Time does not tell us how the running time of an algorithm is affected, **when the size of input data changes**
- ◉ Using time as the metric, makes it difficult to **compare algorithms in a systematic way**.

Asymptotic analysis of algorithms

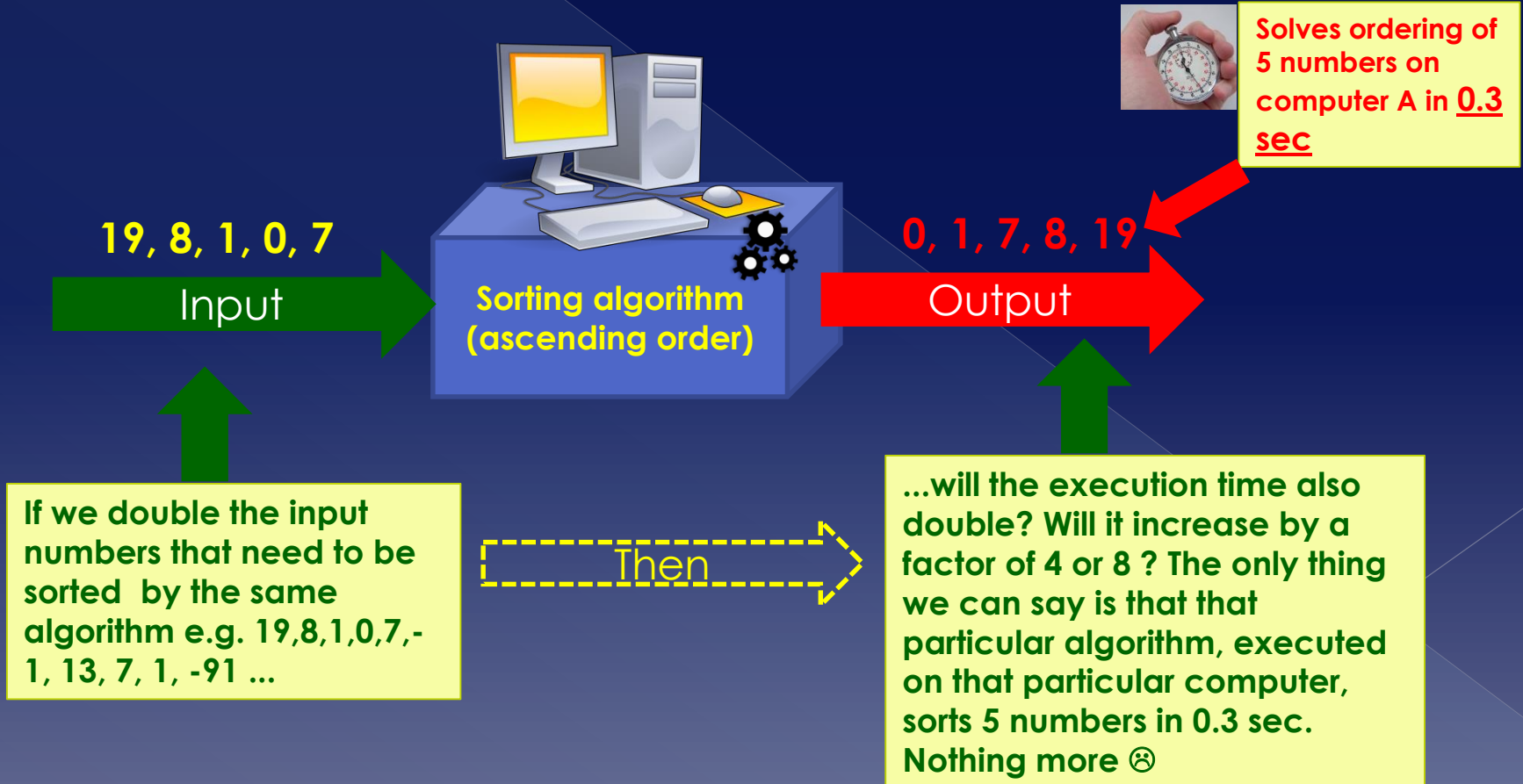
- 1) Time depends on computer the algorithm executes on.



What is the execution time of the algorithm (same in both situations)?
The average? What if the same algorithm executes on another computer and sorts these 5 numbers in say 0.08 secs?

Asymptotic analysis of algorithms

- 2) Time does not tell us what happens, when input data changes e.g. is increased in size.



Asymptotic analysis of algorithms

- ⦿ Time is not an appropriate metric for assessing the speed of algorithms
 - > Can't compare different algorithms that solve the same problems
 - > In practice though, time is used under very specific circumstances and as an indication of the efficiency of an algorithm
- ⦿ Use a different metric: **count the number of steps** an algorithm performs/executes until it finds the solution.
 - > This is today the established way to assess the running time of algorithms!
 - > Don't forget: an algorithm is a finite number of steps that execute and solve a problem
 - > The idea is that an **algorithm A** who solves a problem is faster than **algorithm B** who solves the same problem, **when A executed fewer number of steps (commands/operations).**

Asymptotic analysis of algorithms

- How to express the number of steps executed?
 - > As a function T of a variable called the “problem size” : $T(n)$.
 - > **Problem size** = An aspect of the input data, that definitely will affect the number of steps or how many times the steps will be executed, until the algorithm finds the solution to the problem
 - Example: In the context of a sorting algorithm, sorting 10 numbers (input data) will require each step of the algorithm to be executed more times than say the sorting of 8 number by the same algorithm (more comparisons). Hence, **Problem size for sorting algorithms = Count of numbers to be sorted (more numbers => more steps)**
 - Example: in the context of an algorithm which finds the greatest from a list of numbers, the count of numbers given as input will affect the times each step will be executed and hence its running time. Hence, **Problem size for finding max = Count of numbers in the input list (more numbers => more comparisons)**
 - Example: in an algorithm that searches if character “a” is contained in a string (say “John” or “Nabuzennezor”), how many times each step is executed depends on the length of the input string (bigger strings will result in more execution of steps). Hence, **Problem size for fining char in string = the length of the input string (more chars in string => more comparisons)**

Asymptotic analysis of algorithms

- Examples of running times expressed as a function T of the problem size, denoted as n :
 - > Number of operations of algorithm, T : **$T(n) = 2n, n > 0$**
 - This means that when given an input of size n , then algorithm will do **$2n$ operations** until it outputs the solution.
 - > Number of operations of algorithm, T : **$T(n) = n^2 + 3$**
 - If given input of size n , algorithm will make **n^2+3 operations** until it solves the problem.
- From now on: Number of operations of algorithm = running time of algorithm
 - > We say: running time of algorithm **$T(n) = 2n, T(n) = n^3+n+4, T(n) = 2^n$** , etc.

Asymptotic analysis of algorithms

● Example of computing running time, $T(n)$

Simple algorithm to find the biggest from a list of 10 numbers. 10 numbers provided by user.

Input: 10 random numbers.

output: the greatest number in input set.

/* generalization: assume that we ask for n numbers*/

```
T=[]
for i in range (0, 10):
    number = int (input("Give me the number: ") )
    T.append(number)
```

#Assume first number as greatest

```
currMax = T[0]
i=1
while (i < len(T)):
    #current number at T[i] greater than current maximum?
    if ( T[i] >= currMax ):
        #Yup! Found a bigger number
        currMax = T[i]
    i = i + 1

print("The result is: ", currMax)
```

Step 1 Executed 1 time
Step 2 Executed n+1 times
Step 3 Executed n times
Step 4 Executed n times

Step 5 Executed 1 time
Step 6 Executed 1 time
Step 7 Executed n+1 times

Step 8 Executed n times

Step 9 Executed n times
Step 10 Executed n times

Step 11 Executed 1 time

+

Running time: $T(n)=7n+6$

Asymptotic analysis of algorithms

- Calculating running time $T(n)$ also referred to as **calculating exact time**
- This form of representing $T(n)$ tells us useful things like
 - > Which algorithm is faster
 - E.g. If algorithm A with running time **$T(n)=n+3$** and algorithm B with running time **$T(n)=2^n$** solve the same problem, then algorithm A is faster.
 - > How running time changes when input changes
 - E.g. If **$T(n) = 2n$** then if input doubles, then number of operations will double, because:
 - **$T(2n) = 4n = 2(2n) = 2T(n)$**

Asymptotic analysis of algorithms

- Expressing running time of algorithm in the form of $T(n)$ usually captures the **worst case of the running time** of the algorithm
 - > **“Worst case”**: an input for which the algorithm will do its **maximum number of operations**
 - E.g. when sorting ascending order, worst case is input in descending order
 - > **“Best case”**: an input for which the algorithm will do its **minimum number of operations** to solve the problem
 - E.g. giving an already sorted list of number as input to an sorting algorithm
 - > **“Average case”**: a “typical”, “average” input (which is encountered in the real world most of the time)

Asymptotic analysis of algorithms

Simple algorithm to find the biggest from a list of 10 numbers. 10 numbers provided by user.

Input: 10 random numbers.

output: the greatest number in input set.

/* generalization: assume that we ask for n numbers*/

```
T=[]
for i in range (0, 10):
    number = int (input("Give me the number: ") )
    T.append(number)
```

#Assume first number as greatest

```
currMax = T[0]
i=1
while (i < len(T)):
    #current number at T[i] greater than current maximum?
    if ( T[i] >= currMax ):
        #Yup! Found a bigger number
        currMax = T[i]
    i = i + 1

print("The result is: ", currMax)
```

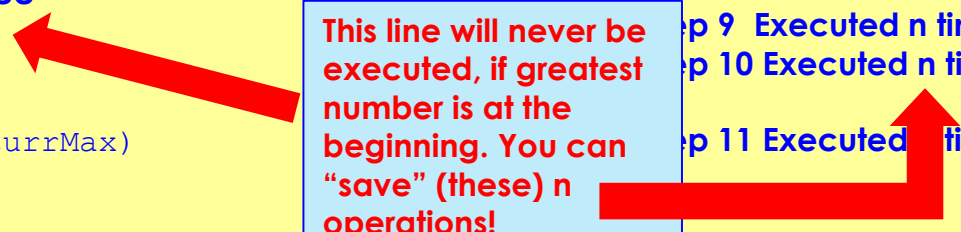
Step 1 Executed 1 time
Step 2 Executed n+1 times
Step 3 Executed n times
Step 4 Executed n times

Step 5 Executed 1 time
Step 6 Executed 1 time
Step 7 Executed n+1 times

Step 8 Executed n times

Step 9 Executed n times
Step 10 Executed n times
Step 11 Executed 1 time

This line will never be executed, if greatest number is at the beginning. You can "save" (these) n operations!



Asymptotic analysis of algorithms

- Running time of algorithm also referred to as **time complexity**
 - > There is also space complexity (guess what this measures...)
- What is asymptotic analysis of algorithms?
 - > It's when we are interested **what happens when $n \rightarrow \infty$ i.e. problem size "goes to infinity"**
 - > Does it matter?
 - Yes, very much!

Asymptotic analysis of algorithms

Why does it matter?

- > Assume algorithm A has time complexity $T(n) = 2^n$ and algorithm B has time complexity $T(n) = n^{20} + 4$. Assume that both solve the same problem.
 - Which algorithm is faster?
 - **Practically**, i.e. for some $n < n_0$, faster is A (i.e. for $n=1,2,3,4,..$)
 - **Theoretically**, when $n > 143$ and $n \rightarrow \infty$, things change: B will always be faster since then $2^n > n^{20} + 4$. Hence B is asymptotically faster.
 - Solve $2^n > n^{20} + 4 \Rightarrow n > 143$
 - Useless note to show how big this is: Our sun has $\sim 2^{80}$ hydrogen atoms. (so 2^{143} is really, really, really huuuuge).

Asymptotic analysis of algorithms

⦿ Explanation?

- > 2^n grows faster than $n^{20}+4$.
- > In general the biggest term in running time determines the magnitude
 - i.e. if $T(n) = 3n^2 + 45n + 2$, then factor $3n^2$ would be the most important **when $n \rightarrow \infty$** , since $3n^2 \gg 45n$ and $3n^2 \gg 2$
 - $3n^2$ grows faster than $45n$ and (of course constant) 2

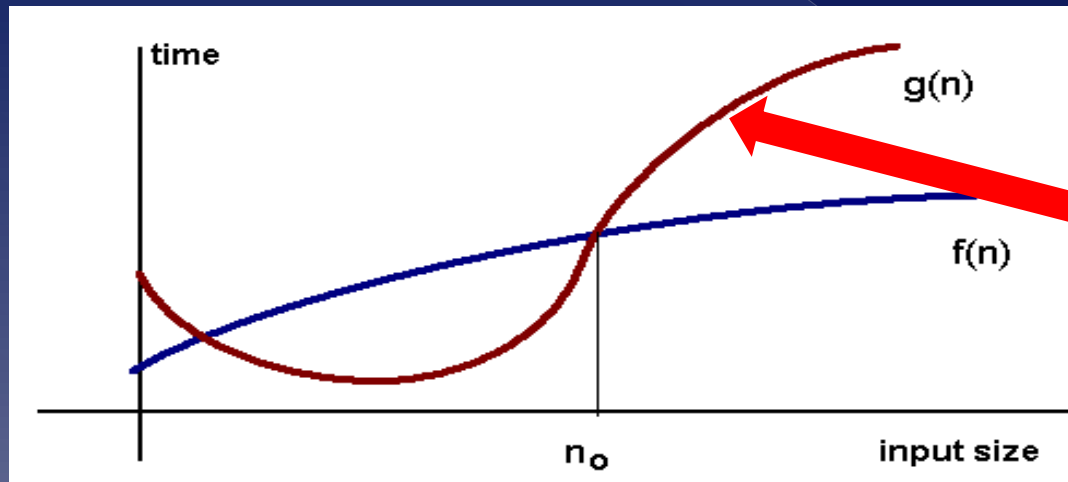
Asymptotic analysis of algorithms

- ◎ Take this as a starting point **to define formalisms** that characterize the asymptotic running time of algorithms (or the theoretical running time:
 - > $O()$, $\Theta()$, $\Omega()$, $\omega()$
- ◎ Definition of these formalisms?

Asymptotic analysis of algorithms

Definition $O()$ (Big-O notation)

- > Let there be an algorithm with running time $T(n)$. We say that $T(n) = O(g(n))$ if there exist a constant c and a n_0 such that $T(n) \leq cg(n)$ for every $n > n_0$
 - This means that saying $f(n) = O(g(n))$ tells us that $g(n)$ is an upper bound of the number of steps an algorithm executes to solve a problem of size n .



Upper bound:
number of operations
will never exceed this
after some problem
size n_0

Asymptotic analysis of algorithms

Examples of Big-O notation

- > **$T(n) = 2n+3 \Rightarrow T(n) = O(n)$. We say that algorithm with running time $T(n)$ has time complexity $O(n)$.**
 - Why? We can find constant **c** and a **n_0 such that $2n+3 \leq cn$** , for each **$n > n_0$** (e.g. for each $c \geq 5$ and $n_0 = 1$)
 - Since time complexity is $O(n)$ we say that running time is linear $O(n)$
- > $T(n) = \lg(n) + 5 \Rightarrow T(n) = O(\lg(n))$
 - Logarithmic time (note: \lg is base-2 logarithm)
- > **$T(n) = n^2+n+7 \Rightarrow T(n) = O(n^2)$**
 - Polynomimic time algorithm
- > **$T(n) = 2^n + 3 \Rightarrow T(n) = O(2^n)$**
 - Exponential time algorithm

Asymptotic analysis of algorithms

Examples of Big-O

> There is also running time of **$O(1)$**

- **Means constant time** i.e. running time independent of input size

- Whether you give input of size 1 or 10^{12} , algorithm will perform constant number of steps ($=c$).

- E.g. assigning a number (value) to variable like $x=123$

- **$O()$ notation is not a tight bound!**

- E.g. $T(n) = n+3 \Rightarrow T(n) = O(n)$ but also $T(n) = O(5^n)$

- **But in general, it is used as a “tight bound”**

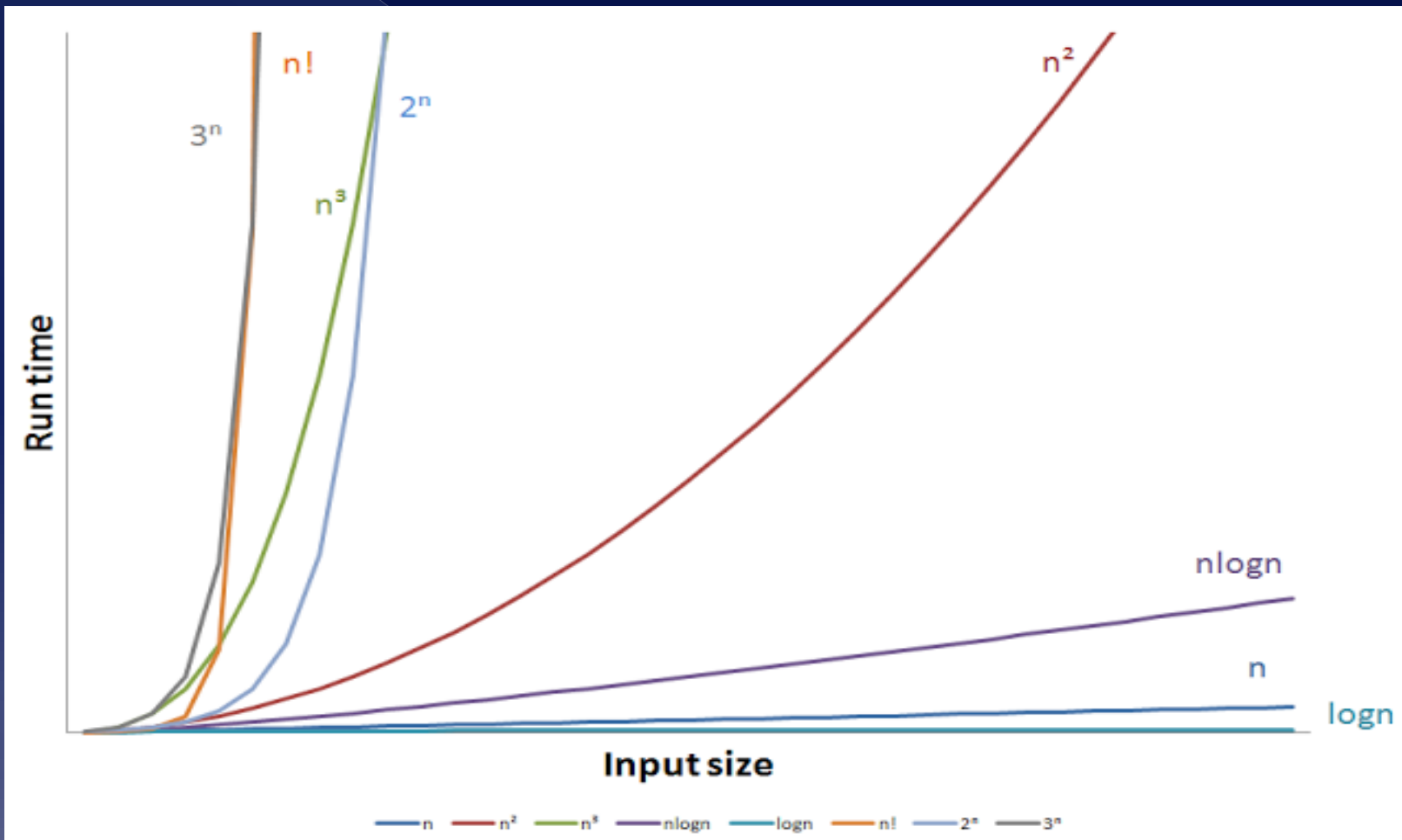
- So when you see that an algorithm **has time complexity of $O(n^2)$** you **can assume that it will never execute more than cn^2 operations** if given an input size of n .

Asymptotic analysis of algorithms

- ◉ What is the constant, **c**, in the definition of Big-O representing?
 - > E.g. in the definition on $T(n) = O(g(n))$ we have $cg(n) \leq T(n)$.
 - > Intuitively: **represents the environment the algorithm runs on**
 - E.g. speed of computer, memory of computer, what other apps were running etc
 - Such **aspects are modelled**, but are **considered a constant, c**. Different computers will have different values of c.

Asymptotic analysis of algorithms

- Examples of **asymptotic time complexity and growth of functions.**



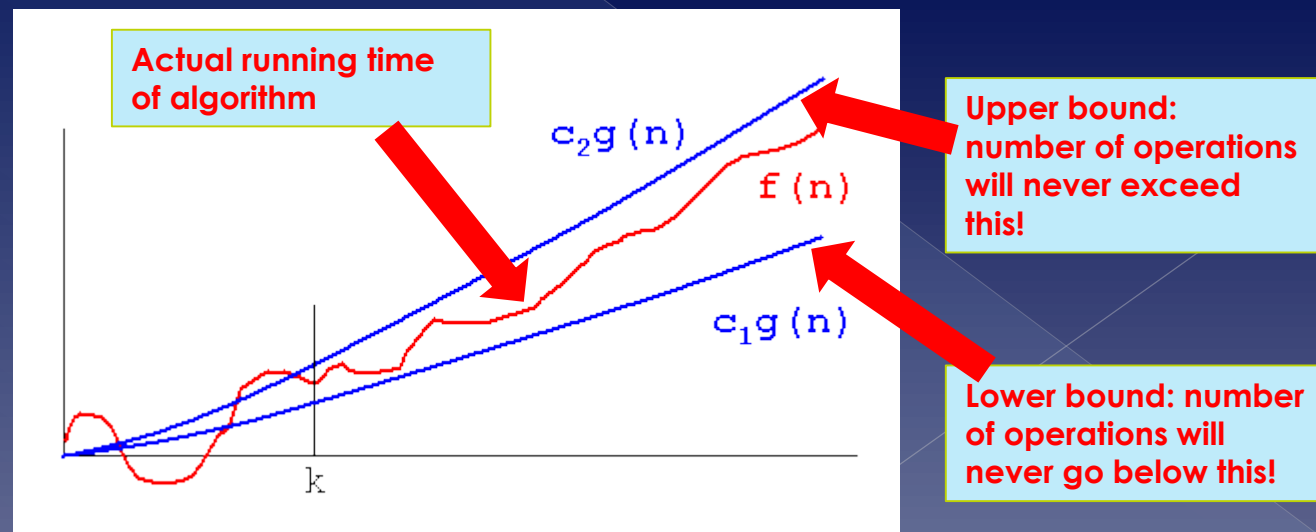
Asymptotic analysis of algorithms

- ◉ In general, Big-O helps in **grouping algorithms**
 - > E.g. Class of algorithms with **$O(n)$** are said to be **linear**
 - > E.g. Class of algorithms with **$O(\log n)$** are said to be **logarithmic**
- ◉ From now on, you'll see **$O()$ as a measure** of the running time of algorithms
 - > For example when someone tells you that an algorithm has time complexity of **$O(n^3)$** then you can understand that running time is proportional to **n^3** of its input n
 - This means, running time can be **$T(n) = 2n^3$** or **$T(n) = 13n^3 + 3n^2 + 3$** but also **$T(n) = 97129n^3 + 34n + 1$** . But never faster growing than **n^3** !

Asymptotic analysis of algorithms

Definition $\Theta()$ (Big-Theta notation)

- > Let there be an algorithm with running time $T(n)$. We say that $T(n) = \Theta(g(n))$ if there exist constants c_1, c_2 and a n_0 such that $c_1g(n) \leq T(n) \leq c_2g(n)$ for every $n > n_0$
 - This means that $g(n)$ is upper and lower bounding $T(n)$ (lower bound=gives also minimum number of steps to solve the problem).



Asymptotic analysis of algorithms

- ◉ In general, algorithms with **polynomial time complexities** are considered **fast and preferred**
 - > Even with great exponent e.g. $O(n^{50})$
- ◉ Algorithms with **exponential time complexity** are slow.
 - > E.g. **$O(3^n)$ is slow**
 - > If an algorithm has time complexity of **$O(2^n)$** , then **if problem size is increased by one ($n+1$)** number of **operations double** (yup, that's sh*tty!).

Asymptotic analysis of algorithms

◉ Example algorithms

> QuickSort:

- Best case: $O(n \lg n)$
- Average case: $O(n \lg n)$
- Worst case: $O(n^2)$

> MergeSort

- Best case: $O(n \lg n)$
- Average case: $O(n \lg n)$
- Worst case: $O(n \lg n)$

> All cryptographic algorithms

- Exponential time complexity to crack password

> Algorithm for making the course schedule for school/university

- Exponential time (that's why humans do it)

Asymptotic analysis of algorithms

Example of time complexities of algorithms in data mining

> **K-means**

- If k and d (dimensions) fixed: $O(n^{dk+1} \log n)$, n number of objects/data to be clustered
- **NP-hard** for general form of the problem
 - What is NP-hard? As hard/difficult as the hardest problems to solve

> **Apriori**

- $O(mn + (1-R^M)/(1-R))$, n =number of input transactions, m =threshold, R =number of unique elements.

About data

About data

- What is data? **A collection of objects and their attributes**
- An **attribute** is a **property or characteristic of an object**
 - Examples: eye color of a person, temperature, etc.
 - Attribute is also known as variable, field, characteristic, feature or dimension
- A **collection of attributes describe an object**
 - Object is also known as *record, point, case, sample, entity, or instance*

**Objects,
records,
instances,
observations,
tuples**

Attributes/Dimensions

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

About data

⦿ **Attribute values?**

- > Numbers or symbols assigned to an attribute

⦿ **Attribute vs Attribute values**

- > Same attribute can be mapped to different attribute values
 - E.g. height in feet or meters
- > Different attributes can be mapped to the same set of values
 - E.g. Attribute values for ID and age are integers
 - But with different properties: id's don't have limits, age has

About data

- ◉ There are different types of attributes (based on **how you can reason with these**):
 - > **Nominal**
 - Examples: ID numbers, eye color, zip codes
 - > **Ordinal**
 - Examples: rankings (e.g., taste of potato chips on a scale from 1-10 – Likert scale), grades, height in {tall, medium, short}
 - > **Interval**
 - Examples: calendar dates, temperatures in Celsius or Fahrenheit.
 - > **Ratio**
 - Examples: temperature in Kelvin, length, time, counts

About data

- The **type of an attribute** depends on which of the following properties it possesses (basically **what arithmetic operations** you can do with them):
 - > **Distinctness:** = ≠
 - > **Order:** < >
 - > **Addition:** + -
 - > **Multiplication:** * /

 - > **Nominal attribute:** distinctness
 - > **Ordinal attribute:** distinctness & order
 - > **Interval attribute:** distinctness & order & addition
 - > **Ratio attribute:** distinctness & order & addition & Multiplication

	Attribute Type	Description	Examples	Allowed Operations
Qualitative	Nominal	The values of a nominal attribute are just different names, i.e., nominal attributes provide only enough information to distinguish one object from another. (=, ≠)	zip codes, employee ID numbers, eye color, sex: { <i>male</i> , <i>female</i> }	mode, entropy, contingency correlation, χ^2 test
	Ordinal	The values of an ordinal attribute provide enough information to order objects. (<, >)	hardness of minerals, {good, better, best}, grades, street numbers, Likert scales	median, percentiles, rank correlation, run tests, sign tests
Quantitative	Interval	For interval attributes, the differences between values are meaningful, i.e., a unit of measurement exists. (+, -) but not (*, /) . E.g. 30°C is not twice as hot as 15°C	calendar dates, temperature in Celsius or Fahrenheit	mean, standard deviation, Pearson's correlation, t and F tests
	Ratio	For ratio variables, both differences and ratios are meaningful. (*, /)	temperature in Kelvin, monetary quantities, counts, age, mass, length, electrical current	geometric mean, harmonic mean, percent variation

Allowed transformation i.e. transformation that do not change the meaning of the attribute

Attribute Level	Allowed Transformation	Comments
Nominal	Any permutation of values	If all employee ID numbers were reassigned, would it make any difference?
Ordinal	An order preserving change of values, i.e., $new_value = f(old_value)$ where f is a monotonic function.	An attribute encompassing the notion of good, better best can be represented equally well by the values {1, 2, 3} or by {0.5, 1, 10}.
Interval	$new_value = a * old_value + b$, where a and b are constants	E.g. the Fahrenheit and Celsius temperature scales differ in terms of where their zero value is and the size of a unit (degree).
Ratio	$new_value = a * old_value$	Length can be measured in meters or feet.

Discrete and continuous

⦿ **Discrete Attribute**

- > Has only a finite (or countable infinite set) of values (countable means can be ordered with a relationship)
- > Examples: zip codes, counts, or the set of words in a collection of documents
- > Often represented as integer variables.
- > Note: binary attributes are a special case of discrete attributes.

⦿ **Continuous Attribute**

- > Has real numbers as attribute values (cannot be ordered with a relationship)
- > Examples: temperature, height, or weight.
- > Practically, real values can only be measured and represented using a finite number of digits.
- > Continuous attributes are typically represented as floating-point variables.

Types of data sets

- Ways in which they are represented/structured
 - > “Structured” data : ordered/grouped in some particular way
- **Record data**
 - > Data Matrix
 - > Document Data
 - > Transaction Data
- **Graph data**
 - > World Wide Web
 - > Molecular Structures
- **Ordered data**
 - > Spatial Data
 - > Temporal Data
 - > Sequential Data
 - > Genetic Sequence Data

Important characteristics of structured data

◉ Dimensionality

- > How many dimensions the data has (here **dimensions**: number of features, attributes)
- > Dimensionality is a big problem (curse of dimensionality)

◉ Sparsity

- > How many values are present (or non-present/zero)?

◉ Resolution

- > Different patterns at different scales

Record data

- Record: a **fixed set of attributed**, handled as one entity
- Record data: collection of records

Record data

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

One record

Data matrix

- If data objects have the same fixed set of numeric attributes, then the data objects **can be thought of as points in a multi-dimensional space**, where each dimension represents a distinct attribute
- Such data set can be represented by an m by n matrix, where there are m rows, one for each object, and n columns, one for each attribute

Projection of x Load	Projection of y load	Distance	Load	Thickness
10.23	5.27	15.22	2.7	1.2
12.65	6.25	16.22	2.2	1.1

Document data

- Each document becomes a **'term' vector**,
 - each term is a component (attribute) of the vector,
 - the value of each component is the number of times the corresponding term occurs in the document.

Note: Search engines like google, Bing do this

Term/lemma/word **'team'** appears in **'Document 1'** 3 times

	team	coach	play	ball	score	game	win	lost	timeout	season
Document 1	3	0	5	0	2	6	0	2	0	2
Document 2	0	7	0	2	1	0	0	3	0	0
Document 3	0	1	0	0	1	2	2	0	3	0

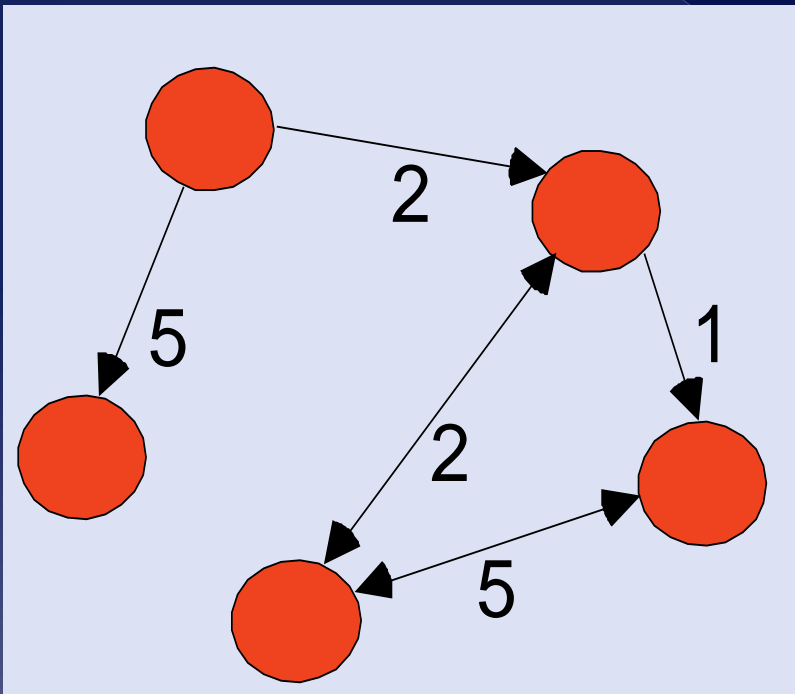
Transaction data

- A special type of record data, where
 - > each record (transaction) involves a set of items.
 - > For example, consider a **grocery store**. The set of products purchased by a customer during one shopping trip constitute a transaction, while the individual products that were purchased are the items.

<i>TID</i>	<i>Items</i>
1	Bread, Coke, Milk
2	Beer, Bread
3	Beer, Coke, Diaper, Milk
4	Beer, Bread, Diaper, Milk
5	Coke, Diaper, Milk

Graph data

- Data represented with nodes and links
(=**graphs**)



- **Examples**

- > World wide web (pages, links)
- > References in scientific articles
 - Who references which paper (link)
- > Calculate
 - **PageRank (google)**
 - **h-index**
 - **Hubs**

Ordered data

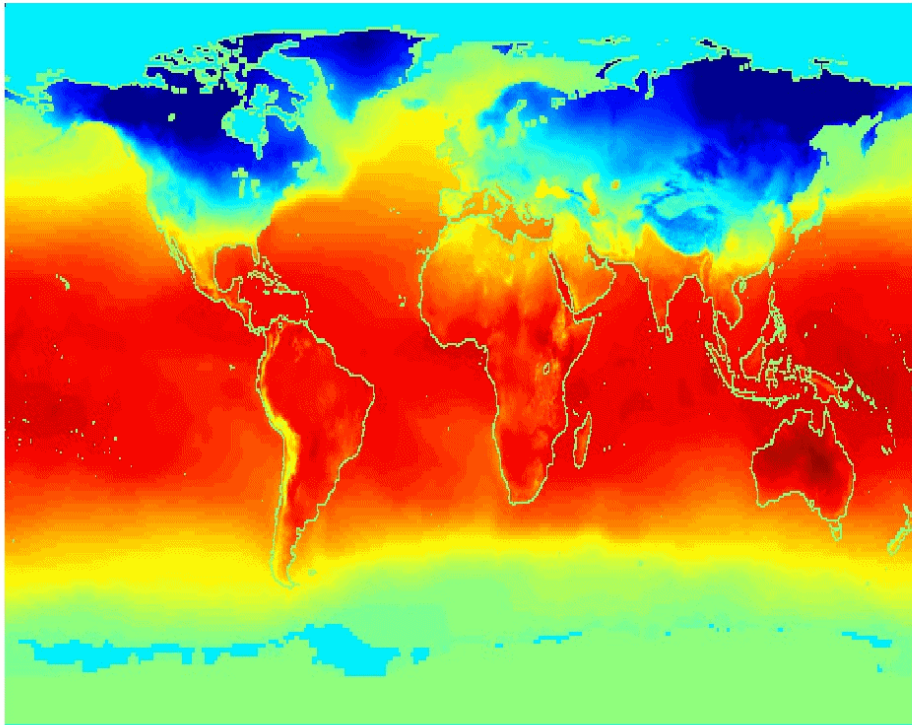
- ◎ Position/rank matters
 - > E.g. genomic sequence

```
GGTTC CGCCTTCAGCCCCGCGCC  
CGCAGGGCCCGCCCCGCGCCGTC  
GAGAAGGGCCCGCCTGGCGGGCG  
GGGGGAGGCGGGGCCGCCCGAGC  
CCAACCGAGTCCGACCAGGTGCC  
CCCTCTGCTCGGCCTAGACCTGA  
GCTCATTAGGCGGCAGCGGACAG  
GCCAAGTAGAACACGCGAAGCGC  
TGGGCTGCCTGCTGCGACCAGGG
```

Ordered data

- Spatio-temporal data

Jan



Average
Monthly
Temperature of
land and ocean

Data quality

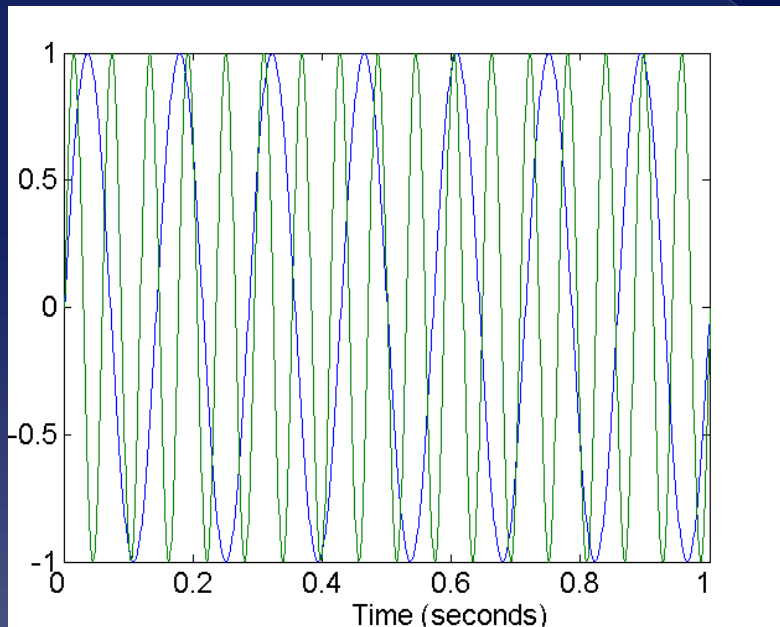
Data quality

◎ Data quality?

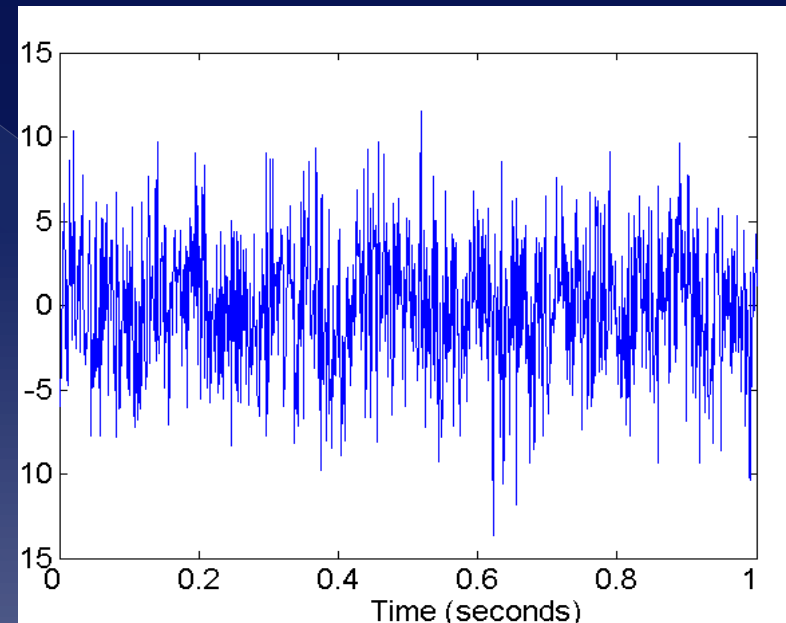
- > The aspects of data sets that make it **suitable/useful** (or not) **for processing to achieve a goal**
- > Common data quality issue/problems
 - Noise and outliers
 - Missing values
 - Duplicate data

Noise

- Noise refers to **involuntarily modification** of **original values**
 - Examples: distortion of a person's voice when talking on a poor phone and "snow" on television screen



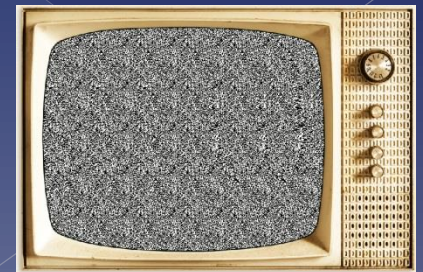
Two Sine Waves (voice)



Two Sine Waves + Noise (voice + distortion). Yup it's a miracle that you can hear a voice.

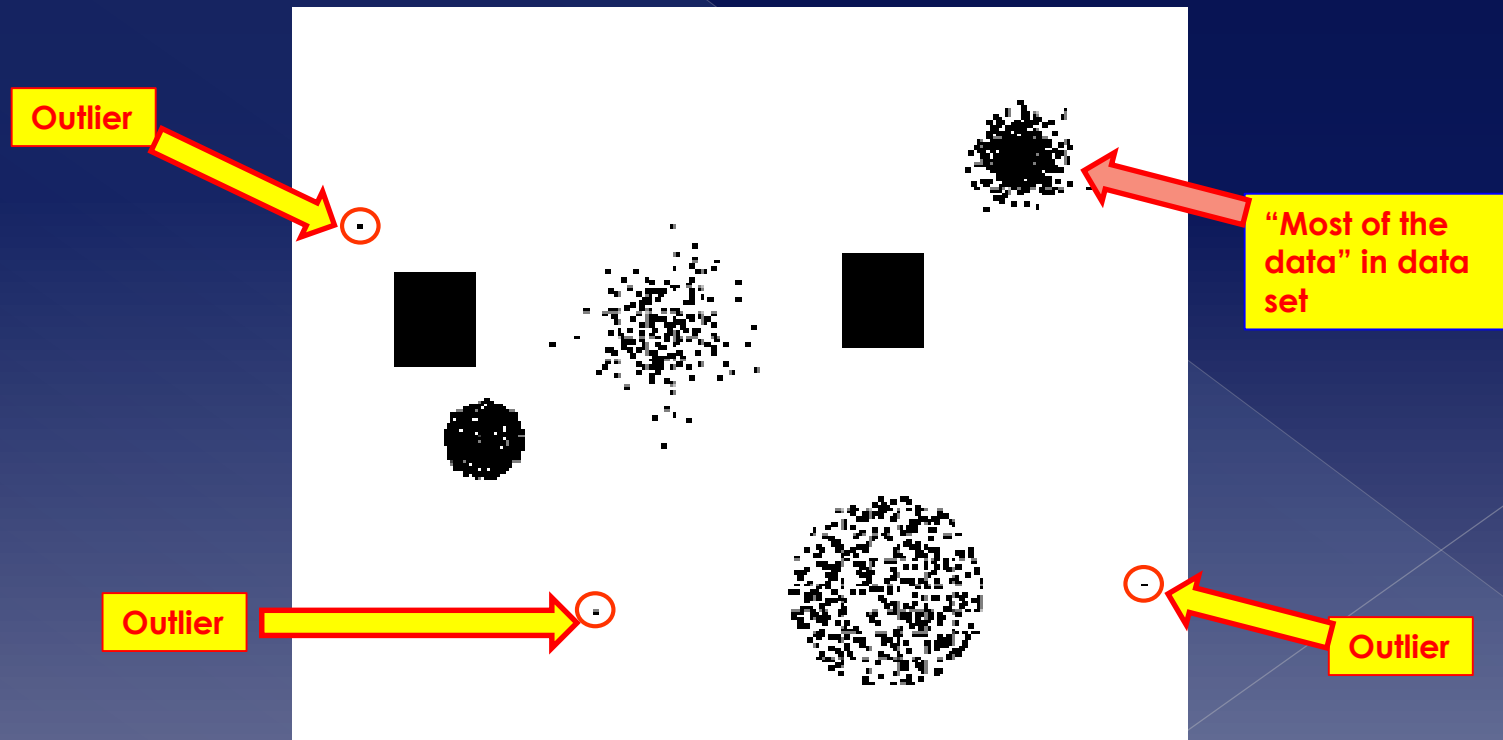
Noise

- ◉ Useless/funny bits of noise
 - > Give rise to **information theory**
 - Claude Shannon aiming at separate voice (information) from not-voice (noise)
 - > Noise **can be interesting**
 - “Snow” on old TVs is **~40% cosmic radiation (from Big Bang)**
 - Meaning the **“snow” that you saw**, was **the cosmos/universe on your TV**
 - Not anymore though due to digital TV.



Outliers

- Outliers are data objects **with characteristics** that **are considerably different** than most of the other data objects in the data set
 - > Practical rule: Normally distributed data, everything that's more than 3 standard deviation away from the mean is suspect of an outlier.
 - > Special graphs to visualize them e.g. Box plot (Θηκόγραμμα)



Outliers

- Can seriously distort your view when analyzing data even in the simplest way
 - > E.g. finding mean/average

Person	Income (in dollars)
Bill	100000000000
Jim	19
John	20
Phil	25
Martha	16

(Naïve) average income:
2000000016 dollars

*#LOL, one person made everyone rich.
No joke pal.*

Missing values

- Reasons for missing values
 - > Information is **not collected** (e.g., people decline to give their age and weight)
 - > Attributes may **not be applicable** to all cases (e.g., annual income is not applicable to children)
 - > Devices may be **faulty** (e.g. kaputt thermometer)
- Handling missing values
 - > Eliminate missing values in some way (commonly used – many techniques)
 - > Ignore the Missing Value During Analysis
 - > Replace with all possible values (weighted by their probabilities)

Duplicate data

- Data set may include data objects that are duplicates, or **almost duplicates** of one another
 - > Major issue when **merging data from heterogeneous sources (typical in greek public sector)**
- Examples:
 - > Same person with **multiple/different email addresses**
 - Yes, I know what you did on facebook, twitter, insta etc.
- The term is “Data cleaning”
 - > Process of **dealing with duplicate data issues**
 - E.g. names: in greek Κων/νος, Κωνσταντινος, Κώστας, Αγ. Βαρβάρα, Αγία Βαρβάρα etc

Data preprocessing

Data preprocessing

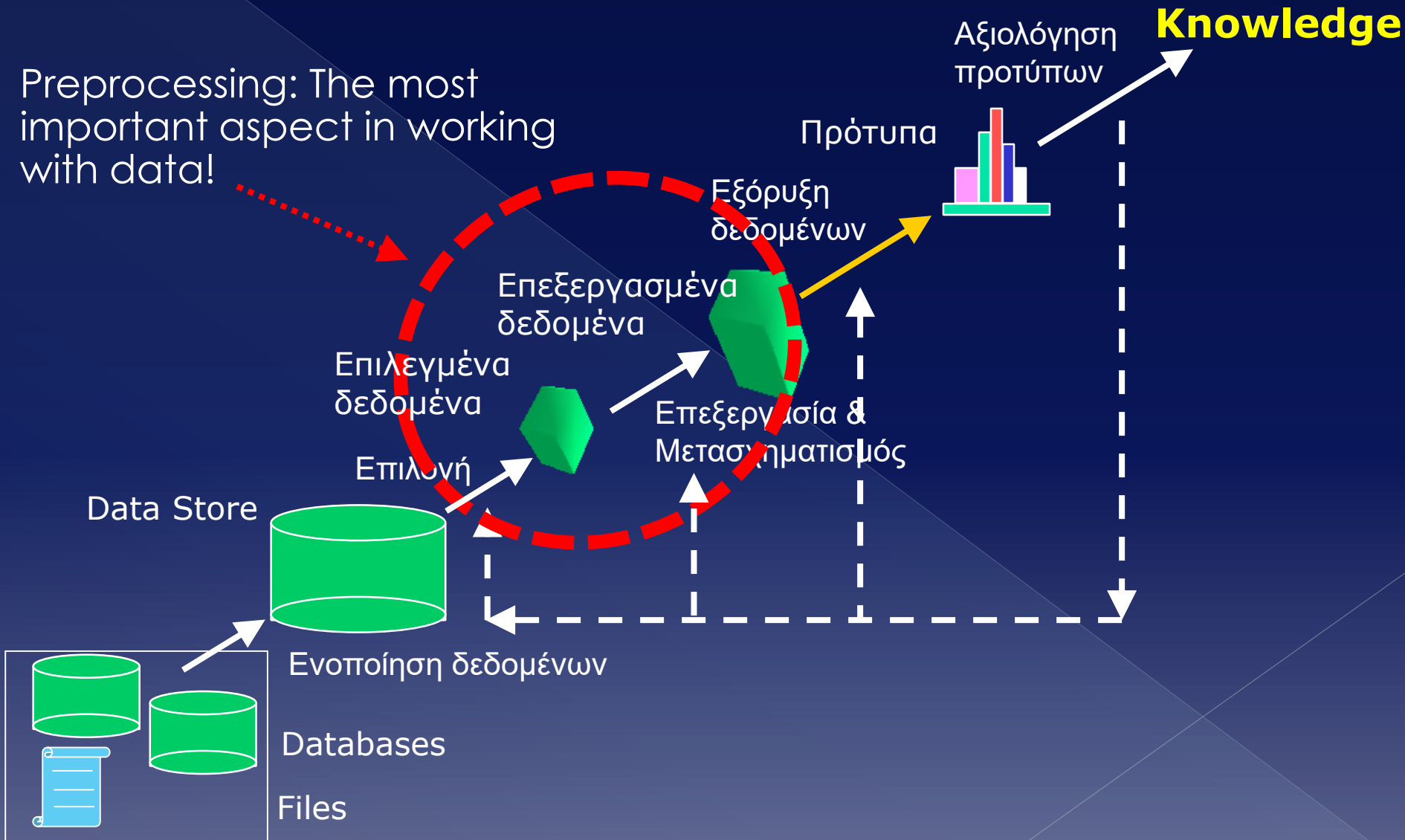
- ◎ Data preprocessing?
 - > Steps that aim **making the data**, before their processing, **suitable for the desired processing**.
 - The issue here is to optimize various aspects that may affect processing, in particular
 - **Required space**
 - **Processing time, minimize running times i.e. $O(g(n))$**
 - Don't forget: we're **working with Big data!**
- ◎ Probably the most important step in data mining
- ◎ In general, **about 70%-80%** of total time is **consumed on data preprocessing tasks**

Data preprocessing

- ⦿ Can definitely shoot your own foot
 - > **Wrong preprocessing yields ALWAYS to wrong results when doing analys. Garbage in-garbage out.**

Preprocessing

Preprocessing: The most important aspect in working with data!



Data preprocessing

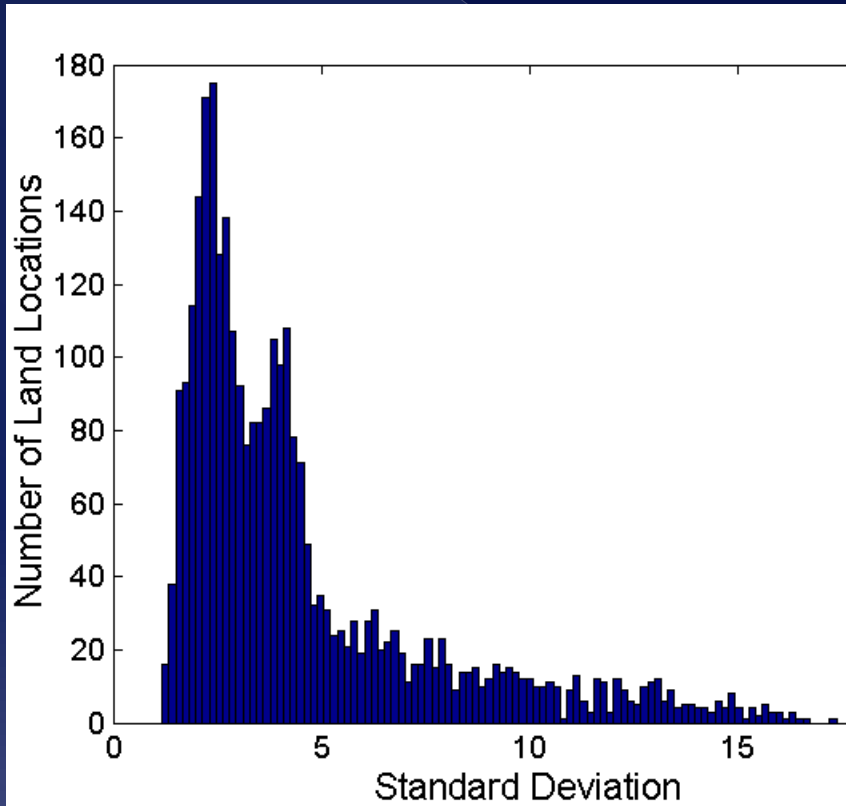
- ⦿ Preprocessing techniques/methods
 - > Aggregation
 - > **Sampling**
 - > **Dimensionality reduction**
 - > Feature subset selection
 - > Feature creation
 - > **Discretization**
 - > Attribute transformation

Aggregation

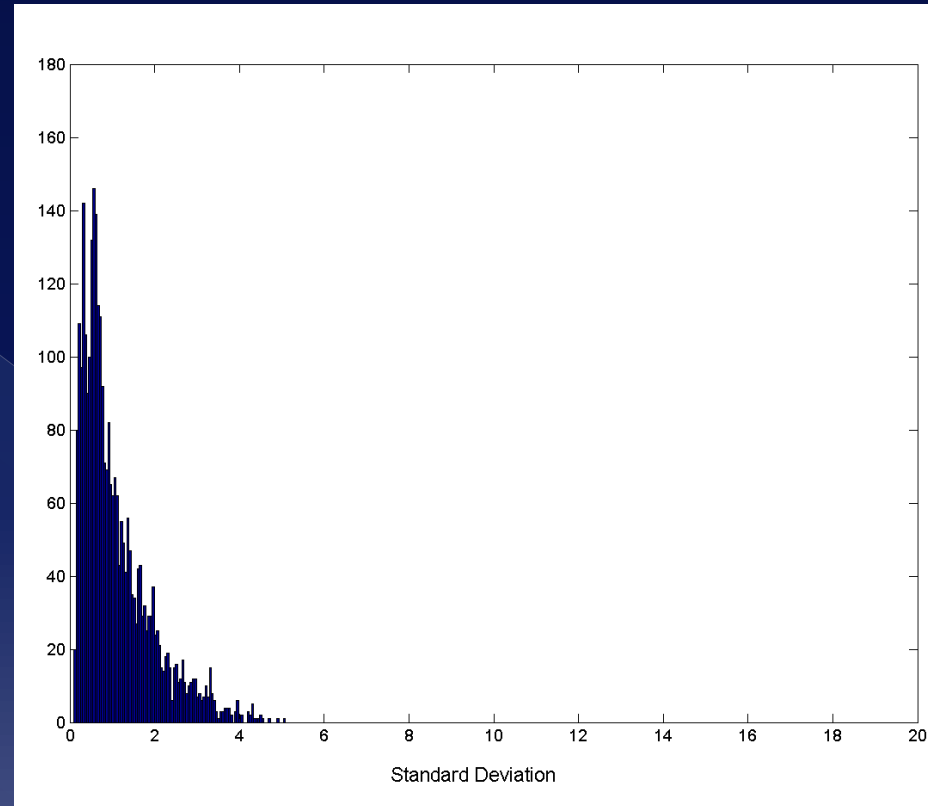
- ◉ Combining two or more attributes (or objects) into a single attribute (or object)
- ◉ Purpose
 - > **Data reduction**
 - Reduce the number of attributes or objects
 - > **Change of scale**
 - Cities aggregated into regions, states, countries, etc
 - > **More “stable” data**
 - Aggregated data tends to have less variability

Aggregation

Variation of Precipitation in Australia



Standard Deviation of
Average Monthly
Precipitation



Standard Deviation of
Average Yearly
Precipitation – **aggregated**
(note smaller variability)

Sampling

- Sampling is the main technique employed for data selection.
 - > It is often used for both the preliminary investigation of the data and the final data analysis.
- Why sampling?
 - > Statisticians sample because obtaining the **entire set** of data of interest is **too expensive or time consuming**.
- Sampling is used in data mining because **processing the entire set** of data of interest is **too expensive or time consuming**.

Sampling

- The key principle for effective sampling is the following:
 - > using a sample will **work almost as well as using the entire data sets, if the sample is representative**
 - > A sample is **representative if it has approximately the same property** (of interest) as the original set of data
 - In terms of its distribution

Sampling

- ◎ **Simple Random Sampling**

- > There is an equal probability of selecting any particular item

- ◎ **Sampling without replacement**

- > As each item is selected, it is removed from the population

- ◎ **Sampling with replacement**

- > Objects are not removed from the population as they are selected for the sample.
 - In sampling with replacement, the same object can be picked up more than once

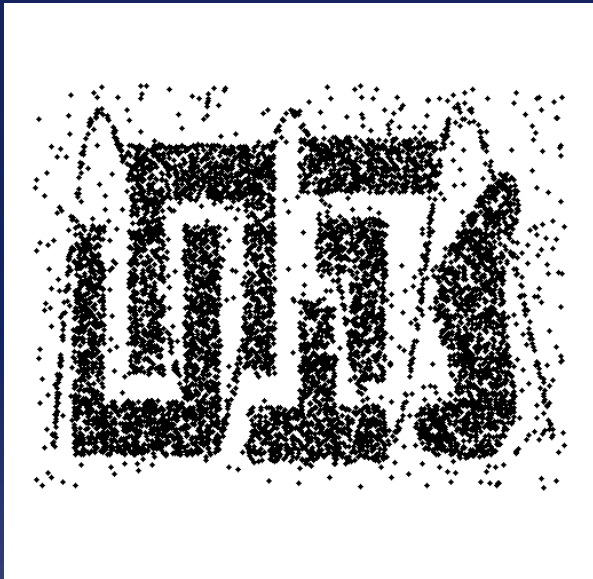
- ◎ **Stratified sampling**

- > Split the data into several partitions; then draw random samples from each partition

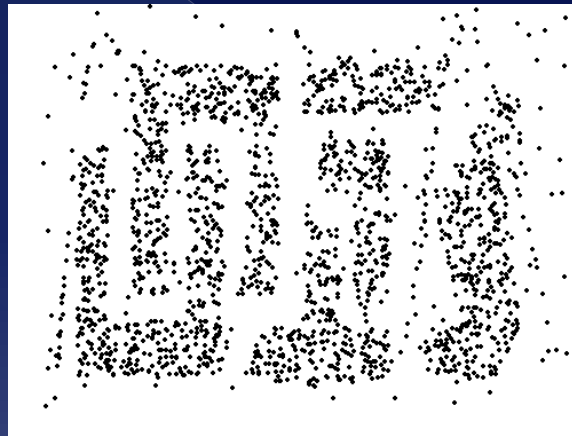
Sampling

- Effect of sample size?

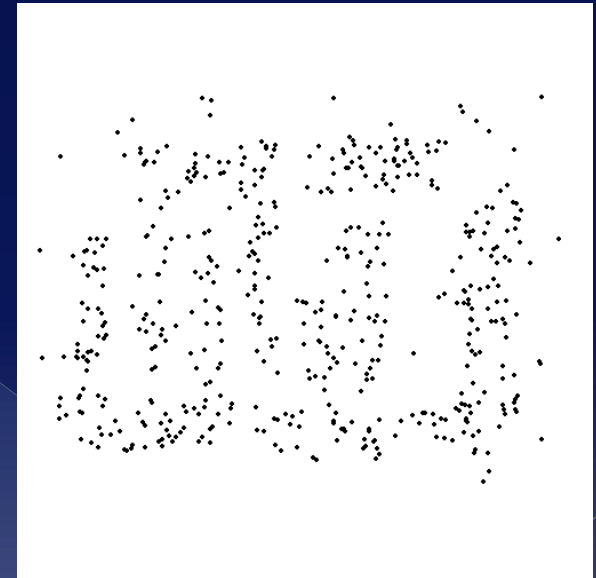
Effects almost any type of data: e.g. when you talk on the phone and your voice is sent to other guy/girl



8000 points



2000 Points

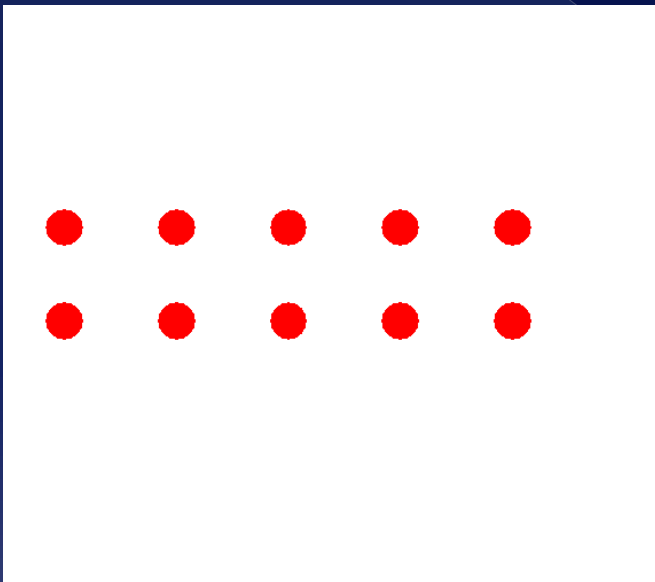


500 Points

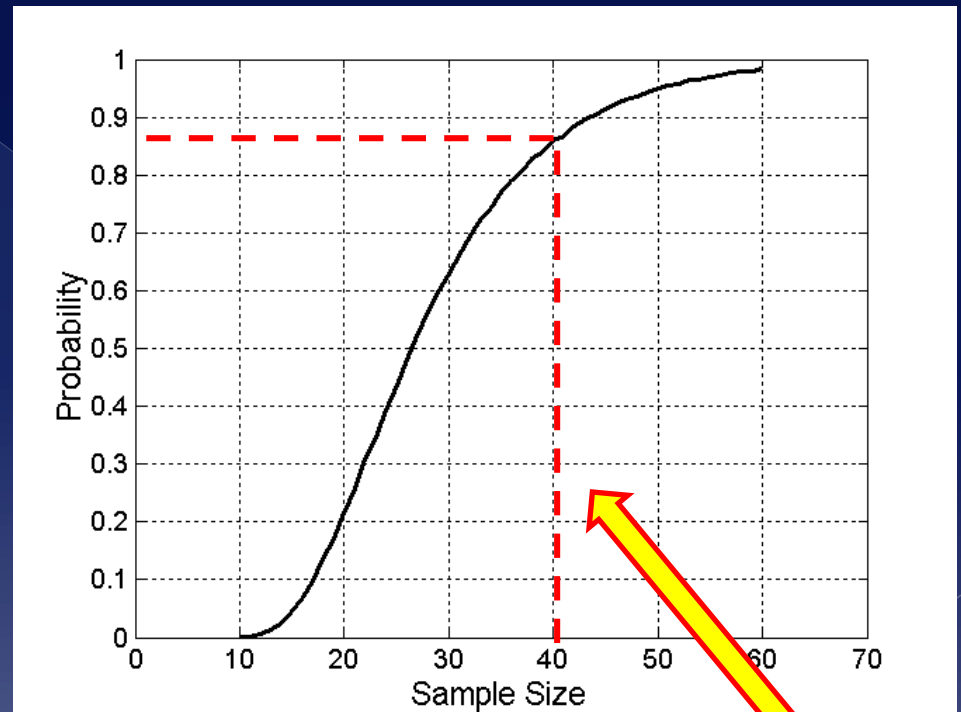
More objects/points/data is in general better. But more objects require more space, more time (preprocessing and analysis). Tradeoff.

Sampling

- What sample size is necessary to get at least one object from each of 10 groups.



10 groups



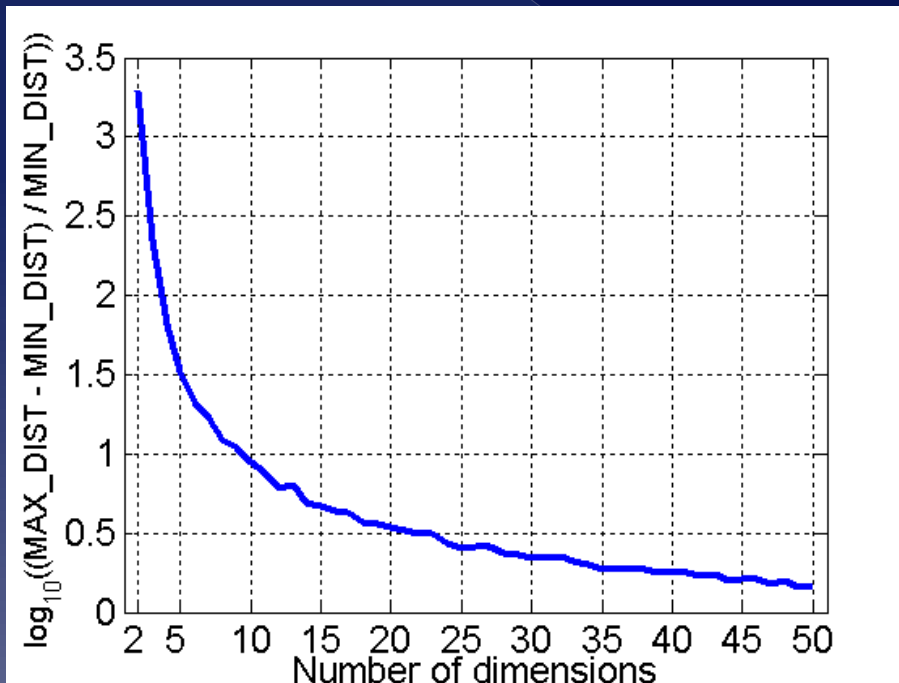
This graph will tell you. For example with a sample size of 40, the probability of having at least one from each group is ~ 0.87 .

Dimensionality reduction

◉ Curse of dimensionality

- > Remember: **dimensions = number of attributes**

- Randomly generate 500 points
- Compute difference between max and min distance between any pair of points
- Note: as dimensions increase, less meaningful distance **which causes problems when clustering!**

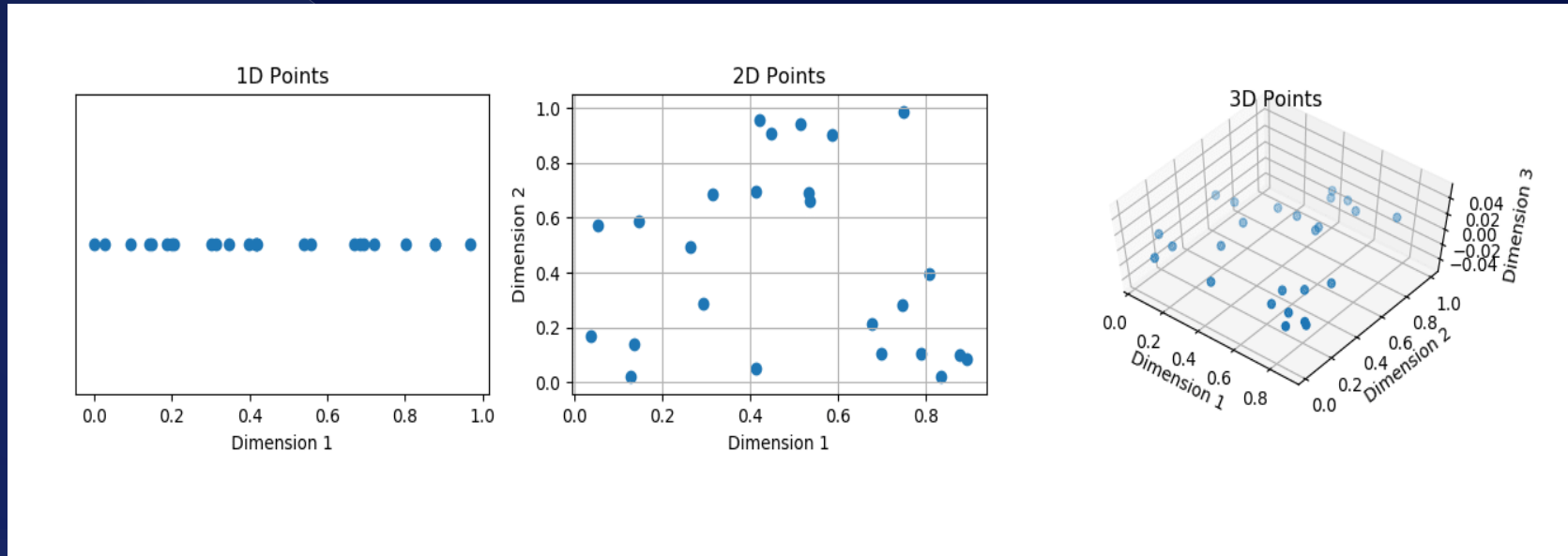


When **dimensionality increases**, data becomes increasingly sparse in the space that it occupies (i.e. many, many, missing or zero values)

Definitions of **density and distance** between points, which is critical for clustering and outlier detection, **become less meaningful**

Dimensionality reduction

Curse of dimensionality –visual example



1 variable

2 variables

3 variables

- **Same data (25 observations)**, when one more variable (dimension) is added. Look at how the space increases exponentially. Data becomes increasingly sparse inside the space that it occupies (i.e. many, many, missing, zero or absent values). **This means that when dimensions increase but not your number of observations, statistical tests lose power. Many (much) more observations are required.**

Dimensionality reduction

- ⦿ **Due to the curse of dimensionality**, you need at least a **minimum number of observations in your dataset**, in order to get reliable results from any statistical test or any machine learning algorithm.
 - > I guess you've heard that.
 - > **Rule of thumb:** in machine learning, at least 5 different values for each variable.

Dimensionality reduction

- ◎ Purpose of dimensionality reduction
 - > Overcome dimensionality curse (ha, take that!)
 - > **Reduce space and time required** by data processing algorithms
 - Data too Big for your machine
 - > **Facilitate easy visualization** of data – i.e. drawing graphs as a first look at your data
 - > May help in reducing noise
- ◎ Techniques/methods
 - > **Principal Components Analysis (PCA)**
 - > Singular Value Decomposition (SVD)
 - > Supervised and non-supervised techniques

Data preprocessing

PCA – Principal Component Analysis

Principal Components Analysis (PCA)

- How to reduce the dimension of a dataset, if it's very large?
 - > E.g. hundreds/thousands of attributes/variables?
 - > You just **can't drop randomly some** of the variables
 - What if you drop/leave out the important ones?
 - Which one are important, which don't?
 - In addition, you want also **to minimize the number of variables** but at the same time keep all the important statistics of your dataset e.g. variance.

Principal Components Analysis (PCA)

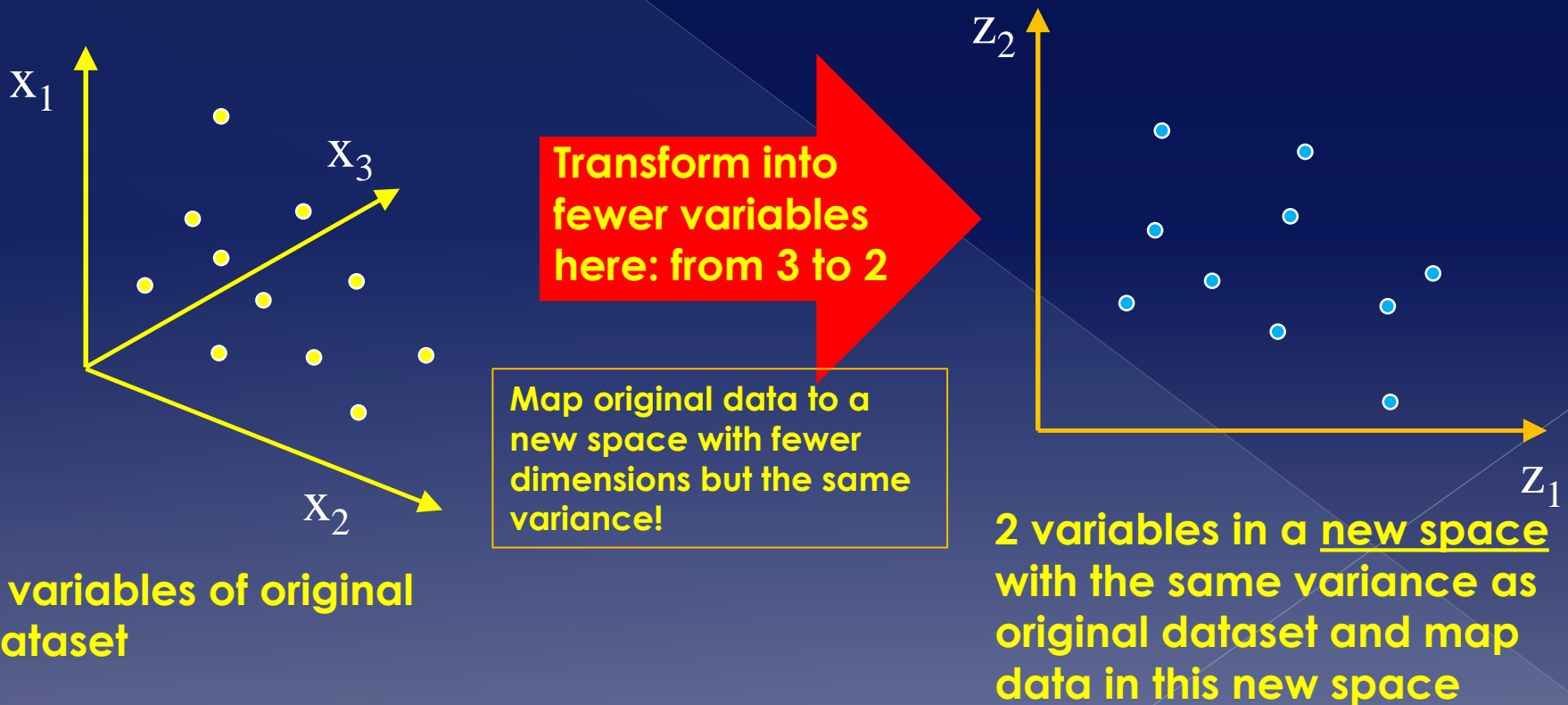
- One approach of doing this is to consider your **original data with the many variables** as points in a space where each variable is a dimension in that space and **transform** this data into data of another space **with fewer dimensions (variables)** but the **same statistical properties**.
 - > And in particular try to **keep** in that new space each variable's variance in the original dataset!

Principal Components Analysis (PCA)

- ◉ But **why keep (or explain) variance** of variables in that new space and **not e.g. mean or any other statistic?**
 - > **Variance is a very, very important aspect** of your data. It's the juicy part.
 - > **Variance tells you how your data varies (goes up and down)** – data with no variance is not interesting – and hence is **more interesting in investigating relationships between variables.**
 - > A lot of existing methods target variance
 - **Linear regression** – explain variance
 - **ANOVA** – compare means by analyzing variance
 - And many, many other...

Principal Components Analysis (PCA)

- So, what we are trying to do is to reduce the variables like the following (note we show here 3 original variables to be reduced to 2 – imagine having 100 variables reducing them to 5 or 10 or 20):



Principal Components Analysis (PCA)

- How to find the new axes (which are in essence variables) of this space which explains most of the variance of the original dataset?
 - > This is what **Principal Component Analysis - PCA** does!

Principal Components Analysis (PCA)

- A method to do this is PCA
 - > What it aims for?
 - It aims to **expressing existing data with high dimensionality (variables/attributes, n)** in the context of a new (optimal) axis system (“subspace”) with fewer dimensions d , i.e. $d < n$.
 - **Goal of PCA:** capture most of the variation in original data set to bring out patterns.
 - **“fewer dimensions”** => reducing dimensionality and hence the curse of dimensionality
 - Basically we **compress the data set.**
 - Note: might lose some of variation of original data, and hence can't perfectly reproduce original data in the new subspace, but this variation is not important (due to being very small/insignificant).
 - This **new “subspace” comprises the Principal Components**
 - **IMPORTANT: PCA works only with numerical vectors!**

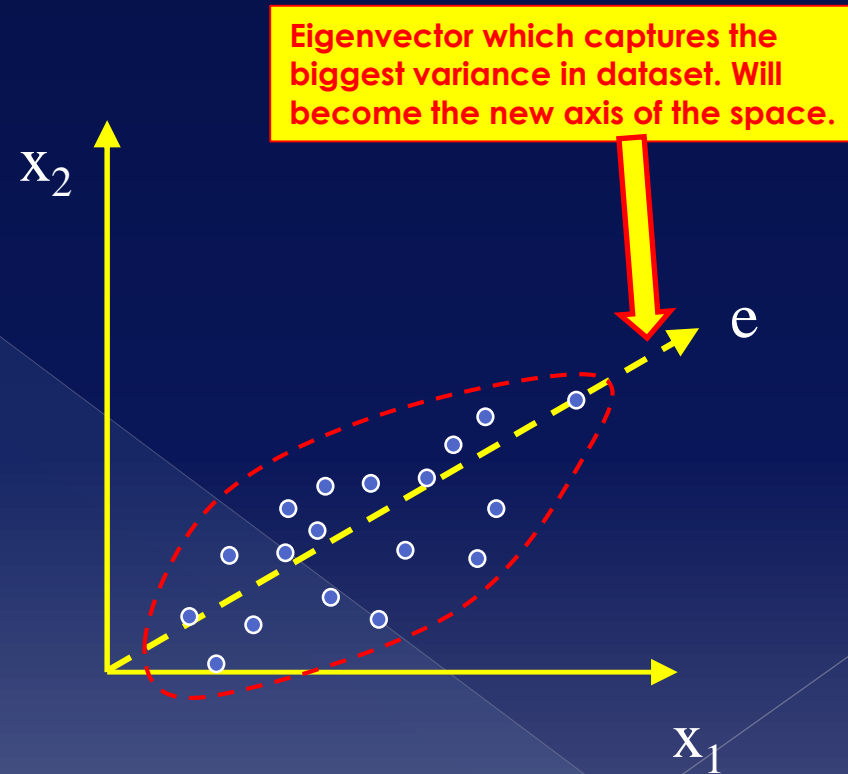
Principal Components Analysis (PCA)

- ◎ A quick look at PCA
 - > Several issues with the new subspace
 - How to choose new dimension d ?
 - How to select **feature space** (“subspace”) that represents our data well (i.e. principal components)?
 - > PCA allows you to create a new space (new variables) with fewer dimensions, which explains variation of the original dataset. You can then map your original data onto this new space and do your analysis there!

Principal Components Analysis (PCA)

How to find the axes (i.e. variables) of new space?

- > The **Eigenvectors, Eigenvalues** of the Covariance matrix define these spaces
 - Eigenvectors are linear independent – i.e. orthogonal to each other
- > The **calculated Eigenvectors** (aka **Principal Components**) will be the **new axes** that define the new space upon which the data will be mapped.



Principal Components Analysis (PCA)

- ◉ What are Eigenvectors/Eigenvalues?
 - > Mathematical definition:
 - Let **A be a nxn matrix**. An **eigenvector v** and an **eigenvalue λ** of matrix A have the following properties:

$$A * v = \lambda * v$$

Principal Components Analysis (PCA)

- PCA is a method for finding the Eigenvectors and Eigenvalues of a dataset and use these vectors (**or a subset of these – the most important ones**) to create a new space with smaller dimensions, upon which the original data will be mapped while maintaining the variance of the dataset.
- You can then do your analysis (any analysis) **in this new space** which has fewer dimensions (variables) and move the data back and forth.

Principal Components Analysis (PCA)

- Steps to calculate Principal Components
 - > Take whole dataset with n dimensions
 - > Normalize the data – make your variable the to have the same SCALE!
 - Not always necessary if scale is not an issue with your data
 - > Compute the dimensional mean-vector (i.e. mean for each dimension/attribute)
 - > Subtract mean from each dimension (make variables have mean =0) – a form of **normalizing the data (THIS IS IMPORTANT!)**
 - > Compute the covariance matrix
 - Indicating how each dimension/attribute varies with respect to all other attributes
 - > Compute Eigenvectors and Eigenvalues of the covariance matrix solving:
 - **$|\lambda I - A| = 0$, λ =eigenvalue, $||$ = determinant, I = unit vector**
 - **$Av = \lambda v$, v = eigenvector**
 - > Choose k largest Eigenvalues and corresponding Eigenvectors
 - > Use these Eigenvectors to form a d x k new matrix W of Eigenvectors – These **explain most of the variance**
 - > Use this d x k Eigenvector matrix to transform each object (vector) onto the new space, as follows:
 - **$\langle \text{New vector} \rangle = W^T x \langle \text{old_vector} \rangle$**


Data preprocessing

PCA Example

Principal Components Analysis (PCA) - Example

- Numerical Example
- Assume the following observations/data about different food items: vitamin C content, protein content)

Variables/features



Vitamin C	Protein
2.5	2.4
0.5	0.7
2.2	2.9
1.9	2.2
3.1	3.0
2.3	2.7
2	1.6
1	1.1
1.5	1.6
1.1	0.9

Records/Observations

Dataset

- Our goal/question to answer?
- **Reduce the number of variables while at the same time keep/explain most of the variance. Here e.g. we want to have only 1 variable**
 - > We have here only 2 variables, so this makes little sense. Imagine e.g. having 250 variables. In such cases you want to reduce the number of variables but “keep” the variance.

Principal Components Analysis (PCA) - Example

- Calculate mean for each variable

Vitamin C	Protein
2.5	2.4
0.5	0.7
2.2	2.9
1.9	2.2
3.1	3.0
2.3	2.7
2	1.6
1	1.1
1.5	1.6
1.1	0.9

mean = 1.81

mean = 1.91

- Scaling our variables not an issue here!
 - > **It would be an issue, if our variables would be measured in different scales. Say one variable is measured in thousands (e.g. population/income) and another in meters (say height of people). In such cases PCA will result in inconsistent results!**
 - > **When scale of any variable is an issue, do min-max normalization or z-score. Will see these methods in detail later.**

Principal Components

Analysis (PCA) - Example

- Subtract mean from each value to normalize the data.

Vitamin C	Protein
$2.5 - 1.81 = \mathbf{0.69}$	$2.4 - 1.91 = \mathbf{0.49}$
$0.5 - 1.81 = \mathbf{-1.31}$	$0.7 - 1.91 = \mathbf{-1.21}$
$2.2 - 1.81 = \mathbf{0.39}$	$2.9 - 1.91 = \mathbf{0.99}$
$1.9 - 1.81 = \mathbf{0.09}$	$2.2 - 1.91 = \mathbf{0.29}$
$3.1 - 1.81 = \mathbf{1.29}$	$3.0 - 1.91 = \mathbf{1.09}$
$2.3 - 1.81 = \mathbf{0.49}$	$2.7 - 1.91 = \mathbf{0.79}$
$2 - 1.81 = \mathbf{0.19}$	$1.6 - 1.91 = \mathbf{-0.31}$
$1 - 1.81 = \mathbf{-0.81}$	$1.1 - 1.91 = \mathbf{-0.81}$
$1.5 - 1.81 = \mathbf{-0.31}$	$1.6 - 1.91 = \mathbf{-0.31}$
$1.1 - 1.81 = \mathbf{-0.71}$	$0.9 - 1.91 = \mathbf{-1.01}$

mean = 1.81

mean = 1.91


NOTE: we will work from now on with the red values!

Hint: mean of both variables is now 0.

Principal Components Analysis (PCA) - Example

- Calculate the covariance matrix

	Vitamin C	Protein
Vitamin C	0.616	0.615
Protein	0.615	0.716

$$\text{cov}(\text{VitaminC}, \text{Protein}) = \frac{\sum_{i=1}^{10} (\text{VitC}_i - \overline{\text{VitC}})(\text{Prot}_i - \overline{\text{Prot}})}{9 = (10 - 1)}$$


NOTE: mean here is calculated on the normalized data (i.,e. mean = 0)

Principal Components Analysis (PCA) - Example

- Calculate **Eigenvalues and Eigenvectors of the Covariance matrix**
- Definition of Eigenvector v with Eigenvalue λ of the covariance matrix $\text{cov}(VitC, Pr)$:
 - > $\text{cov}(VitC, Pr)v = \lambda v \Rightarrow \text{cov}(Vit, Pr)v - \lambda v = 0 \Rightarrow$
 $(\text{cov}(Vit, Pr) - \lambda I_2)v = 0$
- Calculate **Eigenvalues first!**

Principal Components Analysis (PCA) - Example

- Calculate Eigenvalues first

$$(\mathit{cov}(V\mathit{it}, Pr) - \lambda I_2) v = 0$$



Note: 0 is the zero vector. We search for λ (eigenvalue) and corresponding v (eigenvector). Let's remember a little bit of linear algebra: In order for this to have non-zero vector v as solution, the determinant of $(\mathit{cov}(V\mathit{it}, Pr) - \lambda I_2)$ must be zero! Let's do it.

Principal Components Analysis (PCA) - Example

- Calculate Eigenvalues

$$\det(\text{cov}(Vit, Pr) - \lambda I_2) = 0$$

Covariance matrix

0.616	0.615
0.615	0.716

Identity matrix I_2

1	0
0	1

- λ

=

=

$0.616 - \lambda$	0.615
0.615	$0.716 - \lambda$



Determinant of this must be zero.

Principal Components Analysis (PCA) - Example

● Calculate Eigenvalues

$$\text{Det} \left(\begin{array}{|c|c|} \hline 0.616 - \lambda & 0.615 \\ \hline 0.615 & 0.716 - \lambda \\ \hline \end{array} \right) = 0 \Rightarrow$$

$$\Rightarrow (0.616 - \lambda)(0.716 - \lambda) - 0.615 * 0.615 = 0 \Rightarrow \lambda_1 = 0.0489, \lambda_2 = 1.283$$



2 Eigenvalues calculated λ_1, λ_2 !

Principal Components Analysis (PCA) - Example

- Now, **for each Eigenvalue**, calculate the **Eigenvector V**.

For eigenvalue $\lambda = 0.0490$

$0.616 - \lambda$	0.615
0.615	$0.716 - \lambda$

$$* \mathbf{V} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Rightarrow$$

$0.616 - 0.0489$ $= 0.567$	0.615
0.615	$0.716 - 0.0489$ $= 0.667$

$$* \mathbf{V} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Rightarrow$$

Principal Components Analysis (PCA) - Example

- For each Eigenvalue, calculate the Eigenvectors.

For eigenvalue $\lambda = 0.0490$

0.567	0.615
0.615	0.667

$$\begin{matrix} * & \begin{matrix} v1 \\ v2 \end{matrix} & = & \begin{matrix} 0 \\ 0 \end{matrix} & \Rightarrow \end{matrix}$$

$$\begin{aligned} 0.567*v1 + 0.615*v2 &= 0 \\ 0.615*v1 + 0.667*v2 &= 0 \end{aligned}$$

Eigenvalue $\lambda=0.049$

Eigenvector = $\begin{bmatrix} -0.7351 \\ 0.6778 \end{bmatrix}$

Principal Components Analysis (PCA) - Example

- For each Eigenvalue, calculate the Eigenvectors.

For Eigenvalue $\lambda = 1.284$

$0.616 - \lambda$	0.615
0.615	$0.716 - \lambda$

$$* \mathbf{V} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Rightarrow$$

$0.616 - 1.284 =$ -0.668	0.615
0.615	$0.716 - 1.284 =$ -0.568

$$* \mathbf{V} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Rightarrow$$

Principal Components Analysis (PCA) - Example

- For each Eigenvalue, calculate the Eigenvectors.

For Eigenvalue $\lambda = 1.284$

-0.668	0.615
0.615	-0.568

 *

v1
v2

 =

0
0

 =>

$$\begin{aligned} -0.668*v1 + 0.615*v2 &= 0 \\ 0.615*v1 + 0.568*v2 &= 0 \end{aligned}$$

Eigenvalue $\lambda = 1.284$

Eigenvector = $\begin{bmatrix} -0.6778 \\ -0.7351 \end{bmatrix}$

Principal Components Analysis (PCA) - Example

Eigenvalue $\lambda=1.284$

Eigenvector = $\begin{bmatrix} -0.6778 \\ -0.7351 \end{bmatrix}$

Eigenvalue $\lambda=0.049$

Eigenvector = $\begin{bmatrix} -0.7351 \\ 0.6778 \end{bmatrix}$

- These two eigenvectors define a new coordinate system upon which the original data can be projected.
- What variables do these represent?
 - > Define **new variables** as linear combinations of the initial variables i.e.:
 - **New variable 1** = $-0.6778 \cdot \text{VitaminC} - 0.7351 \cdot \text{Protein}$
 - **New variable 2** = $-0.7351 \cdot \text{VitaminC} + 0.6778 \cdot \text{Protein}$

Principal Components Analysis (PCA) - Example

- We found **2 Eigenvalue/Eigenvector** pairs (that's expected. Why expected?)

Eigenvalue $\lambda=1.284$

Eigenvector = $\begin{bmatrix} -0.6778 \\ -0.7351 \end{bmatrix}$

Eigenvalue $\lambda=0.049$

Eigenvector = $\begin{bmatrix} -0.7351 \\ 0.6778 \end{bmatrix}$

- Notice how **one Eigenvalue is greater than the other** ? **$1.284 > 0.049$** .
 - > This means that **the Eigenvector with $\lambda = 1.284$ captures more variance of the dataset than the other Eigenvector!** And in fact, the value of the Eigenvalue is the variance of the data on that (new) dimension!

Principal Components Analysis (PCA) - Example

- **How much variance** does the greatest Eigenvector explain?
 - > Use $\frac{\lambda_k}{\sum_{i=1}^n \lambda_i}$ where n= number of eigenvalues/eigenvectors - to see how much variance Eigenvector with Eigenvalue λ_k explains
 - > In our case Eigenvector with $\lambda=1.284$ explains **1.284 / (1.284+0.049) = 0.96** or **96% of the variance** in the data
 - > In the general case, what you do is **select the k largest Eigenvalues (and corresp. Eigenvectors)** until you are happy with the variance explained – **The selected Eigenvalues/Eigenvectors are the Principal Components!**
 - In this case the explained variance is $\frac{\sum_{i=1}^k \lambda_i}{\sum_{j=1}^n \lambda_j}$
 - Empirical: **aiming at explaining >70% or variance**

Principal Components Analysis (PCA) - Example

- If we are happy with the variance explained, do the following:
 - > Map the original data to the selected k Eigenvectors with the k greatest eigenvalues -in our example, lets say we select only 1 Eigenvalue/Eigenvector pair – the one with the largest Eigenvalue :

Vitamin C	Protein
2.5	2.4
0.5	0.7
2.2	2.9
1.9	2.2
3.1	3.0
2.3	2.7
2	1.6
1	1.1
1.5	1.6
1.1	0.9



Map this data to this feature vector defined by the Eigenvector

Eigenvalue $\lambda=1.284$

Eigenvector = $\begin{bmatrix} -0.6778 \\ -0.7351 \end{bmatrix}$

We aim to do this => Express data solely in terms of the selected Eigenvectors that define a new space!

Note: The two dimensional, original data will be mapped to a one dimensional space explaining large part of the variance!

Principal Components Analysis (PCA) - Example

- What variable(s) do the Eigenvectors (which defined the new space) represent?
 - > It's a **new variable** not in the original dataset! In fact a linear combination of existing variables.

VitaminC

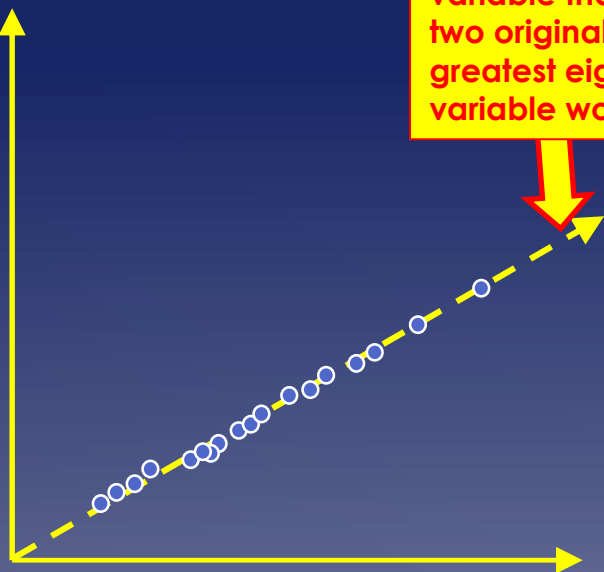
Here we have reduced the two variables (VitaminC, Protein) to only one variable represented by the Eigenvector, and mapped the existing dataset onto that new axis (see the blue points)

Which variable is this?????

None of the existing ones in the dataset! it's a new variable that is created from a linear combination of the two original variables: If this is the eigenvector with the greatest eigenvalue (explaining most of the variance) this variable would be $-0.6778 \cdot \text{VitaminC} - 0.7351 \cdot \text{Protein}$

Eigenvector $\begin{bmatrix} -0.6778 \\ -0.7351 \end{bmatrix}$

Protein



Principal Components Analysis (PCA)

- ◎ Summary
- ◎ Principal Component Analysis - PCA
 - > **What it does**
 - It reduces the number of dimensions in a dataset, while still explaining great amount of variance in the original data
 - > **On what kind of attributes/data does it work?**
 - PCA works **only on Ratio attributes**.
 - Variations of PCA to work in interval data available.
 - > **PCA can make use of the Correlation matrix instead of the Covariance matrix**
 - Important when implementing PCA in R
 - Look at the appropriate parameters!

Principal Components Analysis (PCA)

Principal Component Analysis – PCA

> **When to do it**

- When you want to visualize datasets with many variables (e.g. > 100)
 - Impossible to do with that many variables
 - With PCA, keep e.g. the 3 principal eigenvectors and project data onto that space. 3 variables can be visualized easily.
- Identify correlated variables in datasets with many variables.

Principal Components Analysis (PCA)

- A more concrete example:
- *"In the Places Rated Almanac, Boyer and Savageau rated 329 communities according to the following nine criteria:*
 - > *Climate and Terrain*
 - > *Housing*
 - > *Health Care & the Environment*
 - > *Crime*
 - > *Transportation*
 - > *Education*
 - > *The Arts*
 - > *Recreation*
 - > *Income*
 - > ...

Note that within the dataset, except for housing and crime, the higher the score the better. For housing and crime, the lower the score the better. Where some communities might do better in the arts, other communities might be rated better in other areas such as having a lower crime rate and good educational opportunities."

Objective: Search for relationships (correlation) between these variables.

Principal Components Analysis (PCA)

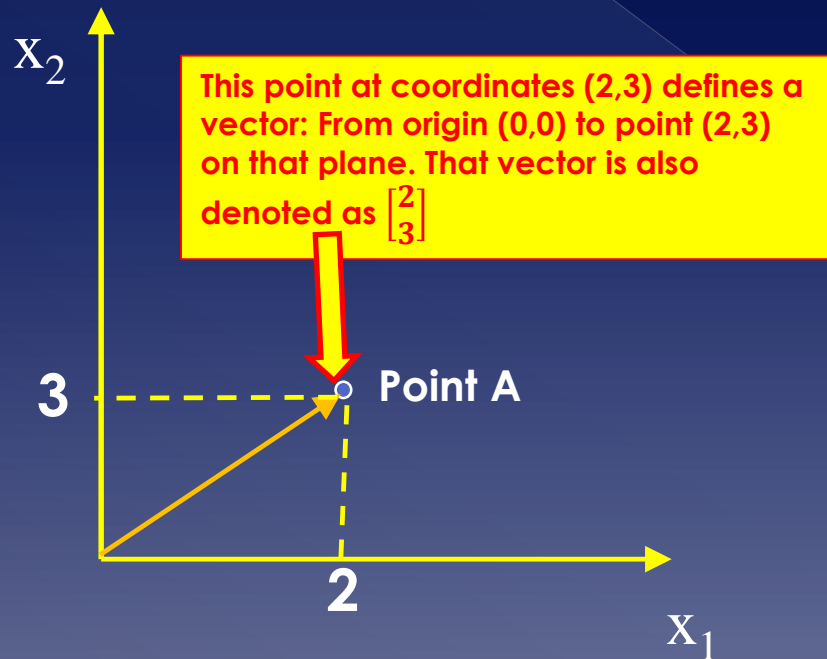
- In order to do this you need to check all combination of variables and expect a linear correlation
 - > In this 9-dimensional space, observation which are correlated will appear closely together
 - > **Difficult to see**: too many scatterplots of variables against each other, how to draw a 9-dimensional space etc.
- Or you could **do a PCA**, find principal components and project data onto these
 - > Such projection **gives you a quick view of the grouping** which implies correlation.

Data preprocessing

PCA – Sorry, I still don't get it!

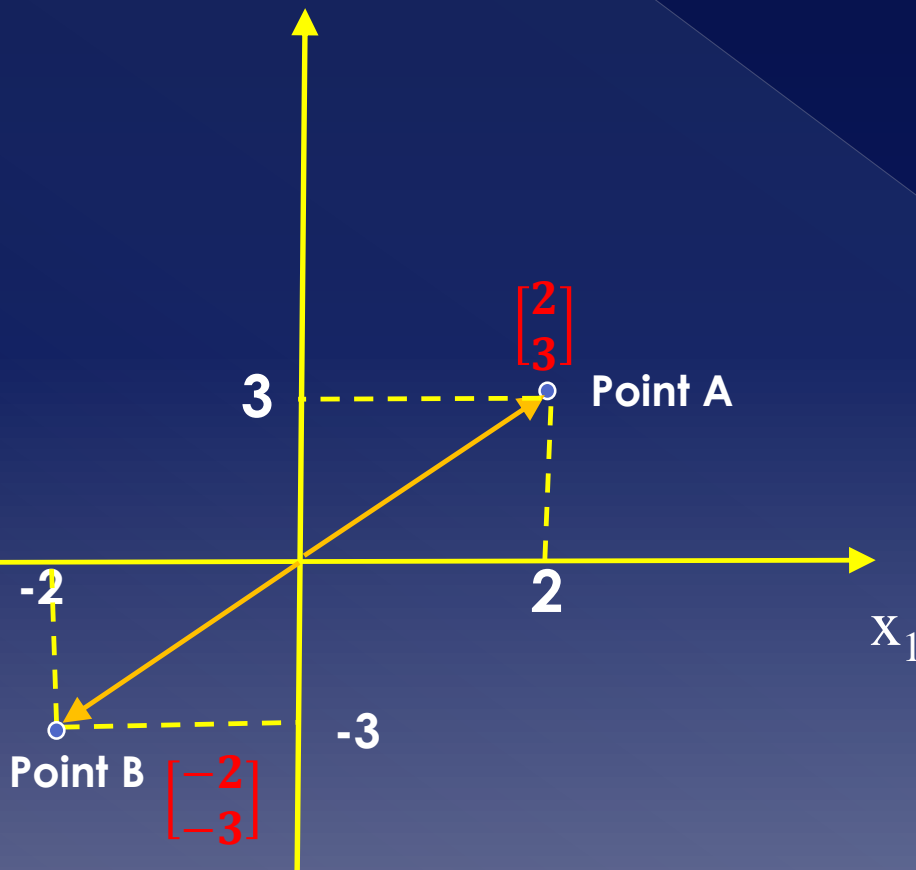
Principal Components Analysis (PCA) – I don't get it

- I'm sorry, I still don't get PCA. Get you draw it for me? **Ok, first some basics about matrix multiplication.**



Principal Components Analysis (PCA)

- Notice how point B is a reflection of Point A on the origin (0,0)?



The question is now: how can we calculate the reflection of any point P in the origin?

Easy: Just multiply the vector with the matrix $\begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}$ this will calculate the vector that is the reflection of the original e.g.

$$\begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} 2 \\ 3 \end{bmatrix} = \begin{bmatrix} -2 \\ -3 \end{bmatrix}$$

From this, please take away the following important message: Matrix multiplication is simply Vector TRANSFORMATIONS (i.e. move vector elsewhere)! Note: you can define matrices for any transformation. If you multiply matrices A and B with vector $\begin{bmatrix} -2 \\ -3 \end{bmatrix}$ i.e. $A*B*\begin{bmatrix} -2 \\ -3 \end{bmatrix}$ that means: transform vector $\begin{bmatrix} -2 \\ -3 \end{bmatrix}$ according to B and the result according to A. This may indicate e.g. Rotate and Mirror vector.

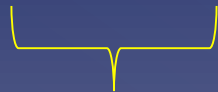
Principal Components Analysis (PCA)

- Multiplication of matrices are transformations

$$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$



$$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} 2 \\ 3 \end{bmatrix}$$



Imagine this matrix as a function f transforming vector $(2,3) \rightarrow$ reflection over x

$$\begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$$



Function or Filter for reflecting over y axis

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$



Function or Filter for reflecting over $y=x$ line

Principal Components Analysis (PCA)

- May also transform onto lower dimensional spaces

$$\underbrace{[2 \quad 7]}_{\text{Function transforming vector } [2 \ 3]} \begin{bmatrix} 2 \\ 3 \end{bmatrix}$$

$$= 25$$



This transformation $[2 \ 7]$ maps a vector onto a line i.e. one dimensional space.

Function
transforming vector
 $[2 \ 3]$

Principal Components Analysis (PCA)

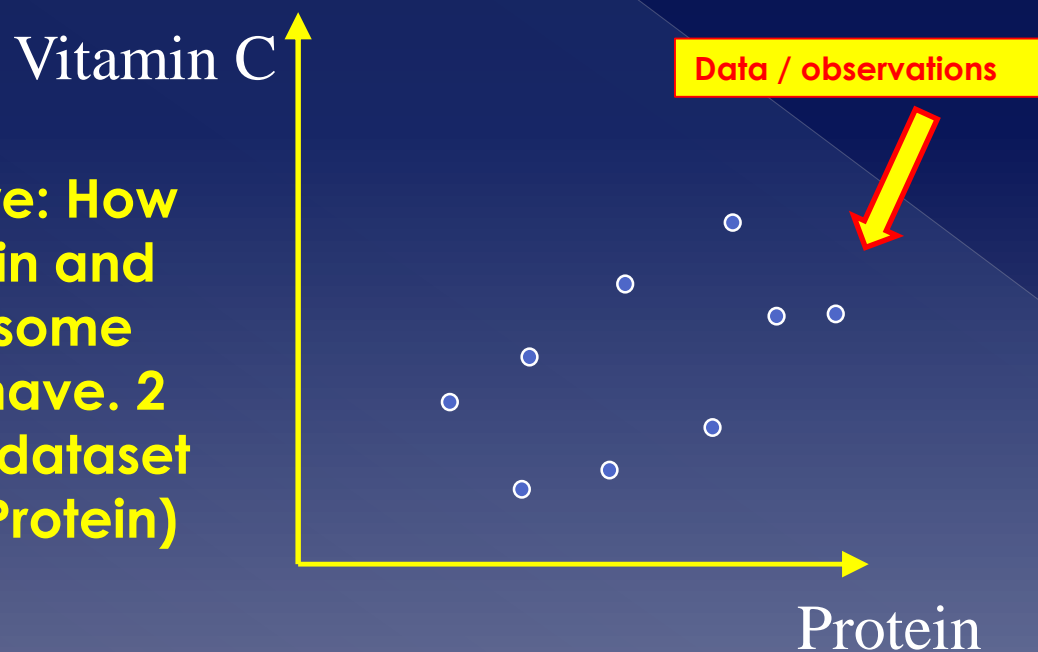
- Matrix multiplication is not multiplication!
It's transformation i.e. moving a point from one position on the same axes to another position, or moving a point from the current axis system to a new/different axis system.

Data preprocessing

PCA – A visual explanation

Principal Components Analysis (PCA)

- Let's assume we have some data with 2 variables (2 dimensions) and we want to reduce the number of variables to 1 while **keeping the variance, as much as possible, of the original data.**

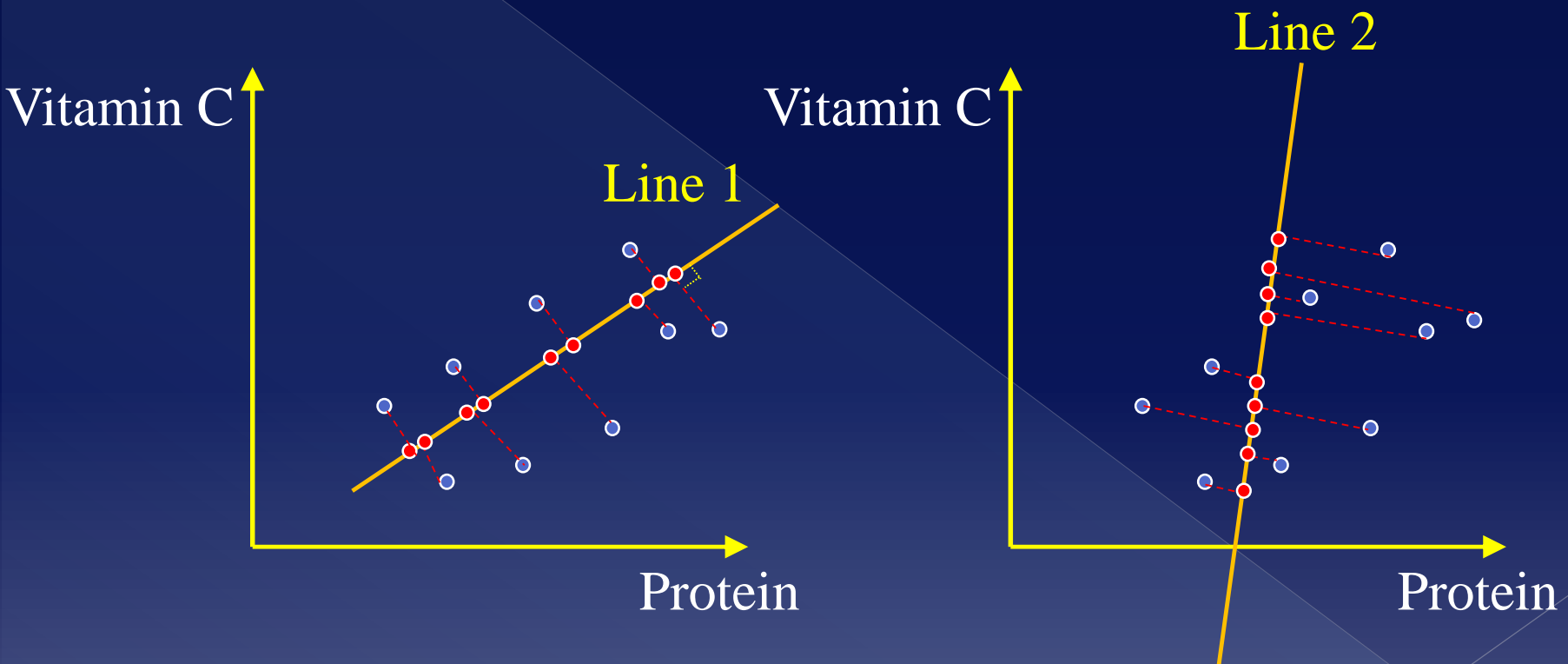


Example here: How much Protein and Vitamin C some food items have. 2 dimensional dataset (Vitamin C, Protein)

Principal Components Analysis (PCA)

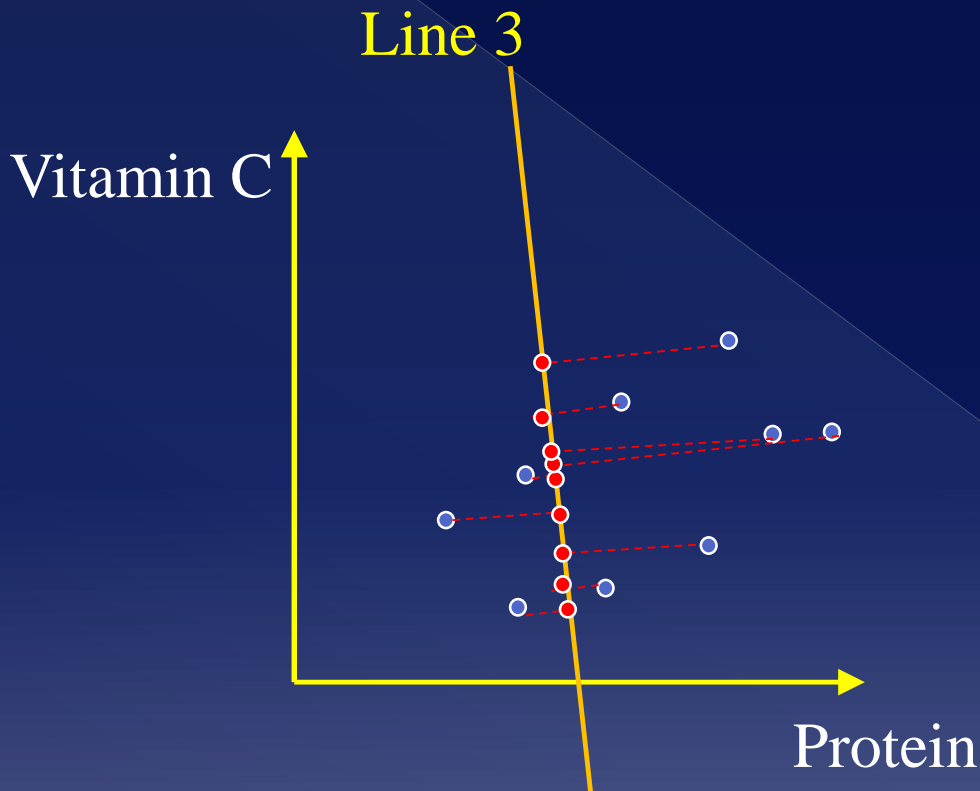
- Let's do the following now: **Draw random lines** on the plane of your data **and project the original data on that line. This would mean in essence projecting my 2-dimensional data on a 1-dimensional space.** How does this look like?
 - > This **line will be the new axis** upon which the 2-dimensional data will be projected and become 1-dimensional
 - > Our goal is to explain/keep most of the variance of the original dataset when the data is projected onto the new space. (**That's what PCA does**)

Principal Components Analysis (PCA)



Note: Red dots on Line 1 and Line 2 are the projections of the 2-dimensional data on each line (1-dimensional space). Projections are perpendicular to the lines. HINT: Notice how the “spread” (aka variance) of the red dots on these lines (Line1, Line2) differ?

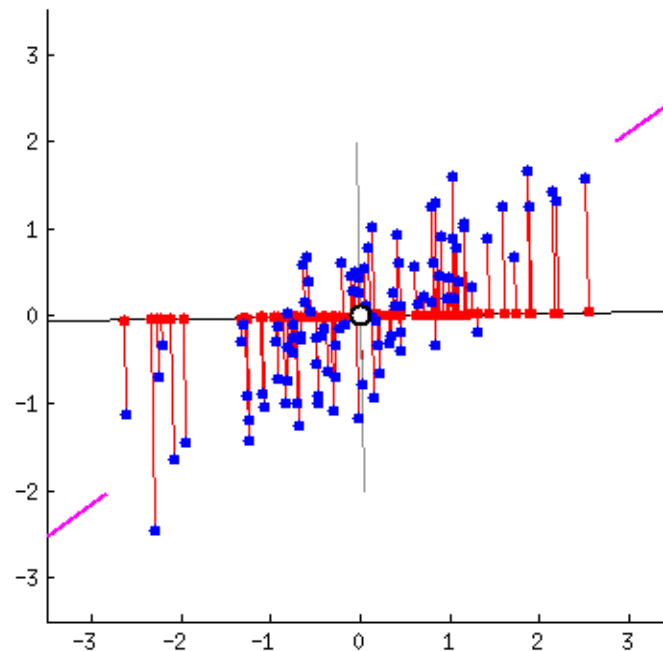
Principal Components Analysis (PCA)



Compare the spacing of the red dots (variance) on Line 3 to Line 1 and 2. See how the spread of red dots on Line 3 is well.... Smaller than on Lines 2 and 3? That means that Line 3 captures a smaller variance of the original dataset!

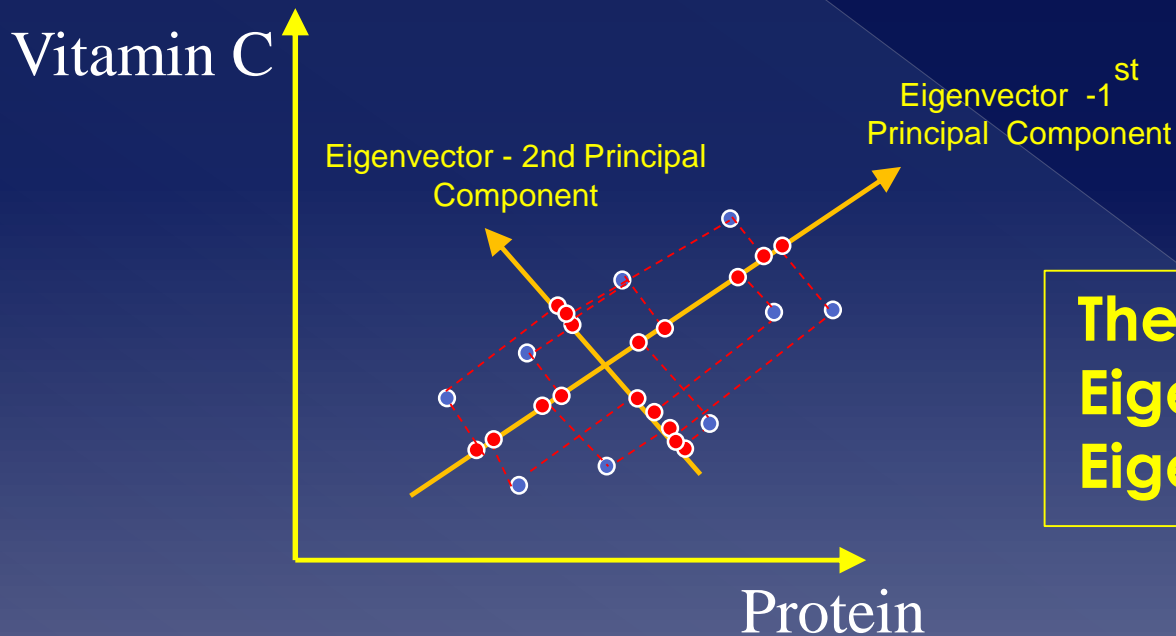
Principal Components Analysis (PCA)

You can draw indefinitely many such lines (see rotating line) and project the data onto them. On some lines, the “spread” of red dots i.e. variance of red dots on the line will be greater than on others. These lines (or vectors) are the Eigenvectors!



Principal Components Analysis (PCA)

- The line where the red dots have **the greatest variance (biggest spread)**, is an **Eigenvector** and the **First Principal Component** of our data! The **variance (spread)** of red dots are the **Eigenvalues of the Eigenvectors!**
 - > The line with **the second biggest spread** is the **second Principal Component**, the line with the **third biggest spread** is the **Third Principal Component** etc.



The length of the Eigenvector is its Eigenvalue λ

Principal Components Analysis (PCA)

- ⦿ Using **Eigenvalues/Eigenvectors** is **one way to do PCA**
- ⦿ **Other ways** also available
 - > E.g. **using Single Value Decomposition – SVD**
- ⦿ Both methods **yield to similar results**
 - > i.e. not much difference.

Principal Components Analysis (PCA)

● PCA in R: the Iris dataset – 1/3

```
#####  
#  
# A simple example of Principal Component Analysis on the built in iris dataset. #  
#  
# Iris is a dataset that contains observations of three iris varieties #  
# (Iris setosa, Iris virginica, Iris Versicolor ) recording five features #  
# for each observaion: Sepal length, Sepal width, Pedal lentgh, Pedal width and #  
# the species it belongs to (Setosa, Virginica, Versicolor ) #  
#  
# For a picture of the Iris plant and what is recorded in the dataset see: #  
# http://5047-presscdn.pagely.netdna-cdn.com/wp-content/uploads/2015/04/iris\_petal\_sepal.png #  
#  
# The Iris dataset contains a total of 150 observations, 50 from each variety. #  
#  
# The aim of the code is to perform Principal Component Analysis on the Iris dataset. #  
# Please note that this may not make sense for only 5 variables as PCA is mostly used in #  
# situations where the number of variables must be reduces, if these are large (dimensionality). #  
#  
#####  
  
# Let's take a quick look at the Iris dataset, by displaying the first 6 observations in the dataset.  
# Note: you can peek at more data by supplying a second argument like this: head(iris, 10) which  
# means show the first 10 observations of the iris dataset.  
head(iris)  
  
# By peeking at the dataset we see that it has 5 variables, of which the first 4 looks to be numerical values  
# and the last one (Species) nominal i.e. not numerical.  
  
# Let's take a summary of the iris data. This will show some first descriptive statistics.  
summary(iris)
```

cont...

PCA in R: the Iris dataset – 2/3

```
# Since the variable 'Species' has nominal values, we can't use them for PCA. PCA works only
# on numerical values. Hence, get rid of the column 'Species' and keep just the other ones.
# This can be done using the following command: iris[, 1:4] => This means from the iris dataset
# return all rows, but only columns 1 until 4 (i.e. excluding column 5 which is the 'Species' variable).
# We store this data in a new variable data pcaData.
# NOTE: We could also do the following: pcaData <- iris[, -5] which means all rows and all columns except
# column 5.
```

```
pcaData<-iris[,1:4]
```

```
# Let's take a look to see if everything is ok
head(pcaData, 12)
```

```
# Yep, looks ok. Now, we need to make sure that there are no
# missing values (that are displayed as NA).
# PCA does not work if there are missing values in any column/variable.
# Let's check this.
```

```
# Does variable Sepal.Length have any (at least 1) missing (na) value?
if (any(is.na(pcaData[,"Sepal.Length"]))) {
  sprintf("Sepal.Length has NA values")
} else
  sprintf("Sepal.Length is ok! ")
```

```
# Does variable Sepal.Width have any (at least 1) missing (na) value?
if (any(is.na(pcaData[,"Sepal.Width"]))) {
  sprintf("Sepal.Width has NA values")
} else
  sprintf("Sepal.Width is ok! ")
```

```
# Does variable Petal.Length have any (at least 1) missing (na) value?
if (any(is.na(pcaData[,"Petal.Length"]))) {
  sprintf("Petal.Length has NA values")
} else
  sprintf("Petal.Length is ok! ")
```

```
# Does variable Petal.Width have any (at least 1) missing (na) value?
if (any(is.na(pcaData[,"Petal.Width"]))) {
  sprintf("Petal.Width has NA values")
} else
  sprintf("Petal.Width is ok! ")
```

#cont...

PCA in R: the Iris dataset – 3/3

```
#
# OK, our data (pcaData) is ok. Now ready to execute PCA
#

# R's princomp() function is one way of executing PCA (there are also other functions available e.g. prcomp() ).
# We use princomp() because it can be configured to use the Covariance matrix to calculate Eigenvalues/Eigenvectors
# Please note that princomp() supports also performing PCA by using a Correlation matrix and SVD. This all depends
# on the parameters that you will provide.
#
# First parameter is our data (pcaData). Since we'll use the Covariance matrix and we signify this by setting
# the parameter cor equal to FALSE. Set to TRUE, princomp() will calculate the Covariance matrix. However you can
# provide yourself the Covariance matrix by setting the covmat parameter.
# If you set parameter cor to TRUE, the Correlation matrix will be used instead.
# Parameter score=TRUE tells princomp to transform each original observation to the new system defined by the principal
# components. Remember that principal components define a new coordinate system and the original data MUST be mapped
# properly onto this new coordinate system.
principalComponents<-princomp(pcaData, cor=FALSE, score=TRUE)

# Ok, done. Now the result of the princomp() function is a new object -that we store in a new variable
# called principalComponents- that has inside all necessary information. This object has attributes that
# you can access.
# But first, let's see what attributes it has.
attributes(principalComponents)

#You can see attributes such sdev, scores etc. You can display their values.
# Here, we display the calculated scores
principalComponents$scores

# Or, you can display a summary, which gives you a better overview.
summary(principalComponents)

# You may also plot the principal components to see the variances of each principal component
# in decreasing order
plot(principalComponents)

# Or you can plot the original data on a coordinate system defined by the two biggest principal
# components (note the bi- in biplot). Note that eigenvectors are the vertical and horizontal axes.
# The red vectors you see are pointing in the direction of the variables, as projected
# into the 2-d plane of the biplot.
biplot(principalComponents)
```

○ PCA in Python: the Iris dataset

```
#  
# Principal Component Analysis in Python  
#  
  
#  
# Load the required libraries.  
# Make sure that you installed properly the sklearn module that contains  
# the required functions and datasets  
#  
  
#Next two lines required to load the PCA function  
from sklearn import decomposition  
from sklearn.decomposition import PCA  
  
# sklearn comes with datasets. Make them available to this program  
from sklearn import datasets  
  
# Load the iris dataset  
iris = datasets.load_iris()  
  
# Get the data of the iris dataset into a new variable  
irisDataset = iris.data  
  
# Initialize the PCA function of the sklearn module.  
# Here, we say how many principal components we want. We will require  
# 2 principal components  
pca = decomposition.PCA(n_components=2)  
  
# Here we actually apply PCA to the iris dataset.  
# After executing PCA on the dataset, you can examine the properties  
# of the pca object to get the necessary information  
pca.fit(irisDataset)  
  
# Now transform the data of the Iris dataset to the new coordinate system,  
# defined by the two Eigenvectors that resulted from PCA  
transformedIrisDataset = pca.transform(irisDataset)
```

Discretization

- Discretization?
 - > **Divide the range of a continuous** attribute **into intervals**
 - > Some classification algorithms only accept categorical attributes.
 - > **Reduce data size by discretization**
 - > Prepare for further analysis
 - > Used in problems that require categorization and correlation analysis

Discretization

- ◎ Two ways to discretization

- > **Unsupervised**

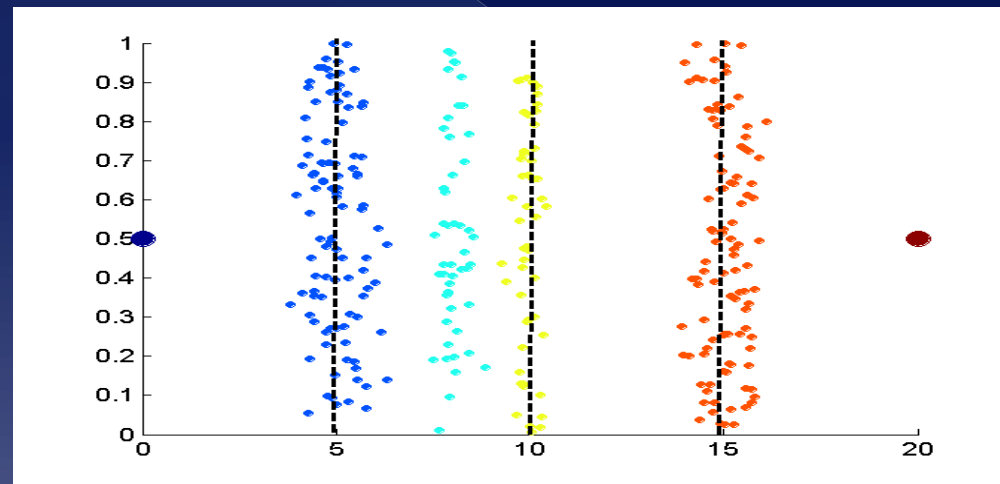
- Don't take into consideration the classes in which the data item belong

- > **Supervised**

- Take into consideration the classes in which data items belong

Discretization

- Unsupervised methods
 - > **Equal interval width** : Split range in **n equal spaces** by specifying **n-1 split points**

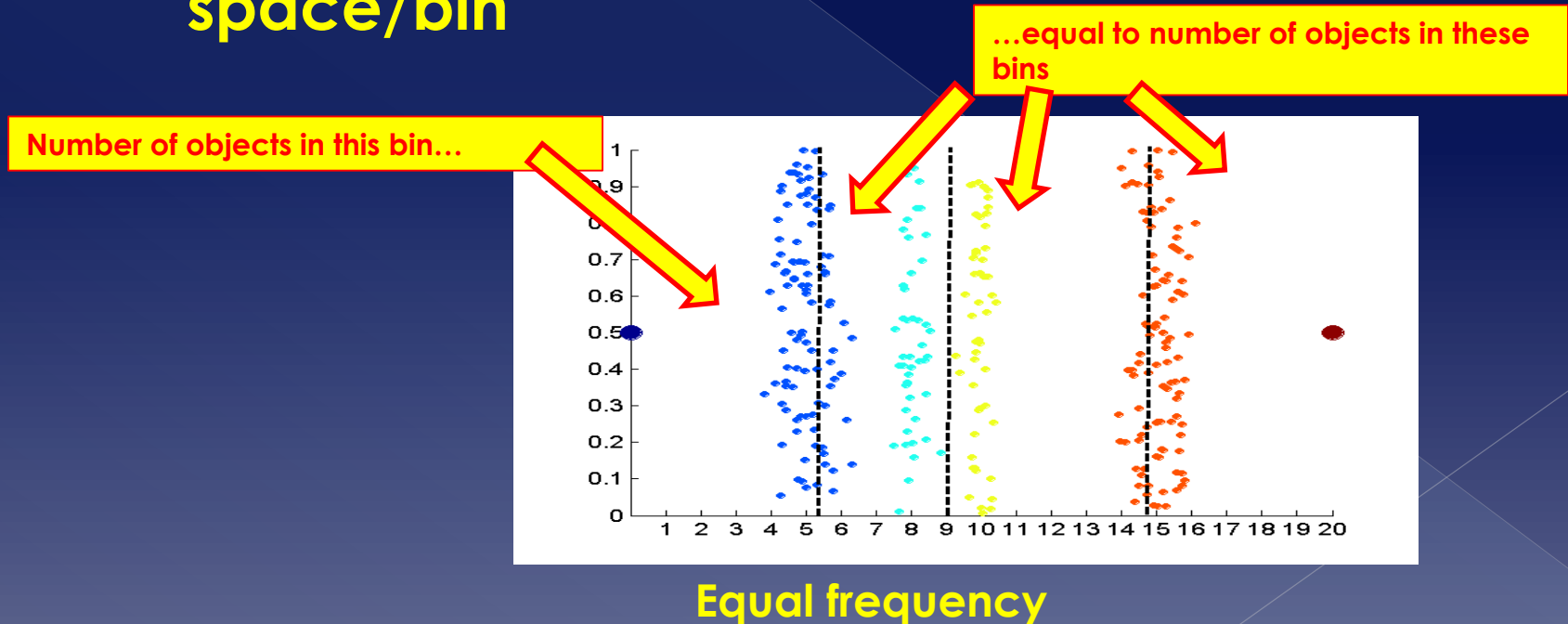


Equal interval width

Discretization

- Unsupervised methods

- > **Equal frequency**: Split range in spaces so **that equal number of data objects are in each space/bin**



Discretization

- Supervised methods
 - > Here **we look at some attribute (class)** of the data and try to take this into consideration when building the bins (hence supervised). Try to **improve quality of bins wrt class**.
 - > Bottom-up approach
 - Each item belongs to its own bin. Then try to produce bigger bins by evaluating some metrics
 - > Goal: create **bins that are as “clean” as possible wrt an attribute**, i.e. minimize “chaos”/”unorderly-ness” in each bin in terms of the class the items belong.

Discretization

- ◉ Can we **measure “chaos”/“unorderly-ness”** in each bin?
 - > Yup, that is what **Entropy** does
 - > Measuring entropy of bin e_i :

$$e_i = \sum_{j=1}^k \frac{m_{i,j}}{m_i} \log_2 \frac{m_{i,j}}{m_i}$$

...where k the number of different bins/classes, m_i the number of items in class i , $m_{i,j}$ the number of items that are in class j found in bin i . $m_{i,j}/m_i$ is the probability of class j in bin i .

Discretization

- Total entropy e of the spaces/partitioning is defined as

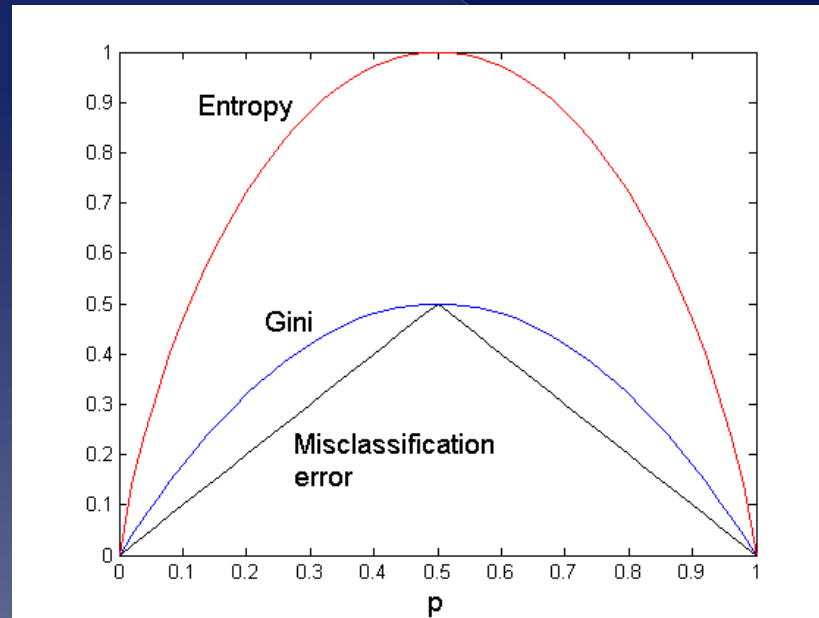
$$e = \sum_{i=1}^n \frac{m_i}{m} e_i$$

...where m total number of data items, m_i the number of data items in bin i (defined in terms of class)

Discretization

Some notes on Entropy

- > **If Entropy = 0** => no chaos, perfect order, clean space/partition. **Minimum entropy**
- > **If Entropy = 1** => biggest chaos, greatest “unorder”, most unclean space/partition. **Maximum entropy**



Similarity and dissimilarity measures

Similarity and dissimilarity measures

◎ **Similarity**

- > Numerical measure of how alike two data objects are.
- > Is higher when objects are more alike.
- > Often falls in the range $[0,1]$

◎ **Dissimilarity**

- > Numerical measure of how different two data objects are
- > Lower when objects are more alike
- > Minimum dissimilarity is often 0
- > Upper limit varies

◎ **Proximity** refers to a similarity or dissimilarity

Similarity and dissimilarity measures

- For simple attributes
 - Note: q, p below are attribute values for two data objects**
 - s, d below stand for (s)imilarity and d(istance)**

Attribute Type	Dissimilarity	Similarity
Nominal	$d = \begin{cases} 0 & \text{if } p = q \\ 1 & \text{if } p \neq q \end{cases}$	$s = \begin{cases} 1 & \text{if } p = q \\ 0 & \text{if } p \neq q \end{cases}$
Ordinal	$d = \frac{ p-q }{n-1}$ <p>(values mapped to integers 0 to $n-1$, where n is the number of values)</p>	$s = 1 - \frac{ p-q }{n-1}$
Interval or Ratio	$d = p - q $	$s = -d, s = \frac{1}{1+d} \text{ or } s = 1 - \frac{d - \min_d}{\max_d - \min_d}$

Table 5.1. Similarity and dissimilarity for simple attributes

Similarity and dissimilarity measures

Distance

- > is an **dissimilarity measure**
- > Observe that **Dissimilarity and Distance** are same things
 - You use distance to measure similarity/dissimilarity
 - You **transform distance** in order to calculate similarity/dissimilarity e.g. ***similarity*** = $\frac{1}{\text{distance}(p_1, p_w)}$ **or**

similarity = $\frac{1}{e^{\text{distance}(p_1, p_2)}}$, etc. In general you choose the proper formula.

Different ways to measure distance

- > **Euclidian** distance
- > **Minkowski** distance
- > **Mahalanobis** distance

Similarity and dissimilarity measures

- You can define **your own distance measure**.
- However, in order to be considered a proper distance measure, it must be a metric. Or more clearly **it has to have the following properties:**

1. $d(x, y) \geq 0$

2. $d(x, y) = 0$ iff $x = y$

3. $d(x, y) = d(y, x)$

4. $d(x, z) \leq d(x, y) + d(y, z)$

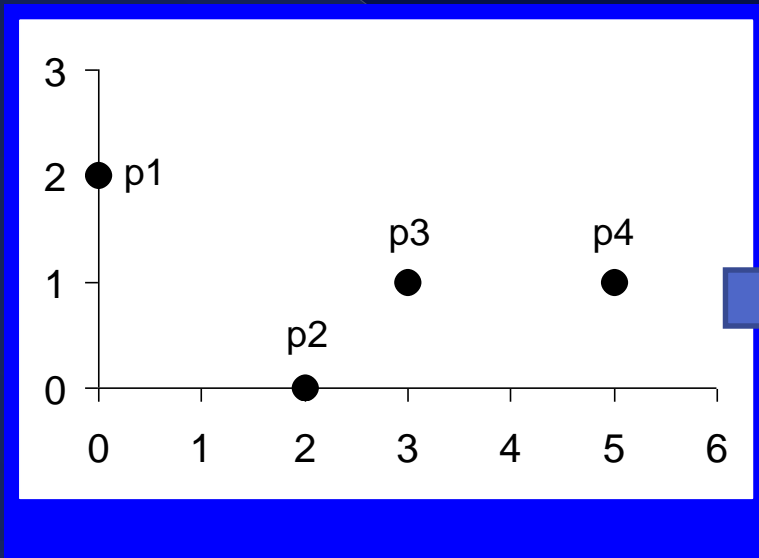
Euclidian Distance

- **“Works” for points x , y in one, two, three or more dimensions**
- (known) Formula

$$d(x, y) = \sqrt{\sum_{k=1}^n (x_k - y_k)^2}$$

...where n is the number of dimensions (attributes) and x_k and y_k are, respectively, the k^{th} attributes (components) or data objects x and y .

Euclidian Distance



point	x	y
p1	0	2
p2	2	0
p3	3	1
p4	5	1

Data objects

	p1	p2	p3	p4
p1	0	2.828	3.162	5.099
p2	2.828	0	1.414	3.162
p3	3.162	1.414	0	2
p4	5.099	3.162	2	0

Euclidean Distance Matrix

Minkowski distance

- Minkowski distance **is a generalization of Euclidean Distance**

$$d(x, y) = \sqrt[r]{\sum_{k=1}^n |x_k - y_k|^r}$$

...where r is a parameter, n is the number of dimensions (attributes) and x_k and y_k are, respectively, the **kth attributes** (components) or **data objects x and y**.

Minkowski distance

- ◉ **Special cases of the Minkowski distance:**
- ◉ **$r = 1$.** City block (Manhattan distance, taxicab, L_1 norm) distance.
 - > A common example of this is the **Hamming distance**, which is just the number of bits that are different between two binary vectors
- ◉ **$r = 2$.** Euclidean distance
- ◉ **$r \rightarrow \infty$.** “Supremum” (L_{\max} norm, L_{∞} norm) distance.
 - > This is the maximum difference between any component of the vectors
- ◉ **IMPORTANT!** Do not confuse r with n , i.e., all these distances are defined for all numbers of dimensions.

Minkowski distance

Manhattan distance, $r=1$



L1	p1	p2	p3	p4
p1	0	4	4	6
p2	4	0	2	4
p3	4	2	0	2
p4	6	4	2	0

Euclidean distance, $r=2$



L2	p1	p2	p3	p4
p1	0	2.828	3.162	5.099
p2	2.828	0	1.414	3.162
p3	3.162	1.414	0	2
p4	5.099	3.162	2	0

Distance $r \rightarrow \infty$,
 L_∞ norm



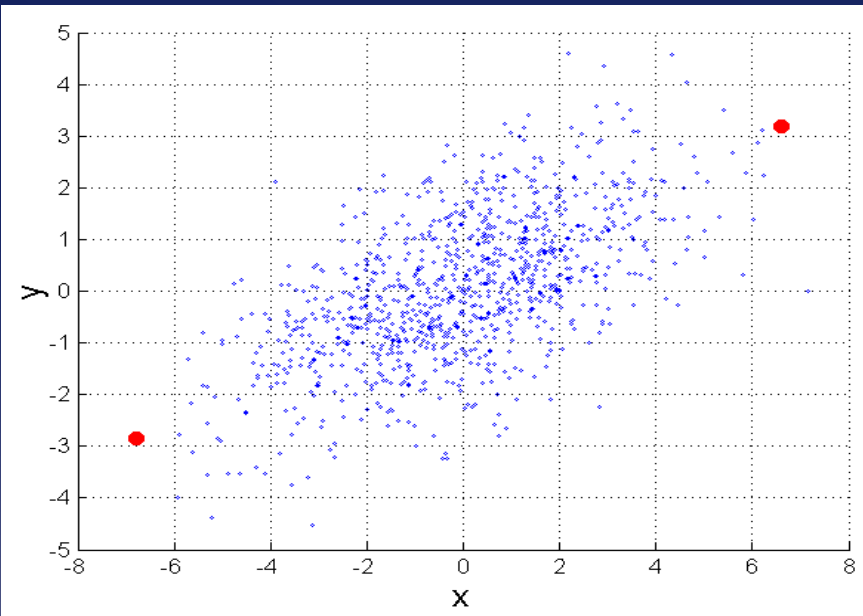
L_∞	p1	p2	p3	p4
p1	0	2	3	5
p2	2	0	1	3
p3	3	1	0	2
p4	5	3	2	0

point	x	y
p1	0	2
p2	2	0
p3	3	1
p4	5	1

Distance Matrices

Mahalanobis distance

- Is the distance **between a point p and a distribution D**
 - If **Mahalanobis distance = 0**, then point is at the mean of D



$$d(x, y) = (p - q)\Sigma^{-1}(p - q)^T$$

Σ is the covariance matrix of the input data X

$$\Sigma_{l,k} = \frac{1}{n-1} \sum_{i=1}^n (X_{ij} - \bar{X}_j)(X_{ik} - \bar{X}_k)$$

For red points, the Euclidean distance is 14.7, Mahalanobis distance is 6.

Cosine similarity

- **Applies to document data**

- If d_1 and d_2 are two document vectors, then

$$\cos(d_1, d_2) = (d_1 \bullet d_2) / (||d_1|| ||d_2||),$$

where \bullet indicates vector dot product and $||d||$ is the length of vector d .

- Example:

$$d_1 = (3, 2, 0, 5, 0, 0, 0, 2, 0, 0)$$

$$d_2 = (1, 0, 0, 0, 0, 0, 0, 1, 0, 2)$$

$$d_1 \bullet d_2 = 3*1 + 2*0 + 0*0 + 5*0 + 0*0 + 0*0 + 0*0 + 2*1 + 0*0 + 0*2 = 5$$

$$||d_1|| = (3*3 + 2*2 + 0*0 + 5*5 + 0*0 + 0*0 + 0*0 + 2*2 + 0*0 + 0*0)^{0.5} = (42)^{0.5} = 6.481$$

$$||d_2|| = (1*1 + 0*0 + 0*0 + 0*0 + 0*0 + 0*0 + 0*0 + 1*1 + 0*0 + 2*2)^{0.5} = (6)^{0.5} = 2.245$$

$$\text{Hence, } \cos(d_1, d_2) = .3150$$

Summary

Summary

- **Running time** of algorithms is characterized in terms of **number of steps they take to solve a problem**
 - > Number of steps expressed as a function of the problem size, $T(n)$ (in general **polynomial time alg = good! Exponential time algorithm = bad!**)
 - **Problem size**: aspect of input data that will influence number of steps
 - Use of asymptotic notations to characterize **running time** (referred to as **time complexity**): $O()$, $\Theta()$
- Data has different types of attributes depending on the type of values they may take

Summary

- ◎ **Different types** imply **different methods of analysis**
 - > Different methods of analysis work for on different types of data
- ◎ **Preprocessing** is one of the most important steps in data mining
 - > Consumes most of the time (70-80% of dm tasks)
- ◎ There are **different objectives** when preprocessing data
 - > Reducing dimensions
 - > Sampling
 - > Discretization

Summary

- ◎ **PCA** most powerful way to reduce dimensions of the dataset (curse of dimensionality) which causes problems in **Big Data**
 - > Used in many-many Big Data environments
- ◎ There are also **different distance metrics**
 - > Depending on objective of task at hand
- ◎ In general, **choose wisely, the appropriate, types of values, preprocessing methods and distance metrics**
 - > Will **influence** your data mining **results!**

Appendices

APPENDIX A: Related Bibliography

- D. P. Ballou and G. K. Tayi. Enhancing data quality in data warehouse environments. *Communications of ACM*, 42:73-78, 1999
- T. Dasu and T. Johnson. *Exploratory Data Mining and Data Cleaning*. John Wiley & Sons, 2003
- T. Dasu, T. Johnson, S. Muthukrishnan, V. Shkapenyuk. [Mining Database Structure; Or, How to Build a Data Quality Browser](#). SIGMOD'02.
- H.V. Jagadish et al., Special Issue on Data Reduction Techniques. *Bulletin of the Technical Committee on Data Engineering*, 20(4), December 1997
- D. Pyle. *Data Preparation for Data Mining*. Morgan Kaufmann, 1999
- E. Rahm and H. H. Do. Data Cleaning: Problems and Current Approaches. *IEEE Bulletin of the Technical Committee on Data Engineering*. Vol.23, No.4
- V. Raman and J. Hellerstein. *Potters Wheel: An Interactive Framework for Data Cleaning and Transformation*, VLDB'2001
- T. Redman. *Data Quality: Management and Technology*. Bantam Books, 1992
- Y. Wand and R. Wang. Anchoring data quality dimensions ontological foundations. *Communications of ACM*, 39:86-95, 1996
- R. Wang, V. Storey, and C. Firth. A framework for analysis of data quality research. *IEEE Trans. Knowledge and Data Engineering*, 7:623-640, 1995