Working with Data

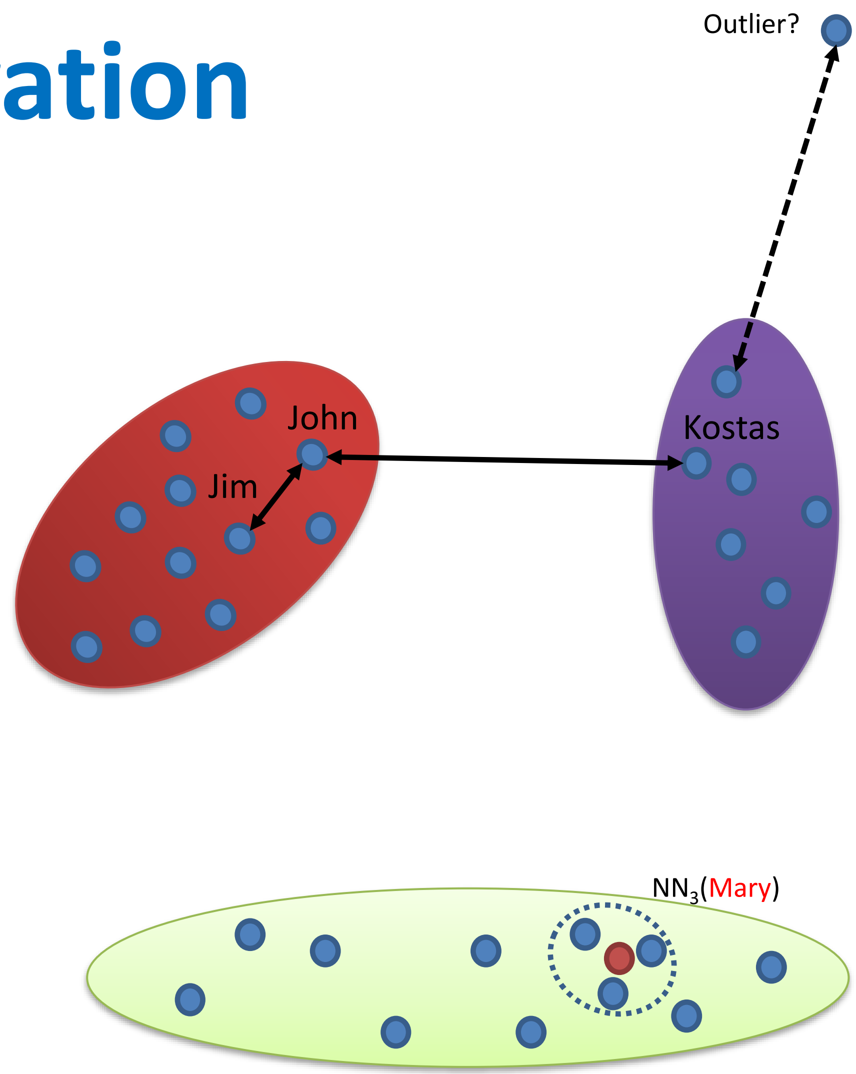
Yannis Kotidis

Department of Informatics

Athens University of Economics and Business

Motivation

- We often need ways to assess how similar or dissimilar objects are in comparison to one another
- Examples: clustering, outlier analysis, nearest-neighbor search, recommendation, visualization, classification



Simple Running Example

- Car dealership
  - a customer inquired about car #1 that was sold
  - which of the other cars is she most likely to buy?

Car	Color	Condition	Mileage (*1000)
1	Blue	Excellent	45
2	Green	Fair	22
3	Red	Good	64
4	Blue	Excellent	28

Roadmap

- We will first discuss simple similarity metrics for common data types: nominal, ordinal and numerical attributes
- We will then extend our techniques to address more complex scenarios
  - Hierarchical domains (e.g. product ⇔ categories)
  - Sets (e.g. basket data), Bags
  - Vectors (multidimensional data)
  - Strings
  - Time Series
  - Graphs

Preliminaries

- Let sim(a,b) denote the similarity of two values a, b of a data attribute
- In order to have a common basis, we will normalize values: sim(a,b) in range [0..1]
  - sim(a,b) = 1, iff a and b are identical
  - sim(a,b) = 0, iff a and b are unalike

- Dissimilarity: d(a,b) = 1 – sim(a,b)
  - Formula assumes sim(a,b) $\in$ [0..1]
  - Notice that, depending on the internals of the sim() function, dissimilarity is not necessarily a distance function (i.e. triangle inequality may not hold)

Running Example

- Dataset describing used cars
- 3 known attributes : *color, condition, mileage* (in 1000 Km)

Car	Color	Condition	Mileage (*1000 Km)
1	Blue	Excellent	45
2	Green	Fair	22
3	Red	Good	64
4	Blue	Excellent	28

Nominal Attributes

- The values are symbols (e.g., names of things) that often represent some category or state
  - Are a type of categorical attributes when there is no ordering/importance implied by the values
    - Cats are not better than dogs or vice-versa
  - Can be used to differentiate records (e.g., pet="cat" vs pet="dog")
- Numerical attributes (e.g., product-ids) may also be treated as nominal

Dissimilarity of nominal attributes

- Let us define

$$d(a,b) = 1 \text{ if } a \neq b, 0 \text{ otherwise}$$

- Examples
  - d(Blue,Green) =1

X

  - d(Green,Red) = 1

X

  - d(Green,Green) = 0

✓

Dissimilarity of nominal attributes

- We can form a ***dissimilarity matrix*** for Color:

$$
\begin{array}{rc}
\begin{array}{r} \\ \text{Car 2} \\ \\ \\ \end{array} & \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix} \\
 & \begin{array}{llll} \text{Car 1} & \quad & \quad & \quad \end{array}
\end{array}
$$

Car	Color	Condition	Mileage (*1000)
1	Blue	Excellent	45
2	Green	Fair	22
3	Red	Good	64
4	Blue	Excellent	28

Note

- Even for medium-sized datasets, the dissimilarity matrix may not fit in memory as it requires $O(n^2)$ space

- We use it in our examples in order to visually inspect the computed pair-wise dissimilarities

Dissimilarity of *ordinal* attributes

- In this case there is an ordering of the symbols
- Consider rank of different values
  - Fair (1) < Good (2) < Excellent (3)
- Let

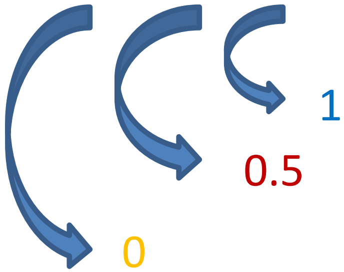
$$d(a,b) = \frac{|rank(a) - rank(b)|}{maxrank - \mathrm{min}rank}$$

- Examples
  - d(Fair, Good)=$\frac{|1-2|}{3-1}$= 0.5
  - d(Excellent, Fair) = $\frac{|3-1|}{3-1}$ = 1

Dissimilarity of ordinal attributes

- Dissimilarity matrix for Condition:

$$
\begin{bmatrix}
0 & & & \\
1 & 0 & & \\
0.5 & 0.5 & 0 & \\
0 & 1 & 0.5 & 0
\end{bmatrix}
$$



Car	Color	Condition	Mileage (*1000)
1	Blue	Excellent	45
2	Green	Fair	22
3	Red	Good	64
4	Blue	Excellent	28

Dissimilarity of numerical attributes

- Compute:

$$d(a,b) = \frac{|a-b|}{maxvalue - \mathrm{min}value}$$

- Example
  - d(45,22)= $\frac{|45-22|}{64-22}$ = 0.55

Car	Color	Condition	Mileage (*1000)
1	Blue	Excellent	45
2	Green	Fair	22
3	Red	Good	64
4	Blue	Excellent	28

Dissimilarity of numerical attributes

- Dissimilarity matrix for Mileage:

$$\begin{bmatrix} 0 & & & \\ 0.55 & 0 & & \\ 0.45 & 1 & 0 & \\ 0.40 & 0.14 & 0.86 & 0 \end{bmatrix}$$

Car	Color	Condition	Mileage (*1000)
1	Blue	Excellent	45
2	Green	Fair	22
3	Red	Good	64
4	Blue	Excellent	28

Combining scores

- Simplest approach: take average of computed dissimilarities

$$d(r_a,r_b)= \frac{1}{3} *(d_{color}(r_a,r_b)+d_{condition}(r_a,r_b)+d_{mileage}(r_a,r_b))$$

- Use weights to prioritize certain attributes
  - e.g. user prioritizes mileage over color ($w_3 > w_1$)

$$d(r_a, r_b)= \frac{w_1 * d_{color}(ra,r_b)+w_2 * d_{condition}(ra,r_b)+w_3 * d_{mileage}(ra,r_b)}{w_1+w_2+w_3}$$

Example (weighted average)

- Assume mileage is more (2x) important than color, condition
  - $w_{color}=1$, $w_{condition}=1$, $w_{mileage}=2$

$$d(r_a, r_b) = \frac{d_{color}(ra, r_b) + d_{condition}(ra, r_b) + 2 * d_{mileage}(ra, r_b)}{1+1+2}$$

Take avg of dissimilarity matrices

$$\frac{\begin{bmatrix} 0 & & & \\ 1 & 0 & & \\ 1 & 1 & 0 & \\ 0 & 1 & 1 & 0 \end{bmatrix} + \begin{bmatrix} 0 & & & \\ 1 & 0 & & \\ 0.5 & 0.5 & 0 & \\ 0 & 1 & 0.5 & 0 \end{bmatrix} + \begin{bmatrix} 0 & & & \\ 0.55 & 0 & & \\ 0.45 & 1 & 0 & \\ 0.40 & 0.14 & 0.86 & 0 \end{bmatrix}}{3}$$

$$\Rightarrow \begin{bmatrix} 0 & & & \\ 0.85 & 0 & & \\ 0.65 & 0.83 & 0 & \\ 0.13 & 0.71 & 0.79 & 0 \end{bmatrix}$$

Outcome

- Data Matrix:

Car	Color	Condition	Millage (*1000)
1	Blue	Excellent	45
2	Green	Fair	22
3	Red	Good	64
4	Blue	Excellent	28

- Dissimilarity Matrix:

$$\begin{bmatrix} 0 & & & \\ 0.85 & 0 & & \\ 0.65 & 0.83 & 0 & \\ 0.13 & 0.71 & 0.79 & 0 \end{bmatrix}$$

Most similar pair of cars?

Most similar pair of cars

- Data Matrix:

Car	Color	Condition	Millage (*1000)
1	**Blue**	**Excellent**	**45**
2	Green	Fair	22
3	Red	Good	64
4	**Blue**	**Excellent**	**28**

- Dissimilarity Matrix:

$$\begin{bmatrix} 0 & & & \\ 0.85 & 0 & & \\ 0.65 & 0.83 & 0 & \\ \mathbf{0.13} & 0.71 & 0.79 & 0 \end{bmatrix}$$

John: I like Car #3

- Data Matrix:

Car	Color	Condition	Millage (*1000)
1	Blue	Excellent	45
2	Green	Fair	22
3	Red	Good	64
4	Blue	Excellent	28

- Dissimilarity Matrix:

$$\begin{bmatrix} 0 & & & \\ 0.85 & 0 & & \\ \boxed{0.65 \quad 0.83} & & \boxed{0} & \\ 0.13 & 0.71 & \boxed{0.79} & 0 \end{bmatrix}$$

Can we rank the cars on our lot based on their dissimilarities to car #3?

Nearest Neighbors of Car #3

- Data Matrix:

Car	Color	Condition	Millage (*1000)
1	Blue	Excellent	45
2	Green	Fair	22
3	Red	Good	64
4	Blue	Excellent	28

- Dissimilarity Matrix:

$$\begin{bmatrix} 0 & & & \\ 0.85 & 0 & & \\ 0.65 & 0.83 & 0 & \\ 0.13 & 0.71 & 0.79 & 0 \end{bmatrix}$$

NN(Car #3):
Car#1 (d(1,3) = 0.65)
Car#4 (d(4,3) = 0.79)
Car#2 (d(3,2) = 0.83)

Most similar

Less similar

Extended Data Matrix

Car	Color	Condition	Millage (*1000)	Type
1	Blue	Excellent	45	Supermini
2	Green	Fair	22	Crossover
3	Red	Good	64	SUV
4	Blue	Excellent	28	Small family

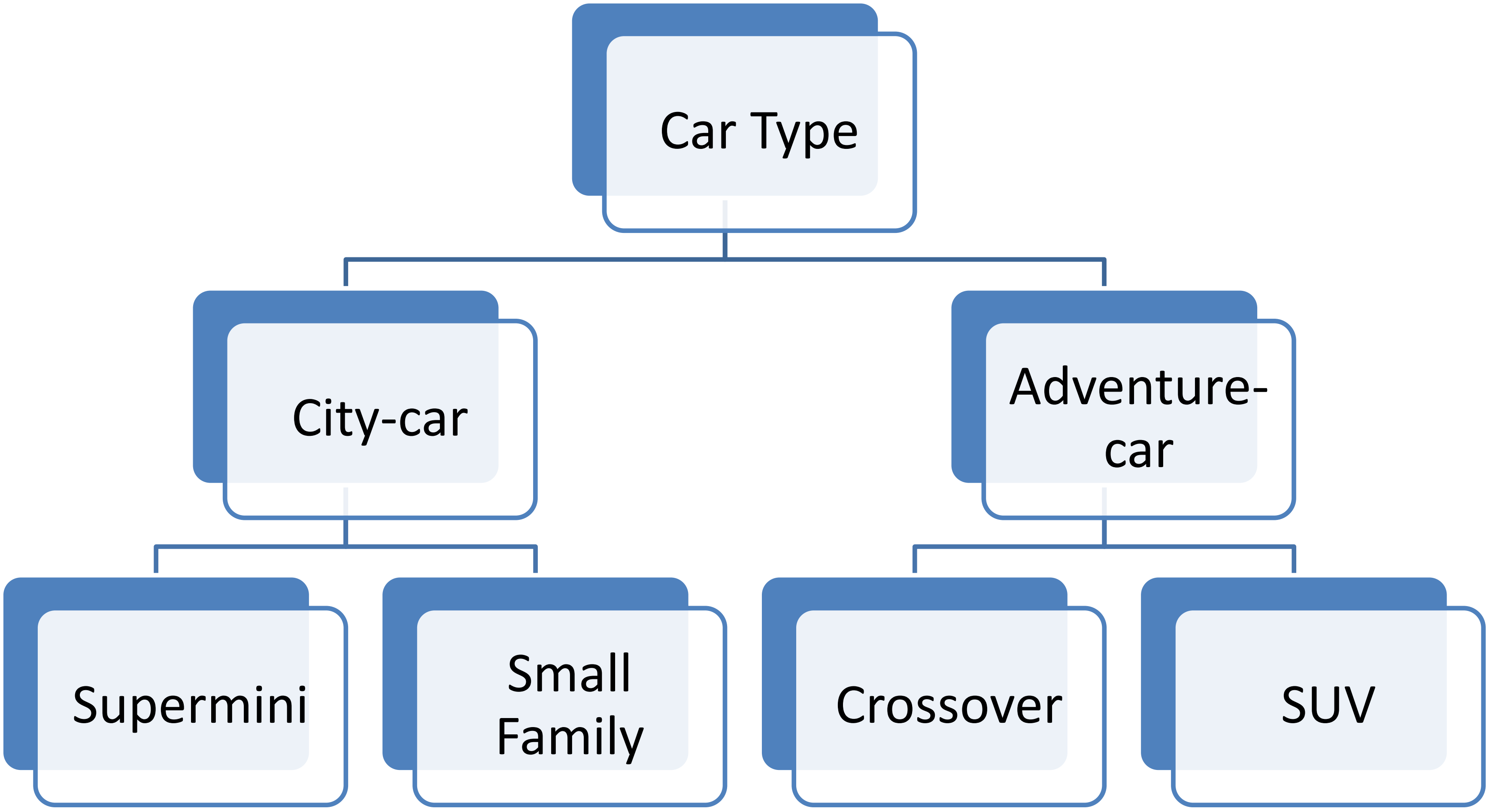
- How do we treat the newly added Type attribute?

Extended Data Matrix

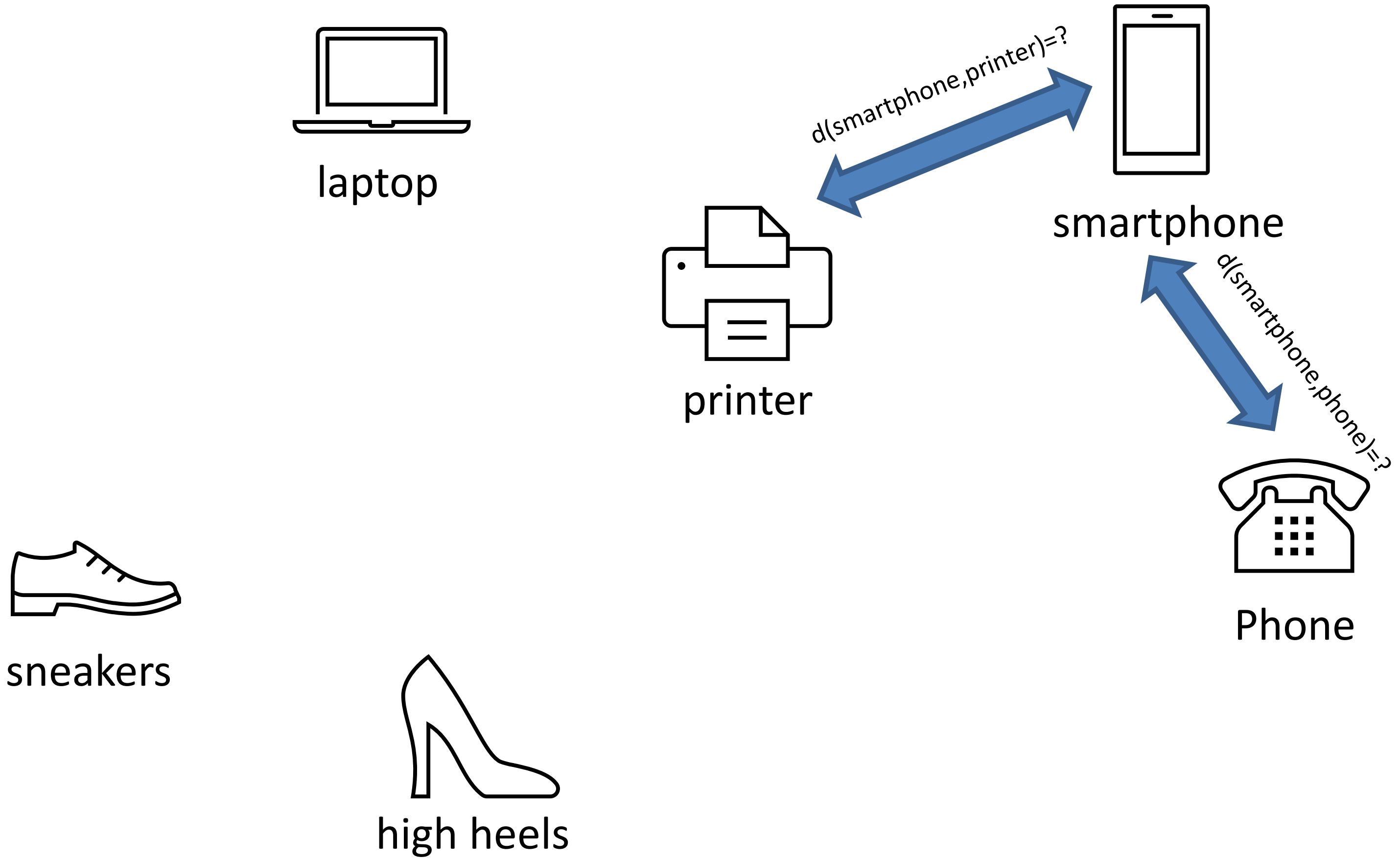
Car	Color	Condition	Millage (*1000)	Type
1	Blue	Excellent	45	Supermini
2	Green	Fair	22	Crossover
3	Red	Good	64	SUV
4	Blue	Excellent	28	Small family

- How do we treat the newly added Type attribute?
  - Small family cars more similar to superminis?
  - SUVs more similar to Crossover?

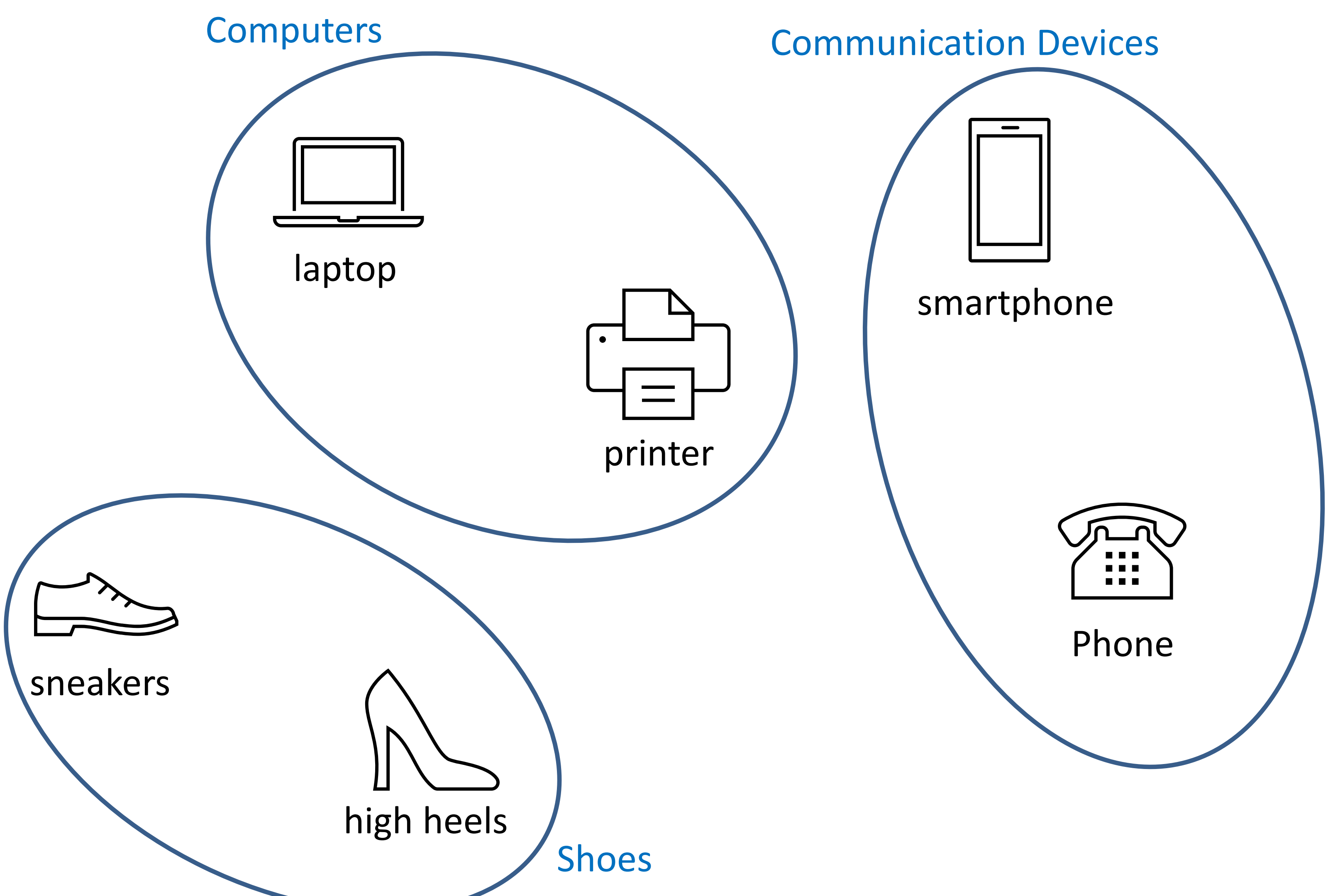
Car-Type Hierarchy?



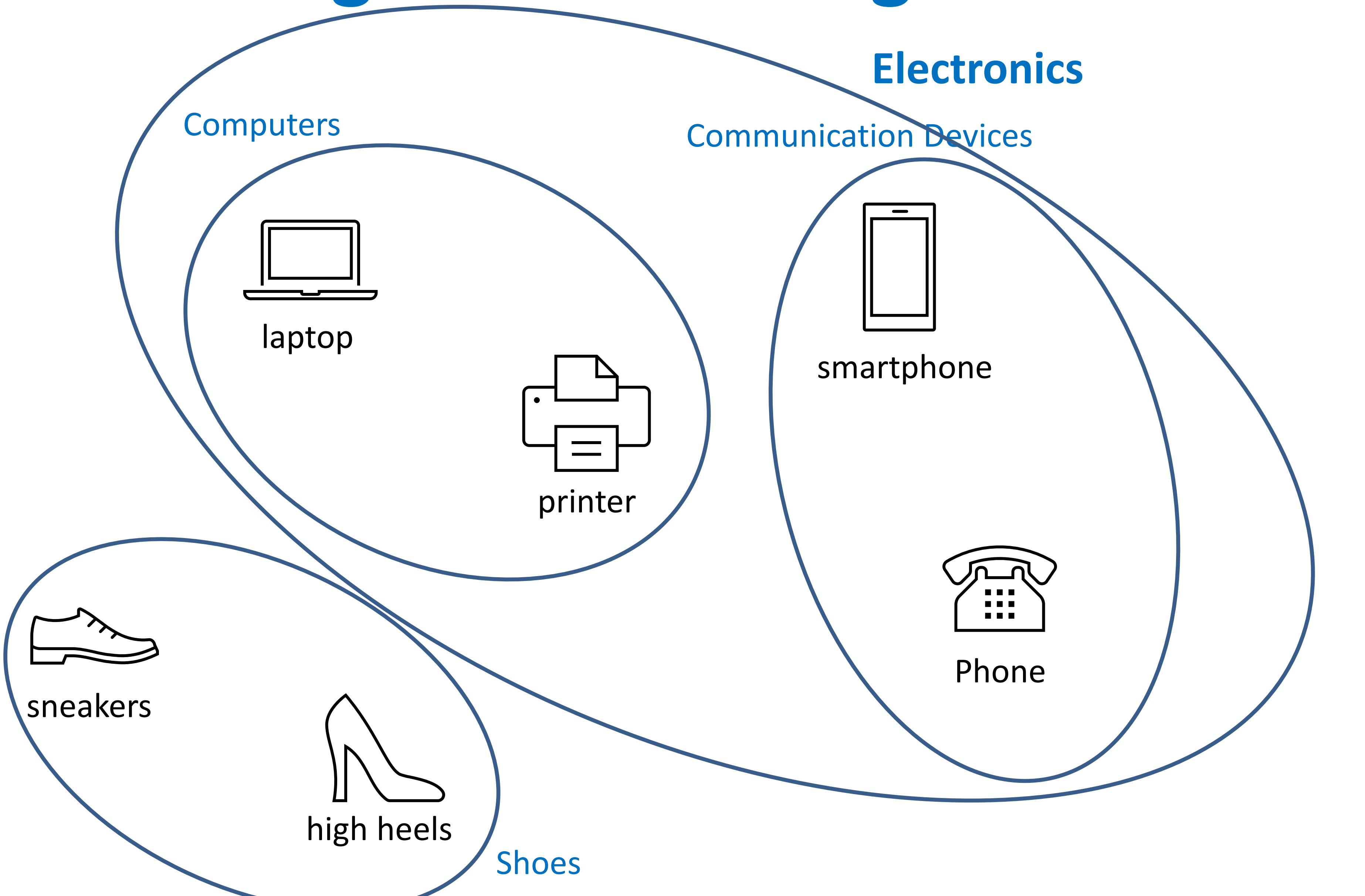
Six products



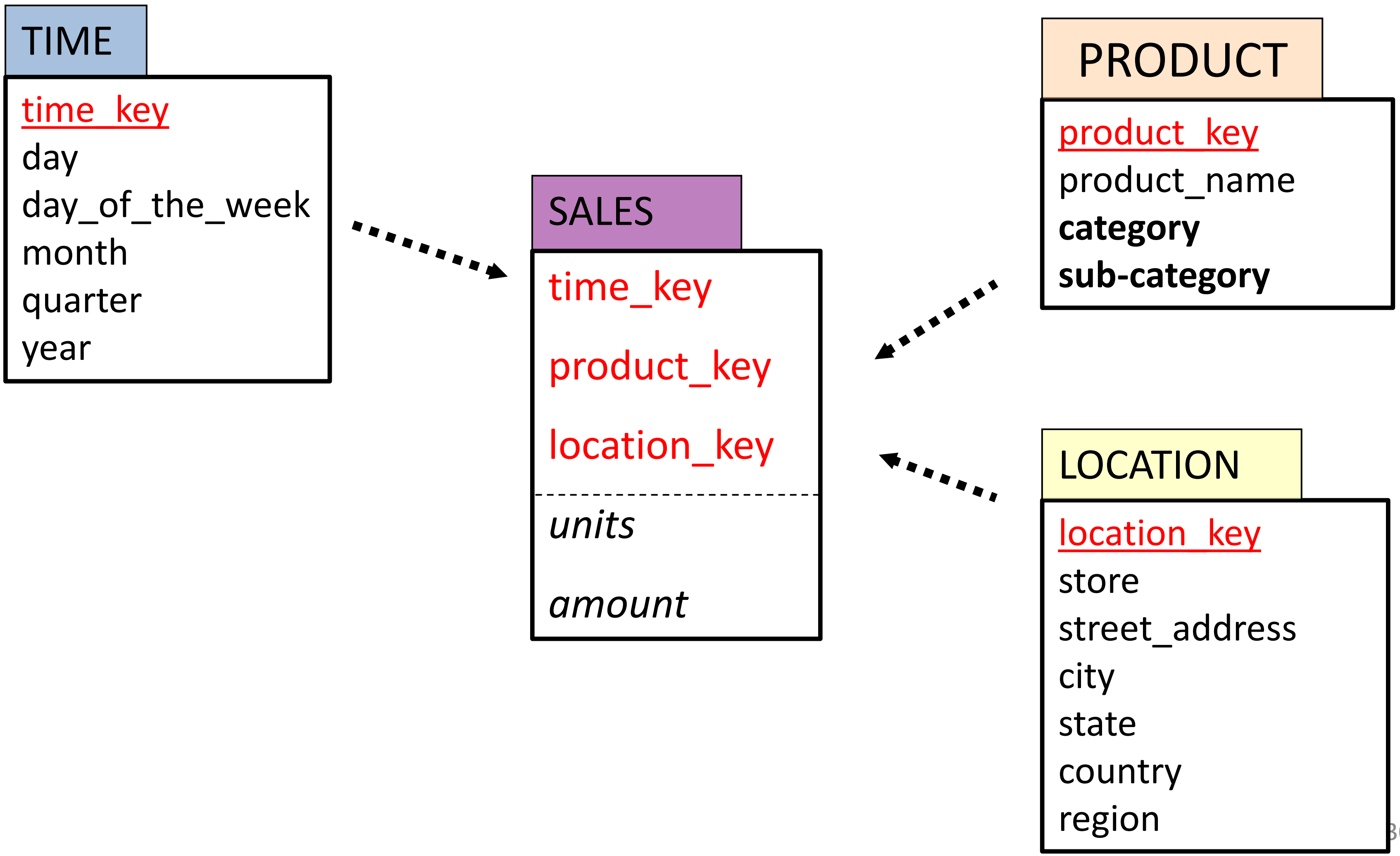
Groups/Categories



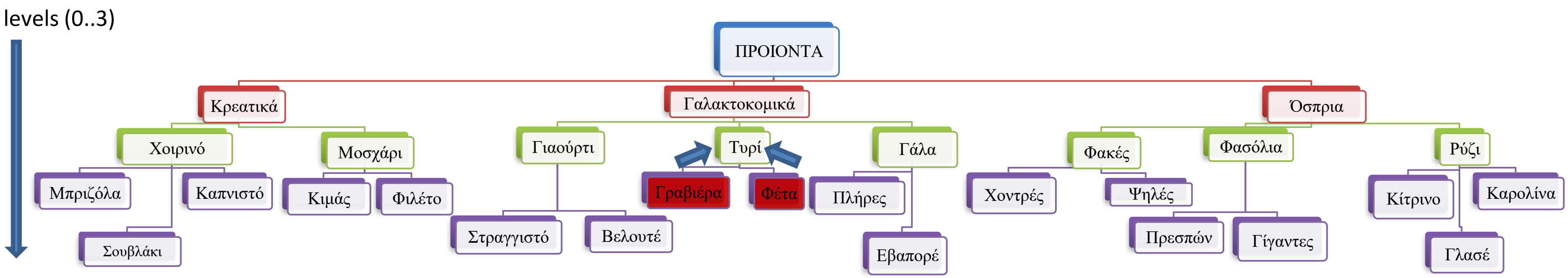
Higher-level Categories



Utilize the Star Schema

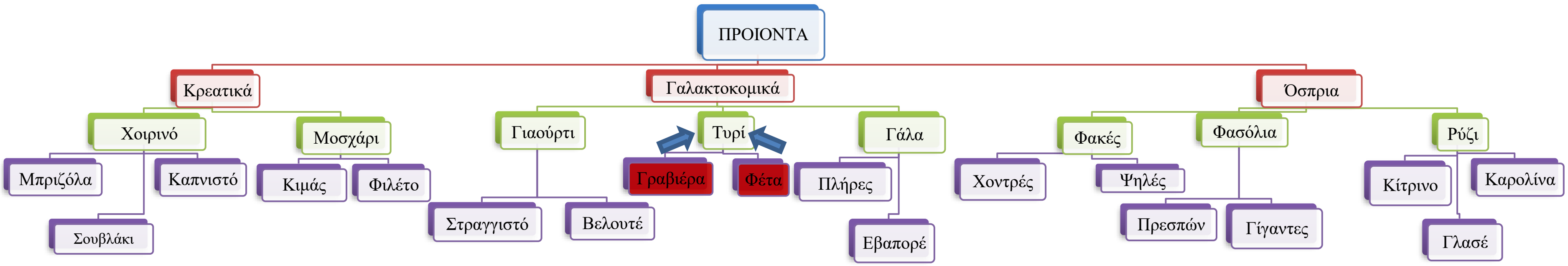


Using Hierarchies (simplified)



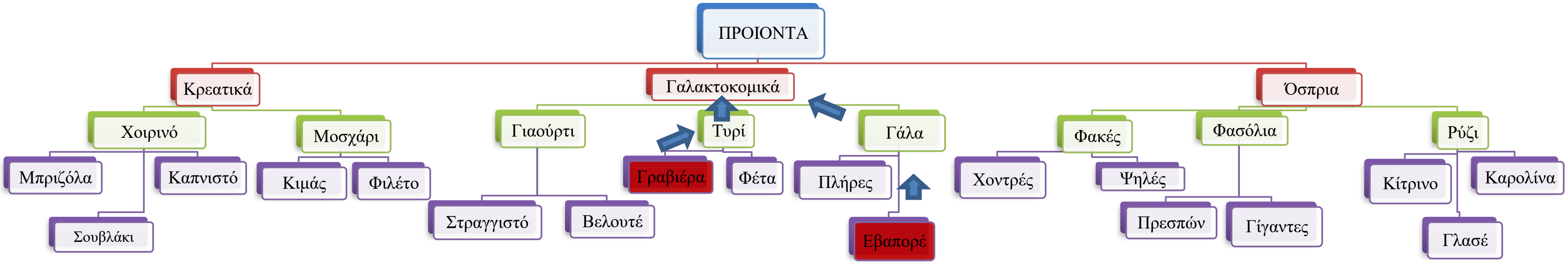
- Assume a,b are leaves and lca is their *lowest common ancestor* in the hierarchy
- Examples
  - lca(”Γραβιέρα”,”Φέτα”) = “Τυρί”
  - lca(”Γραβιέρα”, ”Εβαπορέ”) = “Γαλακτοκομικά”
  - lca(“Γραβιέρα”, ”Γίγαντες”) = “ΠΡΟΙΟΝΤΑ”

Using Hierarchies (simplified)



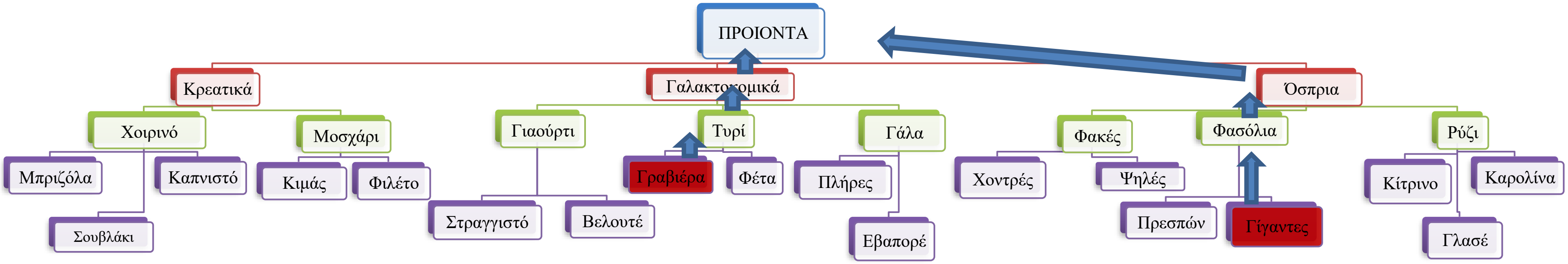
- Assume a,b are leaves and |p_lca|= the length of the path towards their *lowest common ancestor* (lca)
  - Define $d(a,b) = \frac{|p\_lca|}{tree\_height}$ if a≠b, 0 otherwise
- Example
  - d(”Γραβιέρα”,”Φέτα”) = $\frac{1}{3}$ (lca = “Τυρί”)

Using Hierarchies (simplified)



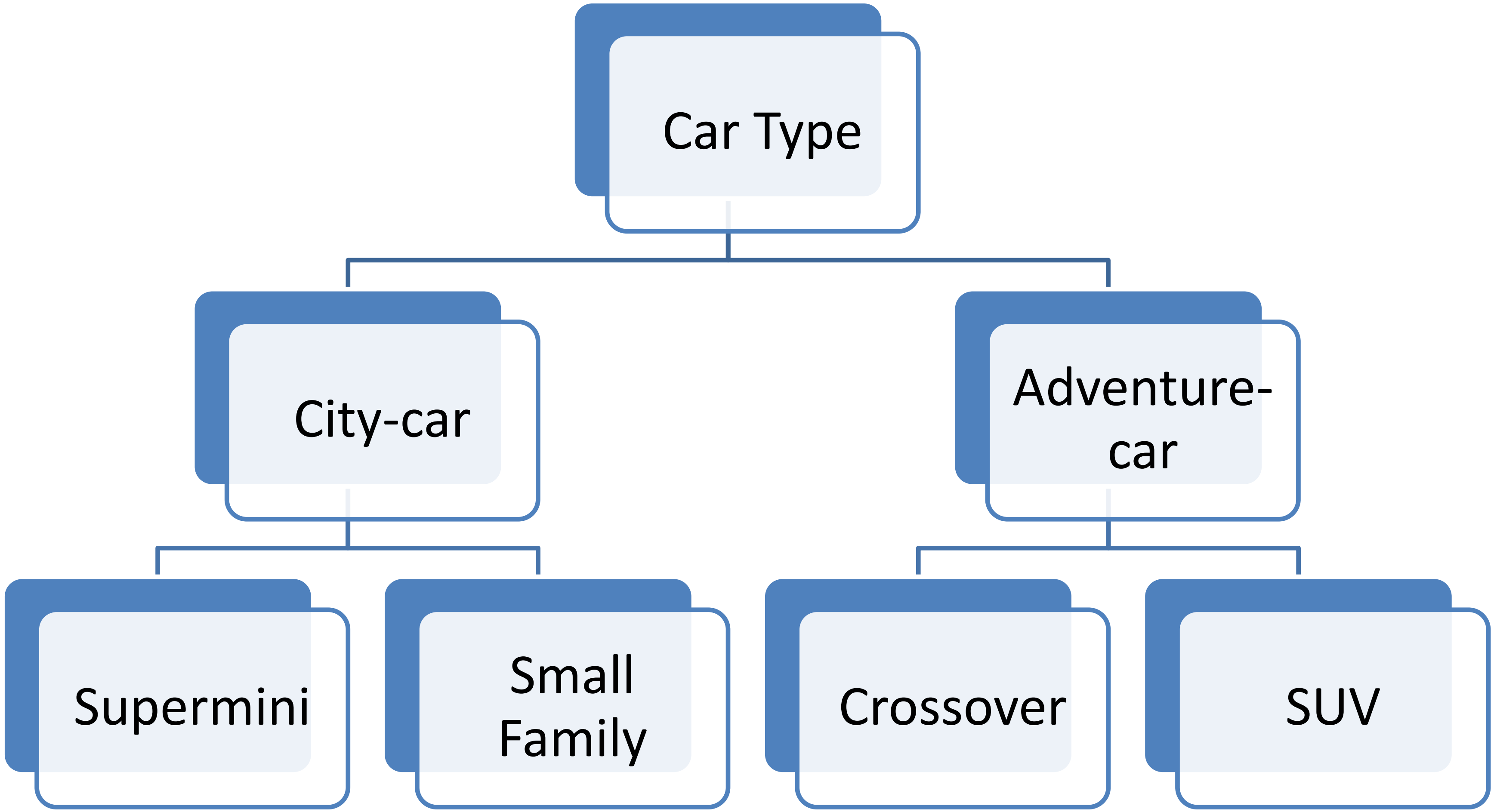
- Assume a,b are leaves and |p_lca|= the length of the path towards their *lowest common ancestor* (lca)
  - Define $d(a,b) = \dfrac{|p\_lca|}{tree\_height}$ if a≠b, 0 otherwise
- Example
  - d(”Γραβιέρα”, ”Εβαπορέ”) = $\frac{2}{3}$ (lca = “Γαλακτοκομικά”)

Using Hierarchies (simplified)

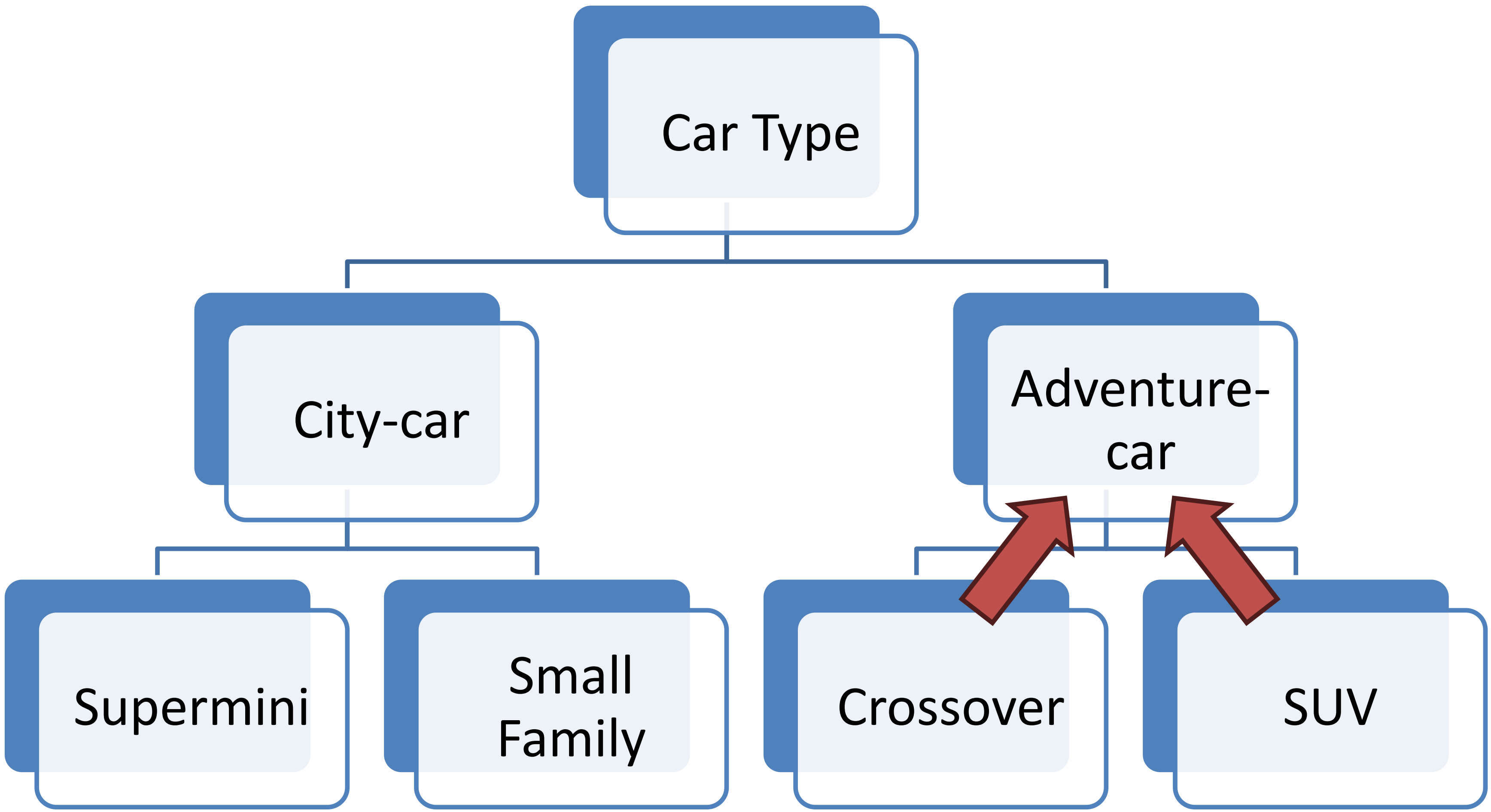


- Assume a,b are leaves and |p_lca|= the length of the path towards their *lowest common ancestor* (lca)
  - Define $d(a,b) = \frac{|p\_lca|}{tree\_height}$ if a≠b, 0 otherwise
- Example
  - d(“Γραβιέρα”, ”Γίγαντες”) $= \frac{3}{3}$ (lca = “ΠΡΟΙΟΝΤΑ”)

d(Crossover, SUV)=?



$$d(\text{Crossover}, \text{SUV}) = \frac{1}{2}$$



COMBINING EVIDENCE

Combining similarities from difference processes/sources

- Assume we have two separate processes for computing similarities between users

- Process 1: assesses demographic data from the user database (gender, age, marital status, etc.)
  - Reports similarity score s_1 based on demographic data
- Process 2: considers their interaction with our systems (e.g. purchases, logins, etc.)
  - Reports similarity score s_2 based on user activity

Taking weighted averages

- We already saw this computation

$$\text{sim} = \frac{w_1 * s_1 + w_2 * s_2}{w_1 + w_2}$$

- Can be fine-tuned to our preferences or trust on these datasets
  - E.g. if we believe that activity data is more reliable or important, use $w_2 > w_1$

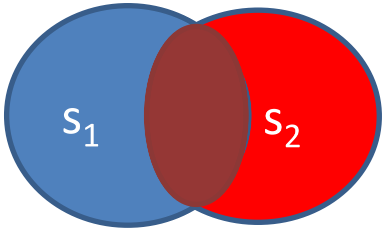
Treating scores as evidence

- One problem with averaging is that low scores from one of the two processes (e.g. due to wrong/missing data) will lower the overall calculation
  - Example: s_1 = 0.7, s_2=0.2, Average(s_1, s_2)=0.45
- Possible solution: take maximum score
  - Take: Max(s_1, s_2)=0.70
- Another idea is to treat each score as independent evidence that each boosts our confidence on the similarity between the users

Treating similarities as probabilities

- Assuming independence, combine scores in a probabilistic manner

$$\mathbf{sim(s_1,s_2) = s_1 + s_2 - s_1 * s_2}$$



- In our example
  - sim(0.7,0.2)=0.7+0.2-0.14=0.76

- Notice that $\mathbf{sim(s_1,s_2) \geq max(s_1,s_2)}$

Using additional sources

- This calculation can be extended in case we have more sources suggesting similarity for the customers
  - e.g. based on customer surveys s_3= 0.8
- Combine scores in a probabilistic manner

$$\mathbf{sim(s_1, s_2, s_3) = sim(s_1, s_2) + s_3 - sim(s_1, s_2) * s_3}$$

- In our running example
  - sim(0.7,0.2,0.8) = 0.76 + 0.8 – 0.76*0.8 = 0.952
  - Compare with average (0.7,0.2,0.8) = 0.56
  - Compare with max (0.7,0.2,0.8) = 0.8
  - Compare with min (0.7,0.2,0.8) = 0.2

WORKING WITH SETS

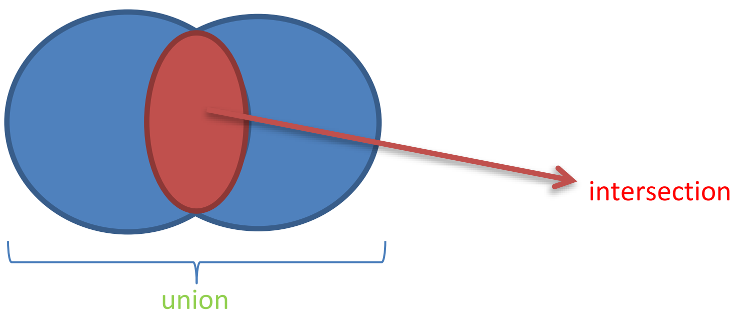
How do we compare <u>sets</u>?

- UserA= {milk, bread, coffee}
- UserB= {milk, bread, donut}
- UserC= {milk, bread, soda, potatoes}

- Straightforward idea: look at their intersection
  - Intersection(UserA,UserB) = {milk,bread}
  - Intersection(UserA,UserC) = {milk,bread}
- Intersection not enough!
  - Need to look at their differences too

Set similarity: Jaccard Index

- Jaccard(S1,S2) = the ratio of the sizes of the intersection and union of S1 and S2
  - Jaccard(S1,S2) = $|S1 \cap S2| / |S1 \cup S2|$



- Note that $|S1 \cap S2| \leq |S1 \cup S2|$
- Thus, $0 \leq$ Jaccard(S1,S2) ≤ 1

Jaccard Index Examples

- Recall: Jaccard(S1,S2) = $|S1 \cap S2| / |S1 \cup S2|$
- Jaccard({potatoes, lettuce}, {potatoes, tomatoes}) = $\frac{1}{3}$
- Jaccard({potatoes, lettuce, cucumbers}, {potatoes, tomatoes, ketchup}) = $\frac{1}{5}$
- Jaccard({potatoes, lettuce}, {potatoes, lettuce, tomatoes}) = $\frac{2}{3}$
- Jaccard({lettuce}, {milk, soda}) = 0
- Jaccard({soda, milk}, {milk, soda}) = 1

Toy exercise
(python jupyter notebook in e-class)

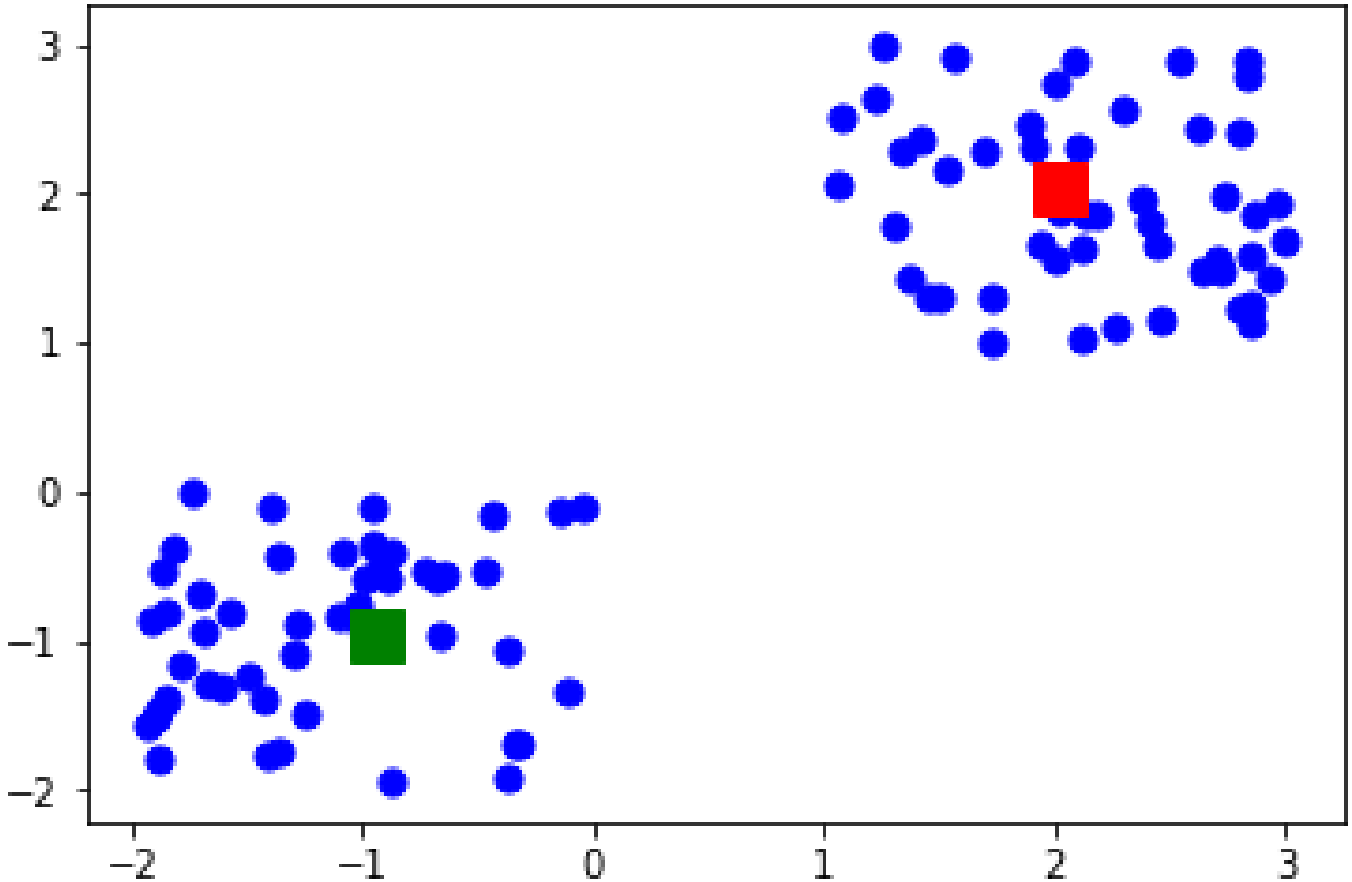
- Assume the following 5 customers with their purchases

```
user1 : ['milk', 'bread', 'coffee’]
user2 : ['milk', 'bread', 'cola’]
user3 : ['cereal', 'milk', 'donut’]
user4 : ['donut', 'cream', 'cola’]
user5 : ['cola', 'milk', 'cereal’, ’tea’]
```

- Can you group these customers into two clusters?

Clustering

- ❑ Separate data into disjoint groups such that:
  - ❑ Increased similarity among members of the same group (cluster cohesion)
  - ❑ Members of different groups are dissimilar



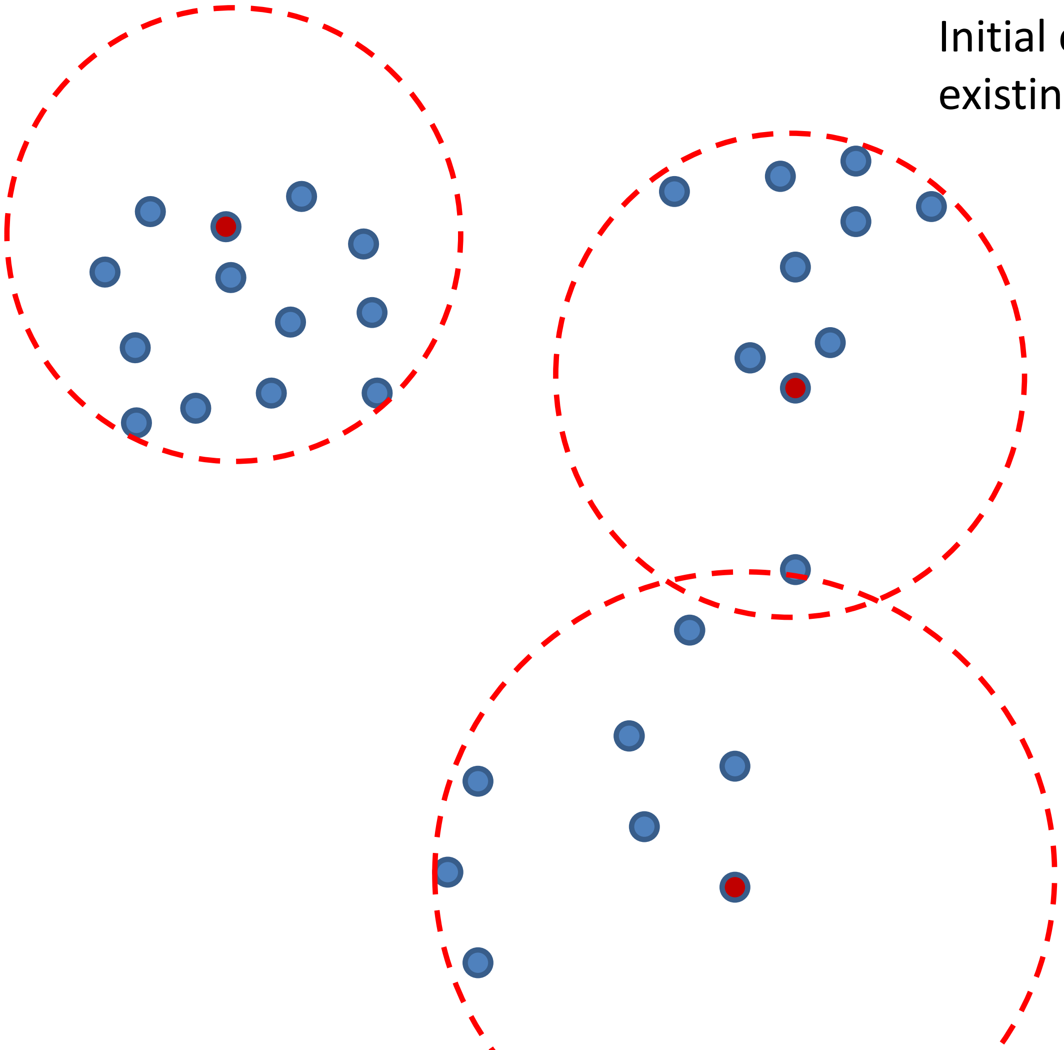
The famous k-Means algorithm

- Assume n points in the Euclidian space and a user-defined value of k=#clusters

1. Pick k points (centroids), one per cluster
2. Assign remaining points to closest centroid
3. In each cluster update location of its centroid
4. Reassign points, if necessary
5. Repeat steps 3-4 until clusters stabilize

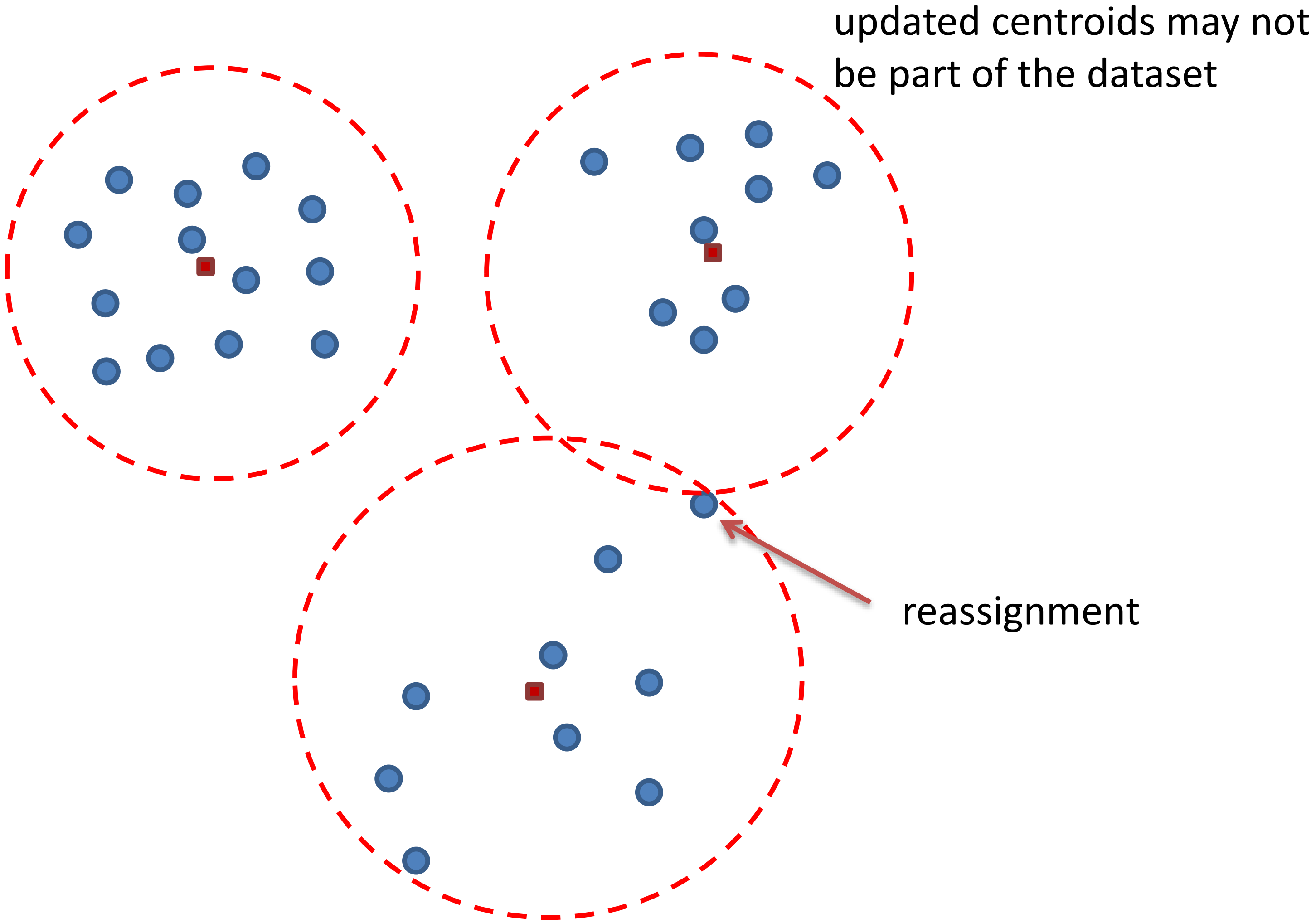
- k-Means seeks to minimize the sum of squared distances (thus the variance of the distances) from the centroids
  - the algorithm always converges to some (local) minimum solution

Example for k=3



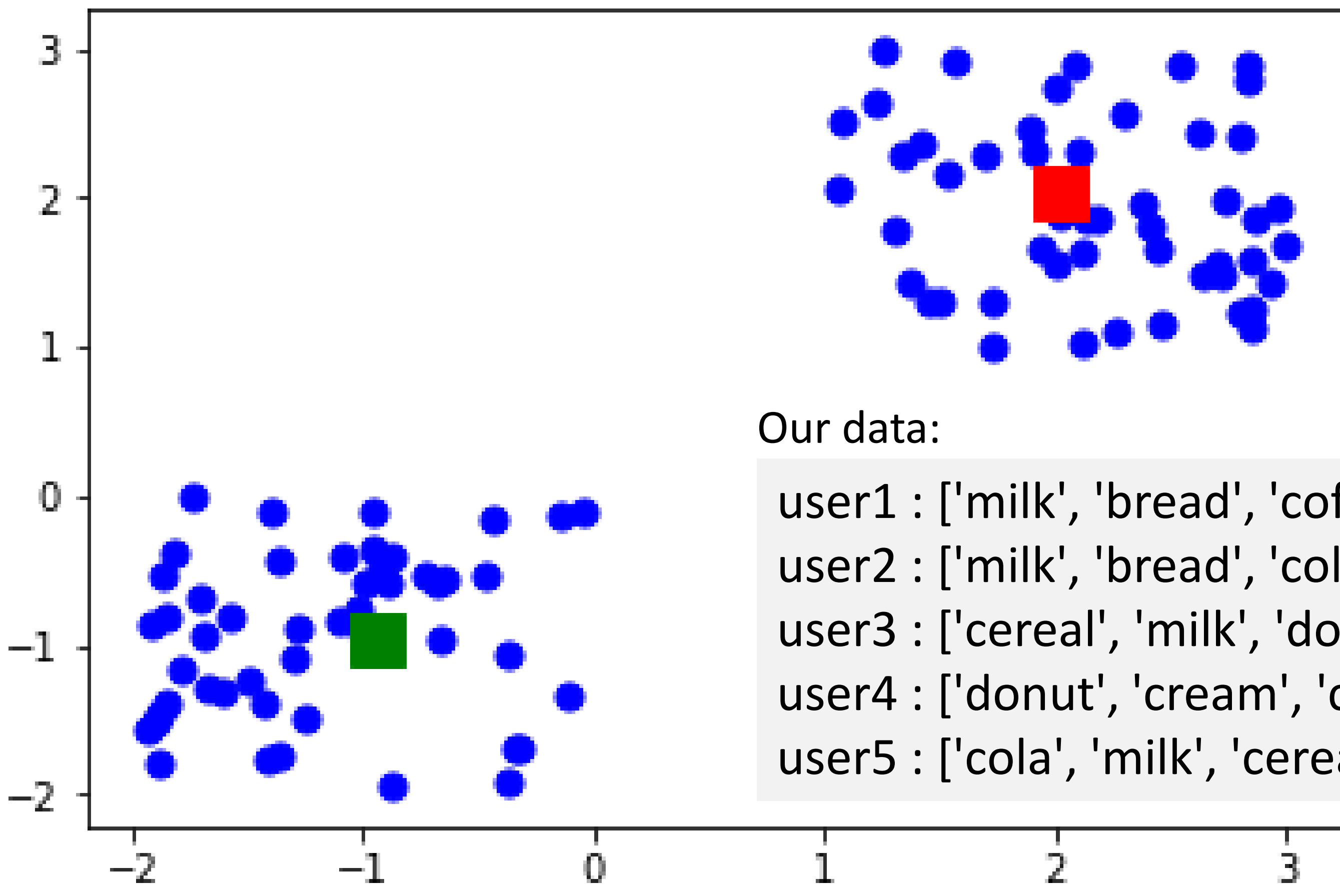
Initial centroids are existing dataset points

New centroids + reassignment



Wait!

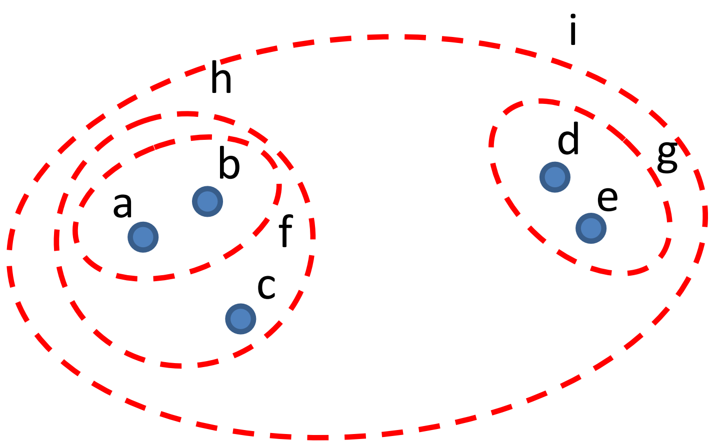
- ❑ Our dataset is not points in a Euclidian space
  - ❑ There is no obvious way to compute a “centroid”

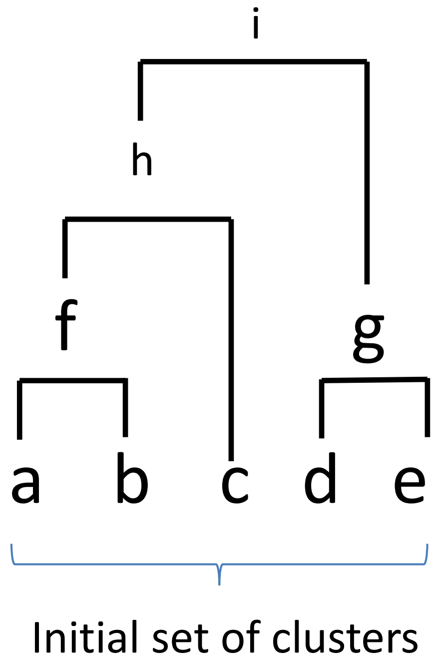


Our data:

```
user1 : ['milk', 'bread', 'coffee’]
user2 : ['milk', 'bread', 'cola’]
user3 : ['cereal', 'milk', 'donut’]
user4 : ['donut', 'cream', 'cola’]
user5 : ['cola', 'milk', 'cereal’, ’tea’]
```

Hierarchical Clustering to the rescue





Executive decision

- Purchases are modelled as sets of items
  - Use Jaccard for computing customer pair-wise similarity

```
user1 : ['milk', 'bread', 'coffee’]
user2 : ['milk', 'bread', 'cola’]
user3 : ['cereal', 'milk', 'donut’]
user4 : ['donut', 'cream', 'cola’]
user5 : ['cola', 'milk', 'cereal’, ’tea’]
```

Jaccard_sim = 2/4 = 50%

Jaccard_sim = 1/6 = 16%

Jaccard Similarity

- All-pair similarity computation

user1 : ['milk', 'bread', 'coffee’]
user2 : ['milk', 'bread', 'cola’]
user3 : ['cereal', 'milk', 'donut’]
user4 : ['donut', 'cream', 'cola’]
user5 : ['cola', 'milk', 'cereal’, ‘tea’]

Jaccard_sim of user1 , user2 is 0.5
Jaccard_sim of user1 , user3 is 0.2
Jaccard_sim of user1 , user4 is 0.0
Jaccard_sim of user1 , user5 is 0.16
Jaccard_sim of user2 , user3 is 0.2
Jaccard_sim of user2 , user4 is 0.2
Jaccard_sim of user2 , user5 is 0.4
Jaccard_sim of user3 , user4 is 0.2
Jaccard_sim of user3 , user5 is 0.4
Jaccard_sim of user4 , user5 is 0.16

Hierarchical Clustering

- Merge most similar pair to form a new cluster

user1 : ['milk', 'bread', 'coffee’]
user2 : ['milk', 'bread', 'cola’]
user3 : ['cereal', 'milk', 'donut’]
user4 : ['donut', 'cream', 'cola’]
user5 : ['cola', 'milk', 'cereal’, ‘tea’]

Jaccard_sim of user1 , user2 is 0.5
Jaccard_sim of user1 , user3 is 0.2
Jaccard_sim of user1 , user4 is 0.0
Jaccard_sim of user1 , user5 is 0.16
Jaccard_sim of user2 , user3 is 0.2
Jaccard_sim of user2 , user4 is 0.2
Jaccard_sim of user2 , user5 is 0.4
Jaccard_sim of user3 , user4 is 0.2
Jaccard_sim of user3 , user5 is 0.4
Jaccard_sim of user4 , user5 is 0.16

New state

- Merge best pair (user1+user2) to form a new cluster
  - Represent cluster of customers as their union (not ideal, other options exist)

~~user1 : ['milk', 'bread', 'coffee’]~~
~~user2 : ['milk', 'bread', 'cola’]~~
user3 : ['cereal', 'milk', 'donut’]
user4 : ['donut', 'cream', 'cola’]
user5 : ['cola', 'milk', 'cereal’, ‘tea’]
user1+user2 : ['bread', 'cola', 'milk', 'coffee’]

Next step
(most similar pair: user3, user5)

user3 : ['cereal', 'milk', 'donut’]
user4 : ['donut', 'cream', 'cola’]
user5 : ['cola', 'milk', 'cereal’, ‘tea’]
user1+user2 :['bread', 'cola', 'milk', 'coffee’]

user4 : ['donut', 'cream', 'cola’]
user1+user2 :['bread', 'cola', 'milk', 'coffee’]
user3+user5 : ['cereal', 'donut', 'milk', 'cola', 'tea’]

Final step
(most similar pair: user4, user3+user5)

user4 : ['donut', 'cream', 'cola’]
user1+user2 :['bread', 'cola', 'milk', 'coffee’]
user3+user5 : ['cereal', 'donut', 'milk', 'cola', 'tea’]

user1+user2 :['bread', 'cola', 'milk', 'coffee’]
user4+user3+user5 : {'donut', 'cereal', 'milk', 'cream', 'cola', 'tea'}

Cluster 1

user1 : ['milk', 'bread', 'coffee’]
user2 : ['milk', 'bread', 'cola’]

Cluster 2

user3 : ['cereal', 'milk', 'donut’]
user4 : ['donut', 'cream', 'cola’]
user5 : ['cola', 'milk', 'cereal’]

Notes

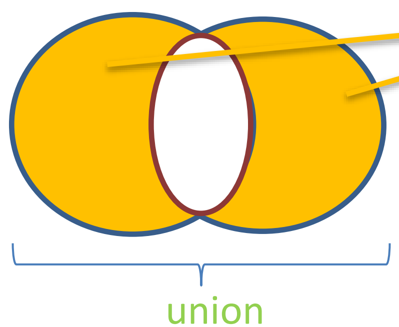
- In this toy example we performed **Hierarchical Clustering** up to 2 clusters without checking the quality of the intermediate clusters
  - Sometimes it is better to stop sooner that later
- To simplify the code, we used as a representative (clustoid) of a cluster the UNION of its members
  - Can you think of examples where this is a bad choice?

Cluster 1	**Cluster 2**
user1 : ['milk', 'bread', 'coffee’] user2 : ['milk', 'bread', 'cola’]	user3 : ['cereal', 'milk', 'donut’] user4 : ['donut', 'cream', 'cola’] user5 : ['cola', 'milk', 'cereal’]

Jaccard Distance between sets

- Can be defined as the complement of their Jaccard similarity
  - $d_{jacc}(S1,S2) = 1 - \frac{|S1 \cap S2|}{|S1 \cup S2|} = \frac{|S1 \cup S2| - |S1 \cap S2|}{|S1 \cup S2|}$

Symmetric difference



How about bags?

Bags are “sets” with repetition of elements allowed

Jaccard can be extended to work with bags

Intersection(S1,S2): count an element n times in the intersection, where n is the **minimum** of the number of times the element appears in S1 and S2	**Union(S1,S2)**: count the element the **sum** of the number of times it appears in S1, S2

Example

- S1 = {a,a,a,b}, S2 = {a,a,b,b,c}
- Then, intersection is {a,a,b} and union {a,a,a,a,a,b,b,b,c}
- Bag-similarity is thus, 3/9 = 1/3

- Note, bag similarity is between 0 and ½ (why?)

Alternative bag similarity

- Count an element n times in the intersection, where n is the minimum of the number of times the element appears in S1 and S2
- In the union, count the element the max of the number of times it appears in S1, S2

Example (alt)

- S1 = {a,a,a,b}, S2 = {a,a,b,b,c}
- Then, intersection is {a,a,b} and union {a,a,a,b,b,c}
- Bag-similarity of S1, S2 is thus, 3/6 = 50%

- Note, alternative bag similarity is between 0 and 1 (why?)

Bag Similarity Example

- Movies ratings dataset
  - John: Star_Wars_I:3/5, Avatar: 4/5, Aliens: 2/5
  - Mary: Star_Wars_I: 2/5, Avatar: 5/5, ET: 4/5
  - Nick: Star_Wars_I: 4/5, Aliens: 2/5, ET: 1/5

- Who is the Nearest Neighbor of John?
- Note: if treated as sets
  - Jaccard(John, Mary) = Jaccard(John, Nick) = 2/4 = 50%
  - Let us consider their bag similarity instead!

Bag Similarity Example

- Convert to bags:
  - John: {Star_Wars_I, Star_Wars_I, Star_Wars_I, Avatar, Avatar, Avatar, Avatar, Aliens, Aliens}
  - Mary: {Star_Wars_I, Star_Wars_I, Avatar, Avatar, Avatar, Avatar, Avatar, ET, ET, ET, ET}
  - Nick: {Star_Wars_I, Star_Wars_I, Star_Wars_I, Star_Wars_I, Aliens, Aliens, ET}
- Bag_similarity_alt(John,Mary) = (2+4)/(3+5+4+2) = 6/14 = 42.9%
- Bag_similarity_alt(John,Nick) = (3+2)/(4+4+2+1) = 5/11 = 45.5%

WORKING WITH VECTORS

Basket data example

- Three distinct products:
  - potato (p), lettuce (l), tomato (t)
- Three users with the following purchases
  - John: 2 potatoes, 1 lettuce
  - Kostas: 1 tomato
  - Mary: 10 potatoes, 6 lettuces

Vector Model

$<\#p,\#l,\#t>$

$\vec{J} = <2,1,0>$

$\vec{K} = <0,0,1>$

$\vec{M} = <10,6,0>$

Definition of Euclidean Distance

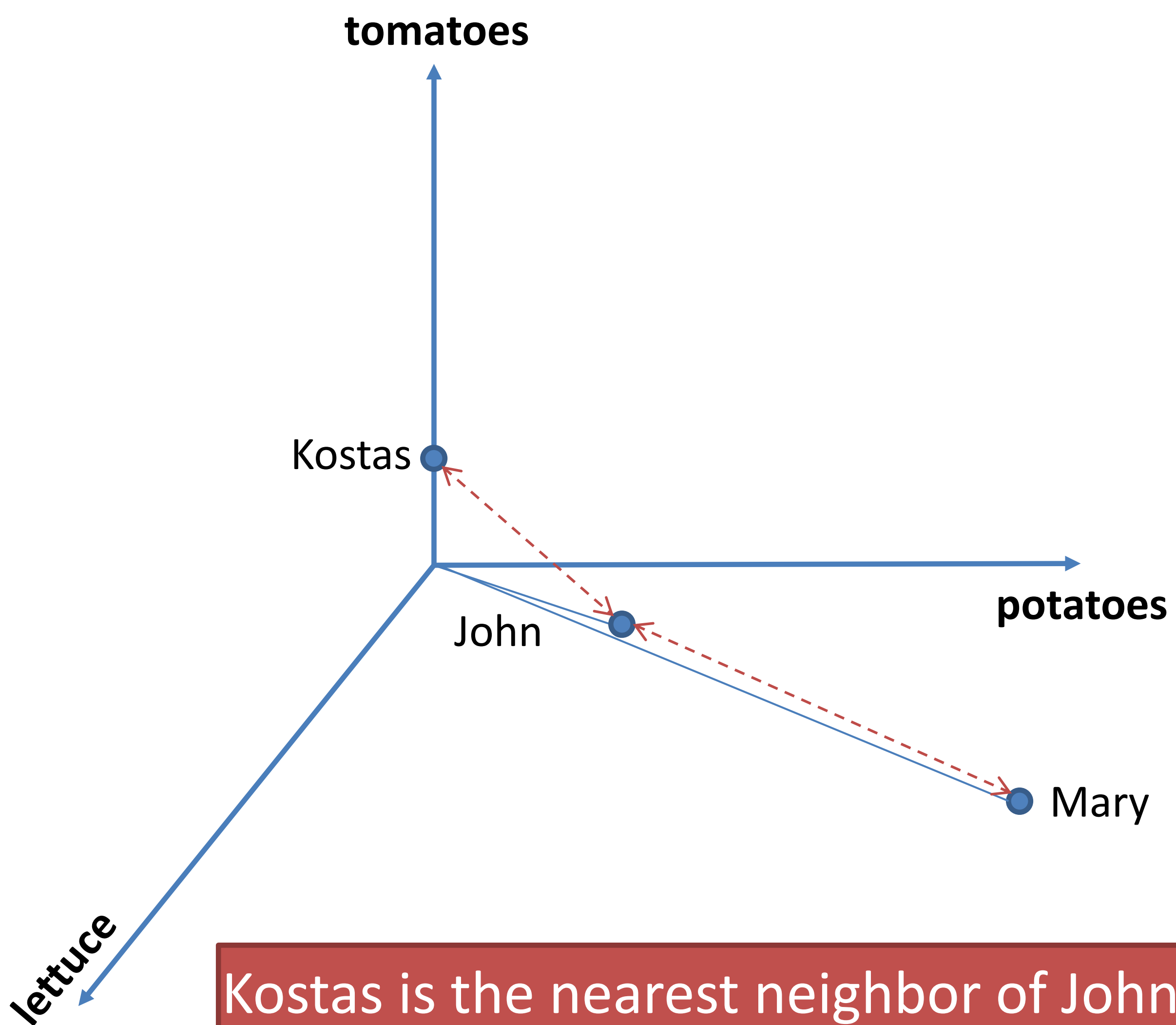
- $\vec{x}$ = <2,1,0,5>
- $\vec{y}$ = <5,6,1,10>

- Recall that:

$$d(\vec{x}, \vec{y}) = \sqrt{(2-5)^2 + (1-6)^2 + (0-1)^2 + (5-10)^2}$$

$$= \sqrt{9 + 25 + 1 + 25} = \sqrt{60} = 7.75$$

Euclidean Distance NN Calculations



Vector Model

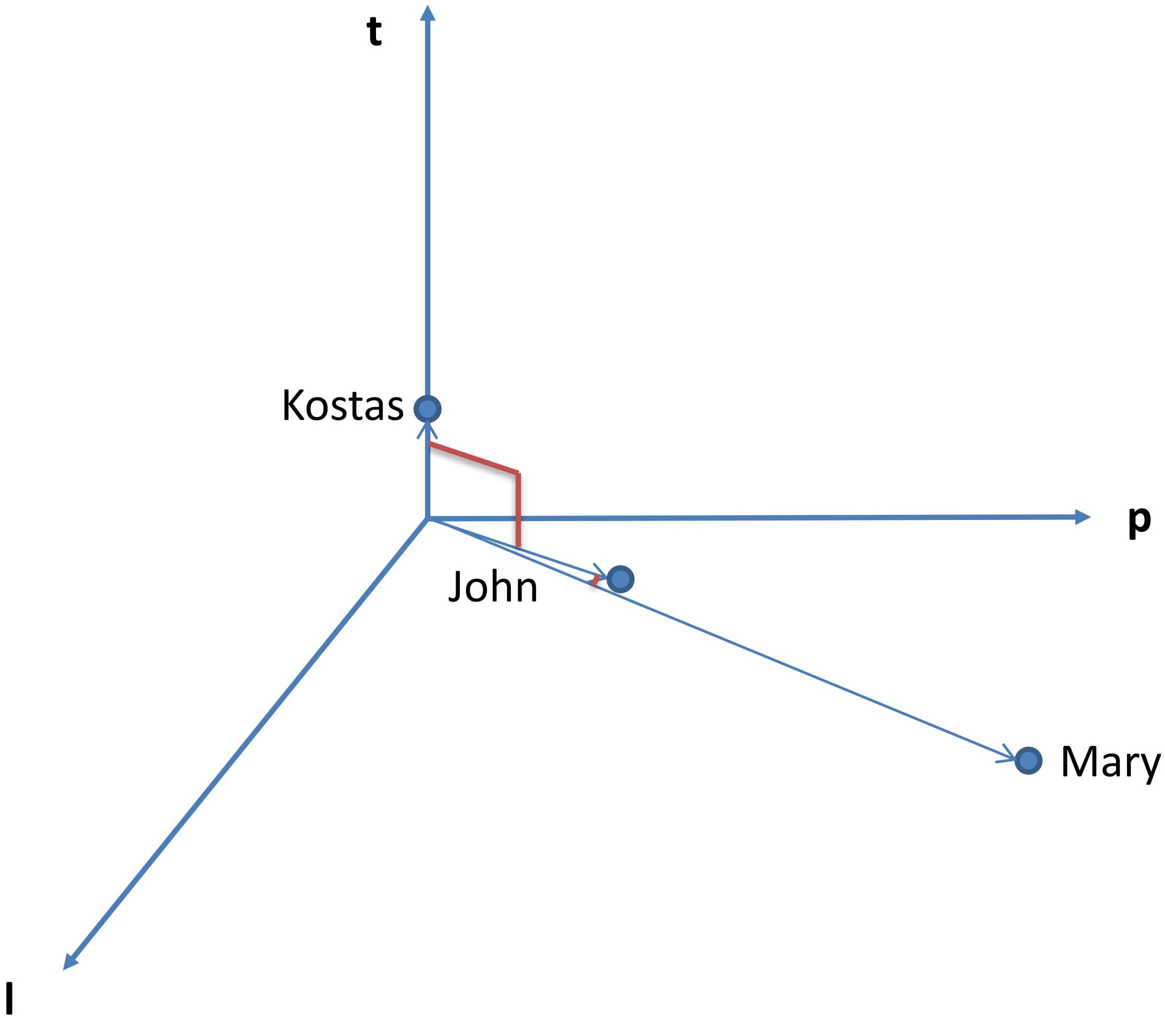
$\vec{J} = <2,1,0>$

$\vec{K} = <0,0,1>$

$\vec{M} = <10,6,0>$

Kostas is the nearest neighbor of John!!!

Angle Calculations: favor direction over length (norm)



Vector Model

$\vec{J} = <2,1,0>$

$\vec{K} = <0,0,1>$

$\vec{M} = <10,6,0>$

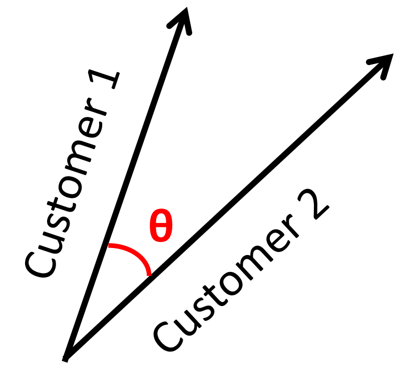
$\theta(\vec{J}, \vec{K}) = 90^o$

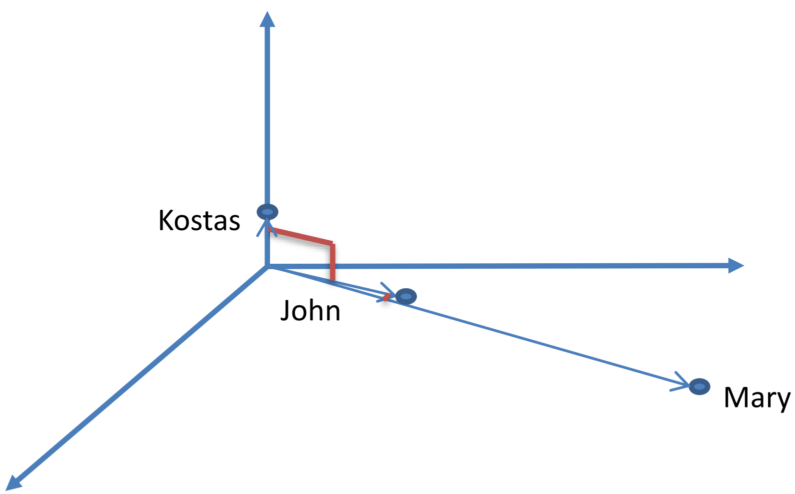
$\theta(\vec{J}, \vec{M}) = 4.3^o$

When to use Cosine?

Cosine Similarity

- $sim(\vec{x},\vec{y}) = \cos(\theta(\vec{x},\vec{y})) \in [-1..+1]$
  - Used in collaborative filtering
  - Popular in document matching





$\cos(\theta(\vec{J}, \vec{K})) = \cos(90^o) = 0$

$\cos(\theta(\vec{J}, \vec{M})) = \cos(4.3^o) = 0.997$

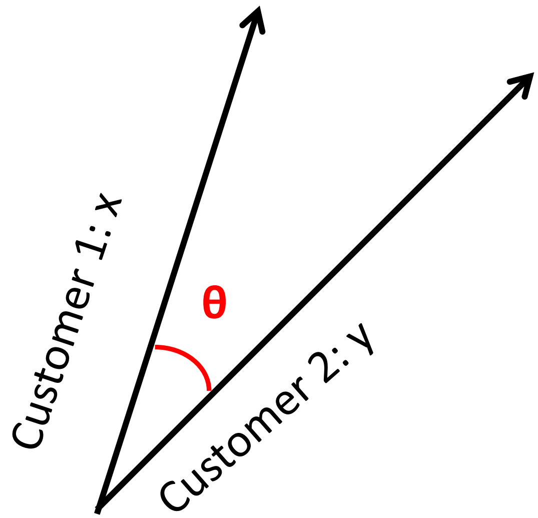
Dot (inner) product between two vectors

- $\vec{x} \,.\, \vec{y} = \Sigma(x_k * y_k)$
- Example:

$$\vec{x} = (1,3,0,5)$$
$$\vec{y} = (1,0,1,6)$$

- Then:

$$\vec{x} \,.\, \vec{y} = 1*1+\boxed{3*0+0*1}+5*6=31$$
$$= |\vec{x}|*|\vec{y}|*\cos(\theta(\vec{x}, \vec{y}))$$



From dot to cosine

$\vec{x} = (1,3,0,5)$

$\vec{y} = (1,0,1,6)$

- $\cos(\theta(\vec{x}, \vec{y})) = \frac{\vec{x} \cdot \vec{y}}{|\vec{x}| * |\vec{y}|}$
- In this example
- $|\vec{x}| = \sqrt{1^2 + 3^2 + 0^2 + 5^2} = \sqrt{35}$
- $|\vec{y}| = \sqrt{1^2 + 0^2 + 1^2 + 6^2} = \sqrt{38}$

- $\cos(\theta(\vec{x}, \vec{y})) = \frac{31}{\sqrt{35} * \sqrt{38}} = 0.85$

Dot product with unit vector

- $\vec{x} \cdot \vec{y} = \Sigma(x_k * y_k)$
- Example for unit vector $\vec{y}$:

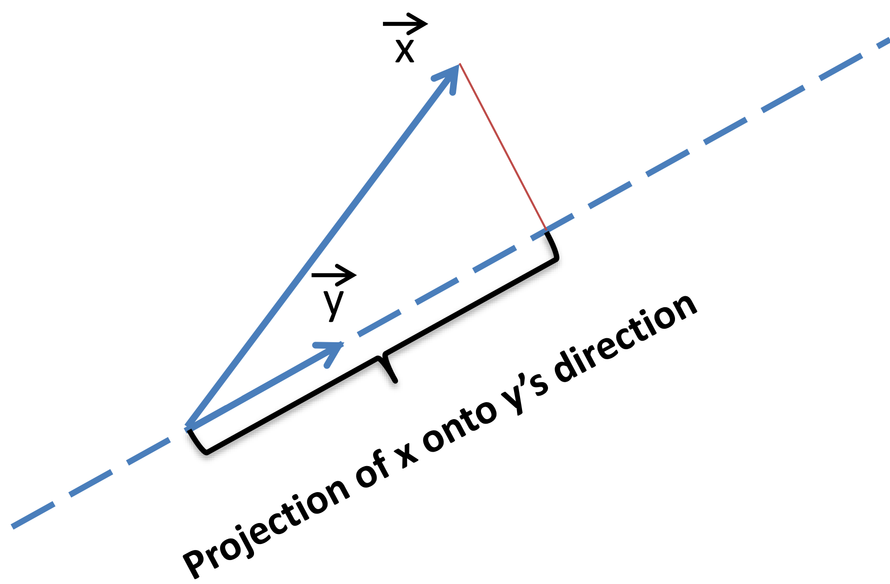
$\vec{x} = (1,3,0,5)$

$\vec{y} = (1/2,1/2,1/2,1/2)$

- Notice that $|\vec{y}|=1$
- Then:

$\vec{x} \cdot \vec{y} = 1/2+3/2+5/2=9/2=4.5$

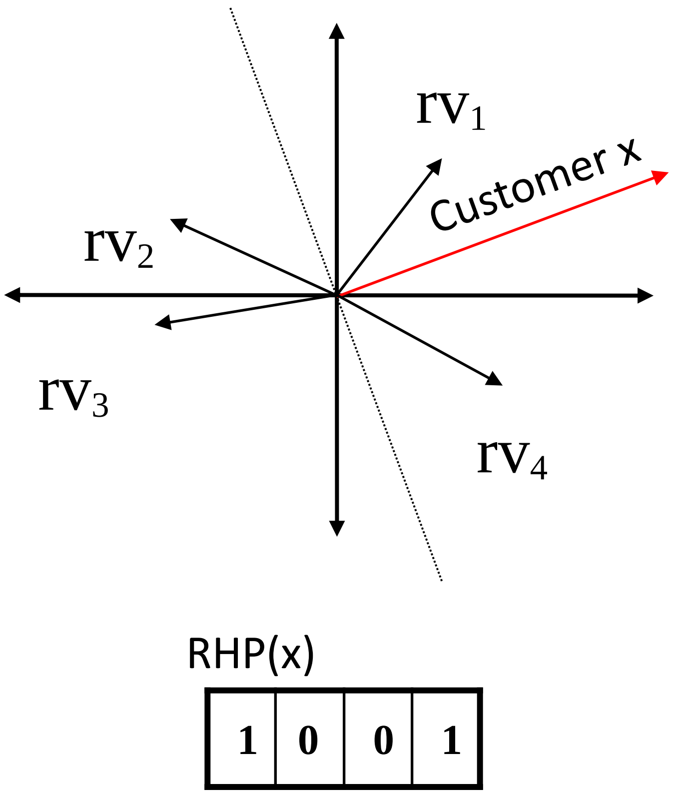
$= |\vec{x}| * 1 * \cos(\theta(\vec{x}, \vec{y}))$



Random Hyperplane Projection

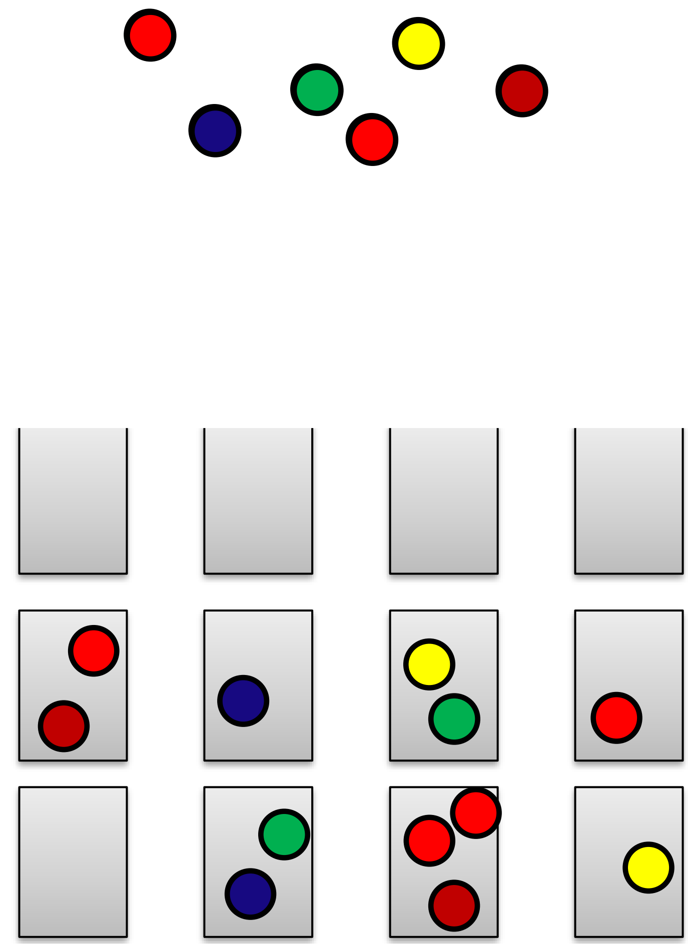
(Mining Massive Data Sets, Sec. 3.7.2)

- uses *n* d-dimensional random vectors (rv_i)
- Generates for each input vector a bitmap of size n as follows:
  - Sets $bit_i=1$ if dot product of input vector with *i-th* random vector is positive
  - Sets $bit_i=0$ if dot product of input vector with *i-th* random vector is negative



Locality Sensitive Hashing (LSH)

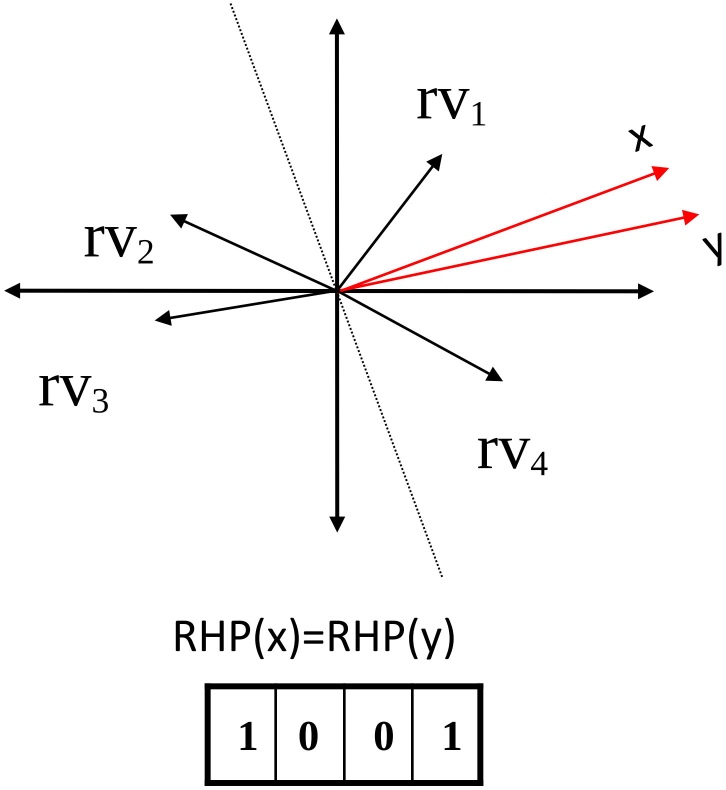
- Assign items to **buckets** using a hash function **h(x)**
  - E.g. h(●)=0110 in binary
  - Details of function h() depend on the preferred similarity metric:
    - Similar objects are hashed to the same bucket with high probability
    - Dissimilar objects are hashed to the same bucket with very small probability
- Repeat several times



Buckets (1-4)

Is RHP a locality-sensitive hashing scheme?

- Assume vectors for customers x and y point (approximately) towards the same direction
  - This means their cosine is close to 1
- We expect that with high probability the RHP values will be identical
- Use RHP encodings as "bucket ids"
  - Similar customers are hashed to the same bucket (with high probability)



Hamming Distance

- The Hamming distance between two equal-length strings of symbols is the number of positions at which the corresponding symbols are different (Wikipedia)
  - D_h('00110101',
    '10110110') = 3
  - D_h ('abc','acc') = 1

Approximate Similarity Computation via Hamming Distance of RHP bitmaps

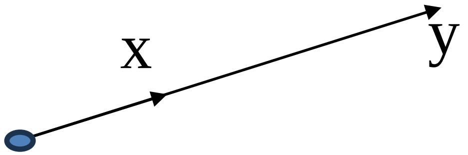
$RHP_{(x)}$:

0	0	0	1	1	1

n bits

$RHP_{(y)}$:

0	0	0	1	1	1



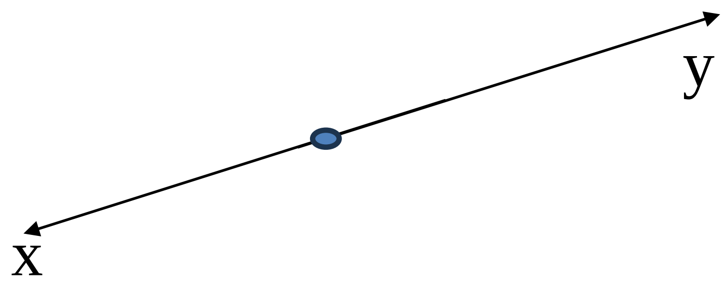
- Vectors are collinear ($\theta(\vec{x},\vec{y})=0$, cosine similarity = 1)

Approximate Similarity Computation via Hamming Distance of RHP bitmaps

$RHP_{(x)}$: | 0 | 0 | 0 | 1 | 1 | 1 |

n bits

$RHP_{(y)}$: | 1 | 1 | 1 | 0 | 0 | 0 |



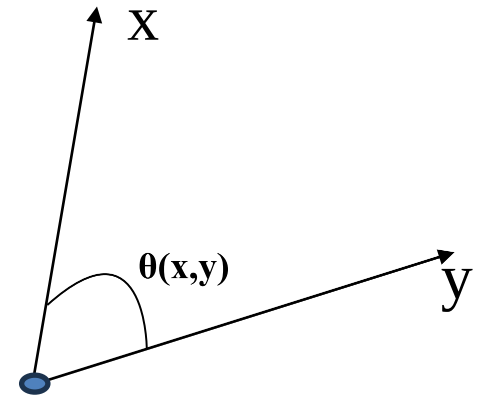
- Vectors are opposite ($\theta(\vec{x},\vec{y})=\pi$, cosine similarity = 0)

Approximate Similarity Computation via Hamming Distance of RHP bitmaps

$RHP_{(x)}$:	**1**	0	**1**	1	1	1

n bits

$RHP_{(y)}$:	**0**	0	**0**	1	1	1

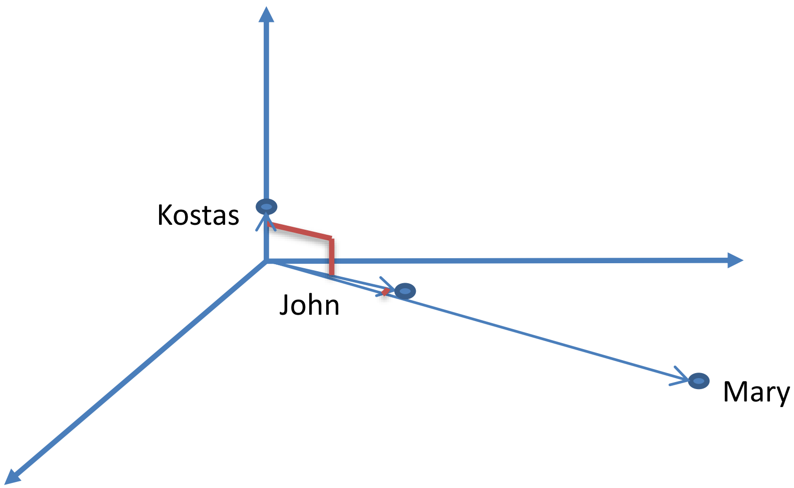


$\hat{\theta}(RHP_{(x)},RHP_{(y)})=2/6*\pi=\pi/3$

Estimate $\hat{\theta}(x,y)=D_h(RHP_{(x)},RHP_{(y)})*\pi/n$

- Also works for the *Pearson correlation*
  - *Cor(x, y) = Cos(x-$\bar{x}$, y-$\bar{y}$)*

RHP Example



J = <2,1,0>

K = <0,0,1>

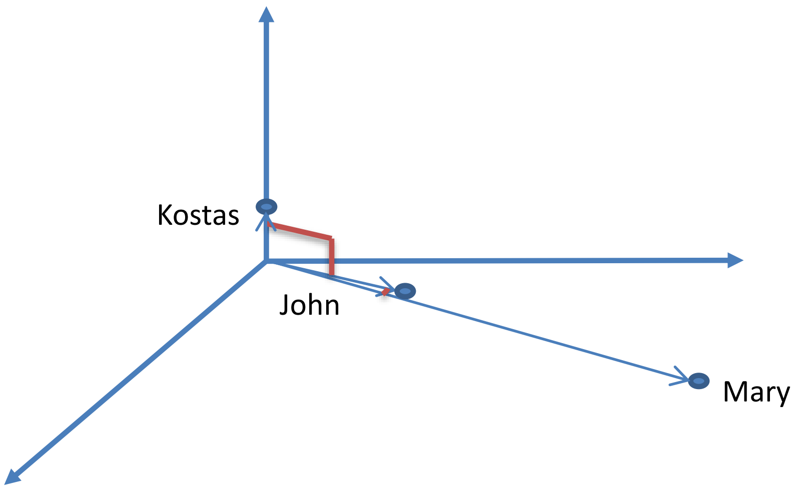
M = <10,6,0>

rv1=<3,1,6>

rv2=<-5,3,2>

- Calculations for John:
- <2,1,0>.<3,1,6> = 2*3+1*1+0*6=+7 $\rightarrow$ bit =1
- <2,1,0>.<-5,3,2> =-10+3=-7 $\rightarrow$ bit = 0
- Thus, **RHP(John) = 10**

RHP Example



J = <2,1,0>

K = <0,0,1>

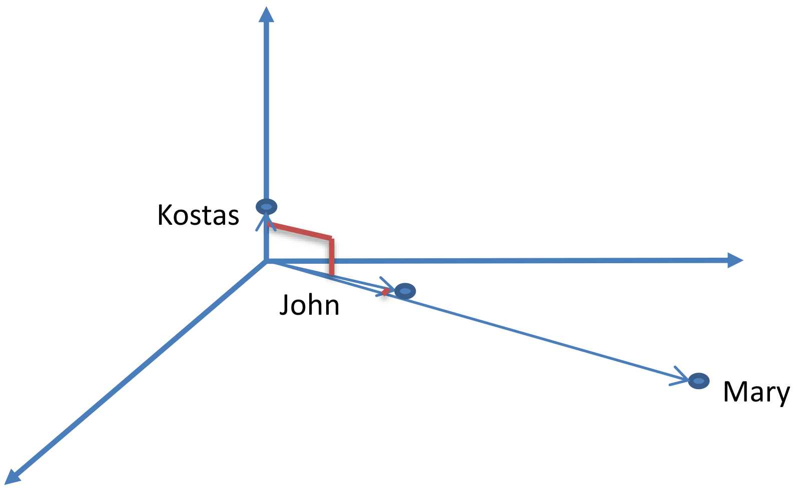
M = <10,6,0>

rv1=<3,1,6>

rv2=<-5,3,2>

- Calculations for Kostas:
- <0,0,1>.<3,1,6> = +6 → bit =1
- <0,0,1>.<-5,3,2> =+2 → bit = 1
- Thus, **RHP(Kostas) = 11**

RHP Example



J = <2,1,0>

K = <0,0,1>

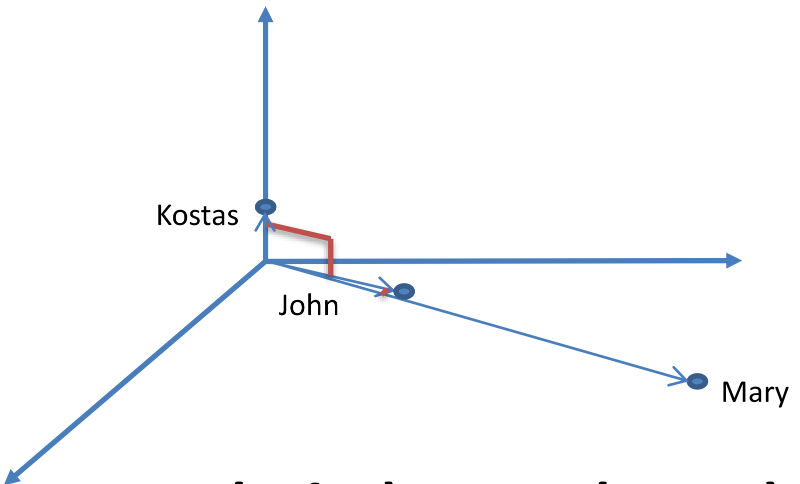
M = <10,6,0>

rv1=<3,1,6>

rv2=<-5,3,2>

- Calculations for Mary:
- <10,6,0>.<3,1,6> = +36 → bit =1
- <10,6,0>.<-5,3,2>=-32 → bit = 0
- Thus, **RHP(Mary) = 10**

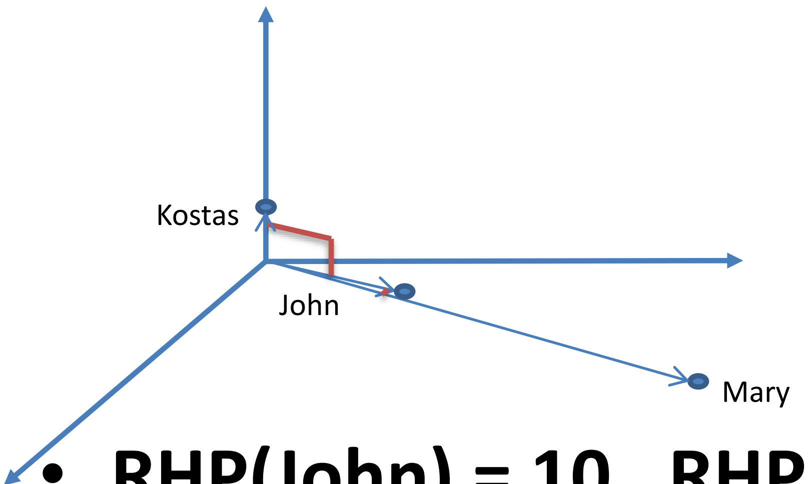
RHP Example



J = <2,1,0>
K = <0,0,1>
M = <10,6,0>

- **RHP(John) = RHP(Mary) = 10**
- Hamming distance = 0
- Estimated angle is 0
  - Thus, estimated cosine similarity = 1
  - True cosine = 0.997
- Good accuracy by using just two bits!
  - Disclaimer: I am cherry picking favorable examples here

RHP Example



J = <2,1,0>

K = <0,0,1>

M = <10,6,0>

- **RHP(John) = 10, RHP(Kostas) = 11**
- Hamming distance = 1 (out of n = 2 bits)
- Estimated angle is $\pi/2 = 90^o$
  - Thus, estimated cosine similarity = $\cos(\pi/2) = 0$
  - This is also the true cosine similarity

WORKING WITH STRINGS

String distance computations

- Why it is useful
  - String Matching
  - Spelling Checking
- Examples
  - Fix data entry errors: replace “Yiannis” with “Yannis”
  - Address matching/correction
    - Compare “Patission” , “Patision Str”, “Patission St”
  - Fraud Detection
    - Are “Kotidis123”, “Kotidis554” and “7Kotidis123” the same user?

String Edit Distance

- The edit distance between strings $x = x_1x_2..x_n$ and $y = y_1y_2..y_m$ is the <u>smallest</u> number of insertions and deletions of single characters that will convert x to y
- As an example to convert x=“abcde” to y =“acfdeg”
  1. delete b and get “a~~b~~cde”
  2. insert f after c and get “acfde”
  3. insert g after e and get “acfdeg” = y
- Thus, d_{edit}(“abcde”,”acfdeg”)=3

Longest Common Subsequence (LCS)

- The LCS of x and y is the longest common string that is constructed by deleting positions from x and y
  - For x="abcde" to y ="acfdeg"
  - LCS(x,y) = "acde"
- It holds that
  - $d_{edit}(x,y) = len(x)+len(y)-2*len(LCS(x,y))$
- In our example $d_{edit}(x,y)=5+6-2*4=3$

Levenshtein Distance

- In addition to insertions and deletions of single characters, ***Levenshtein distance*** also allows substitutions
- As an example, for x="STALL" and y="TABLE", $d_{lev}(x,y)=3$
  1. (starting with "STALL") delete S and get "TALL",
  2. substitute first L with B and get "TABL",
  3. insert E at the end and get "TABLE"
- In comparison d_{edit}("STALL", "TABLE",)=4
  - Notice that 1 substitution $\Leftrightarrow$ 1 deletion + 1 insertion

Note

- In the literature sometimes Levenshtein distance is referred as edit distance (e.g. edit distance adjusted to permit insertions, deletions as well as substitutions)

Additional Metrics for strings

- **Damerau–Levenshtein** distance further allows transportation between two successive characters
  - Corssroads → Crossroads

- **Jaro** distance only allows transportations

Time Series

- Sequence of data points indexed in time order
  - Examples: financial data, sensor data, speech, etc
  - Univariate (running examples) vs multivariate
- Can be compared with Euclidean distance (given two series of same length)
  - The i^{th} point on one time series is aligned with the i^{th} point on the other
- However, this often gives poor results
- Does not work if series have difference lengths
  - Padding?

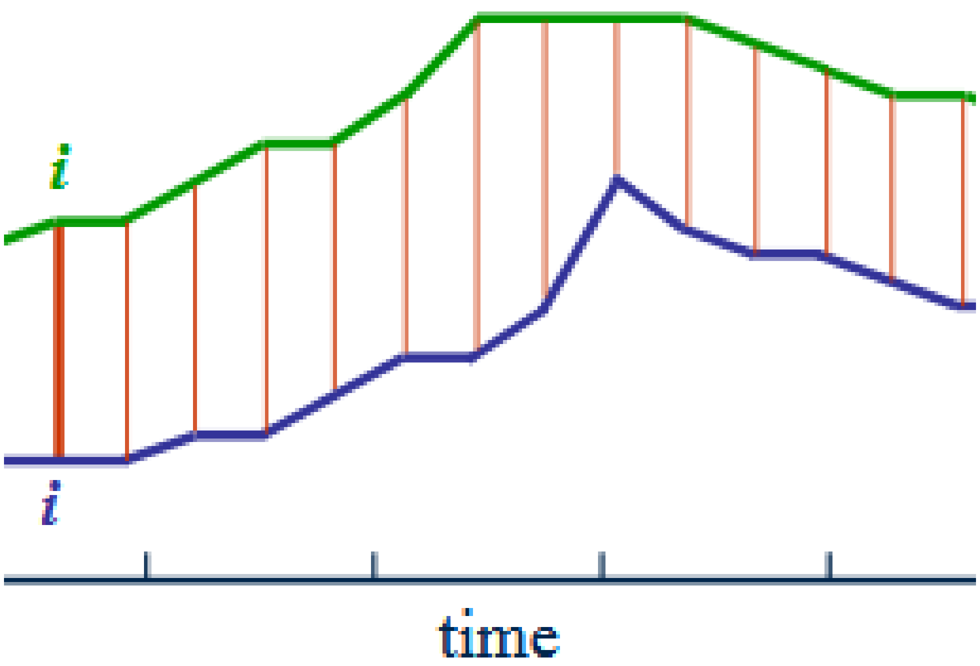


Image from
https://medium.com/datadriveninvestor/dynamic-time-warping-dtw-d51d1a1e4afc

Time Series - Euclidean Distance

- Sx = <2,1,0,1>
- Sy = <2,0,2,3>

Euclidean−distace d(Sx, Sy)=

$$\sqrt{(2-2)^2+(1-0)^2+(0-2)^2\ +(1-3)^2}$$

$$= \sqrt{0^2+1^2+2^2+2^2}$$

$$= \sqrt{0+1+4+4}$$

$$= \sqrt{9} = 3$$

Time Series - Euclidean Distance

- Sx = <2,1,0,1>
- Sy = <2,0,2,3,1,2,2,0,4>

What now?

Time Series - Euclidean Distance

- Sx = <2,1,0,1>
- Sy = <2,0,2,3,1,2,2,0,4>

Padding (convert to same length)?

- Sx = <2,1,0,1,0,0,0,0,0>
- Sy = <2,0,2,3,1,2,2,0,4>

Dynamic Time Wrapping

- DTW computes the best alignment between the two-time series
  - Works even if the input series have different lengths
  - Useful if series have different frequencies or are out of phase (e.g. lag)
- Has been shown to be superior than Euclidean distance for tasks such as time series classification
- Drawback: quadratic complexity $O(n^2)$

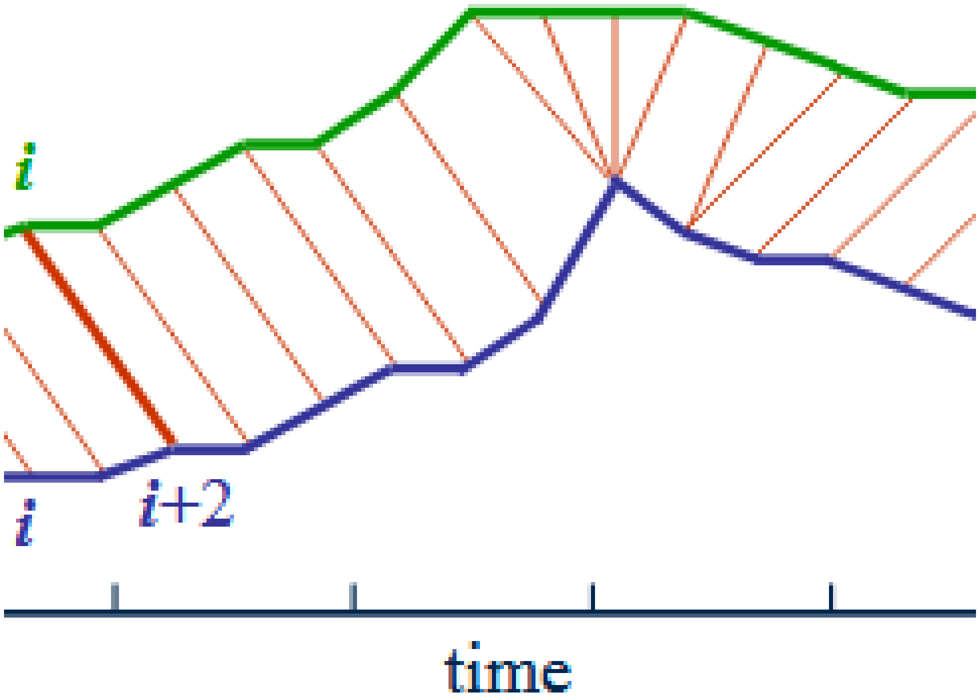
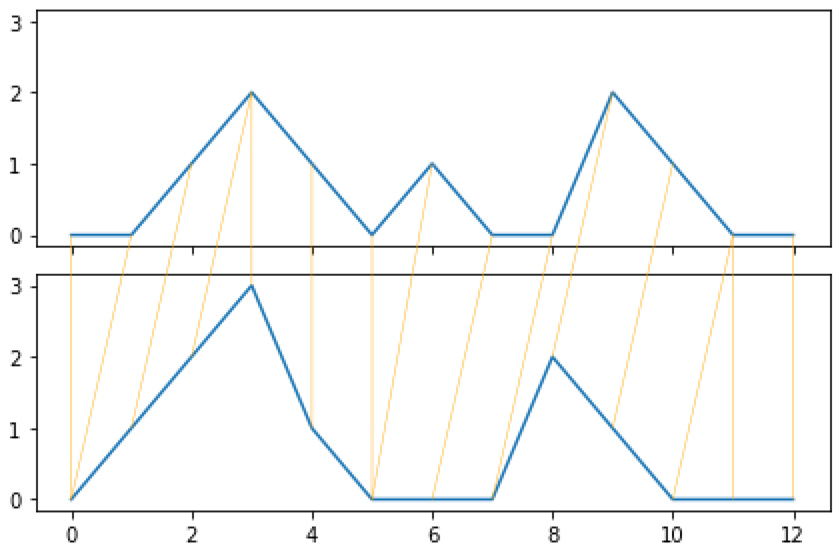
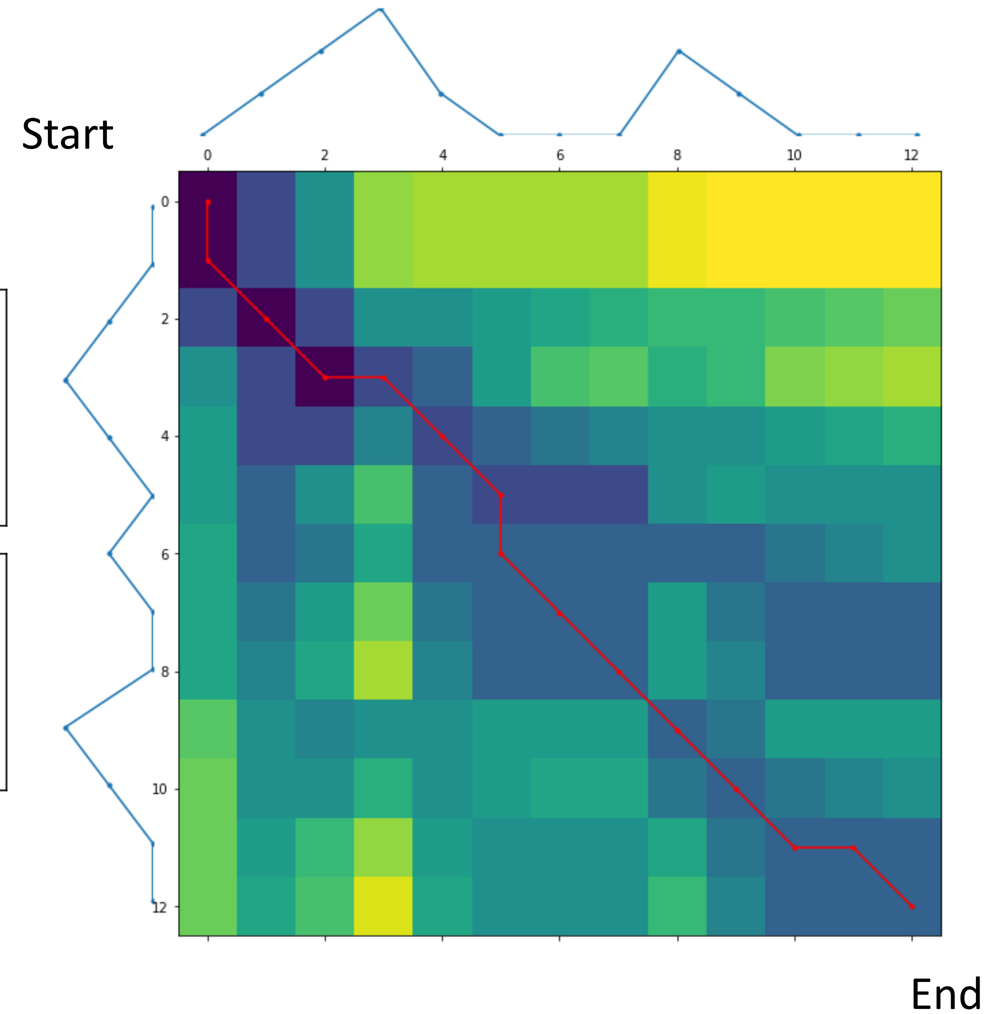


Image from
https://medium.com/datadriveninvestor/dynamic-time-warping-dtw-d51d1a1e4afc

Computation complexity: O(n*m)

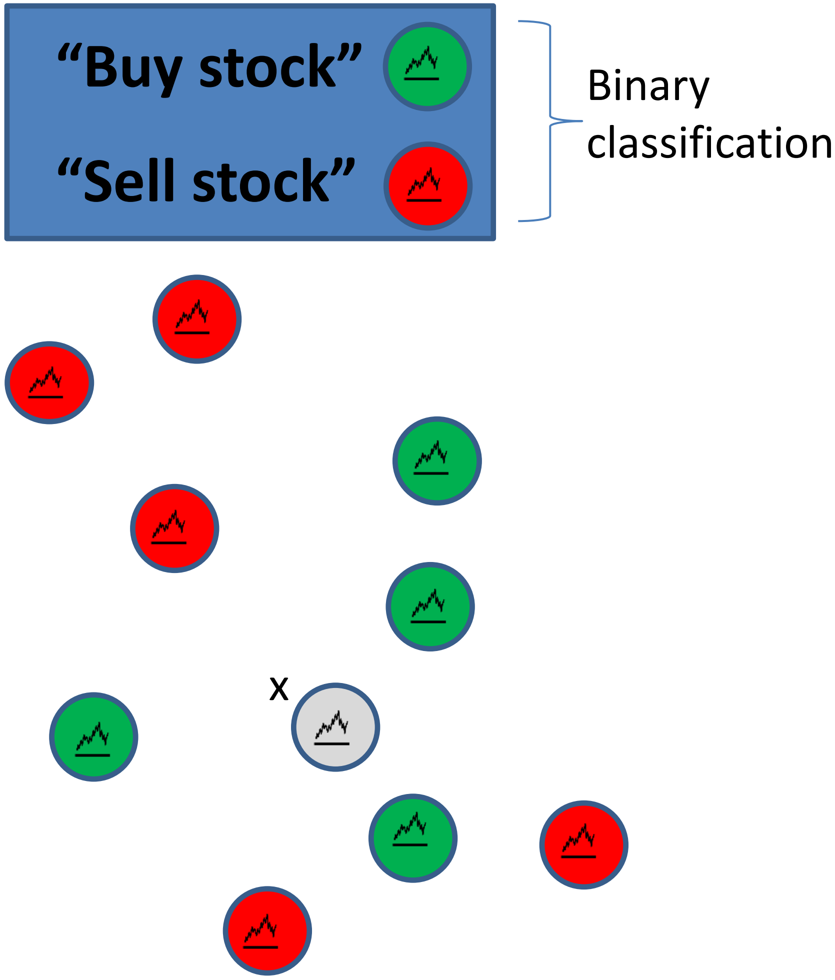




Example: Time Series Classification

Problem Statement

- Given:
  - n time series $x_1,.., x_n$ along with their labels (classes) $y_1,.., y_n$ to be used as training examples
  - a time series **x** with an unknown label
- Goal:
  - classify x: find the class label of x



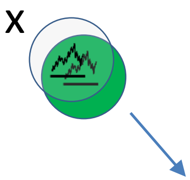
Intuition

In a perfect world:

- Assume there is another data point (time series) x_i that is very similar to the input series x
- I would then pick the label y_i of x_i as my selection
- This decision is **optimal** if $x = x_i$ or, equivalently when $d(x, x_i) \rightarrow 0$

In practice:

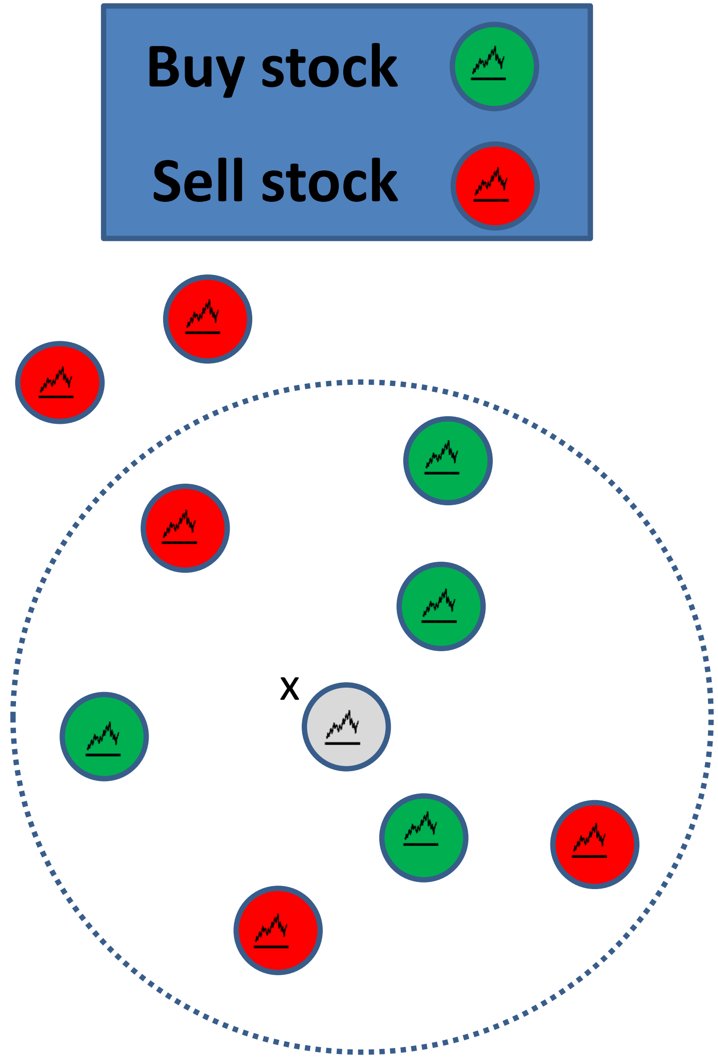
- We will look at labeled data from the neighborhood of x_i



Some example x_i with known label y_i

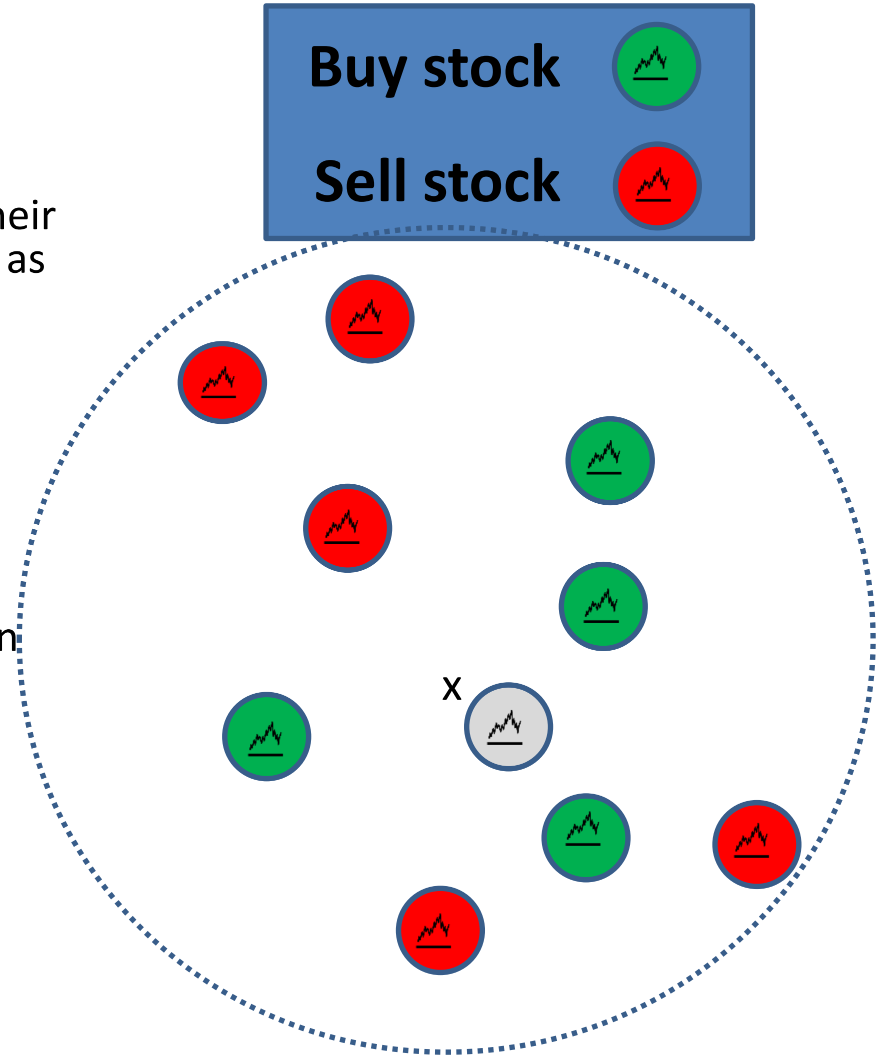
k-NN algorithm

- Given:
  - n time series $x_1,.., x_n$ along with their labels (classes) $y_1,.., y_n$ to be used as training examples
  - a time series x with an unknown label
- Goal:
  - classify x: find the class label of x
- *Intuition:*
  - assign x to the class most common among its *k* nearest neighbours
- Considerations:
  - selection of k
  - weigh neighbours



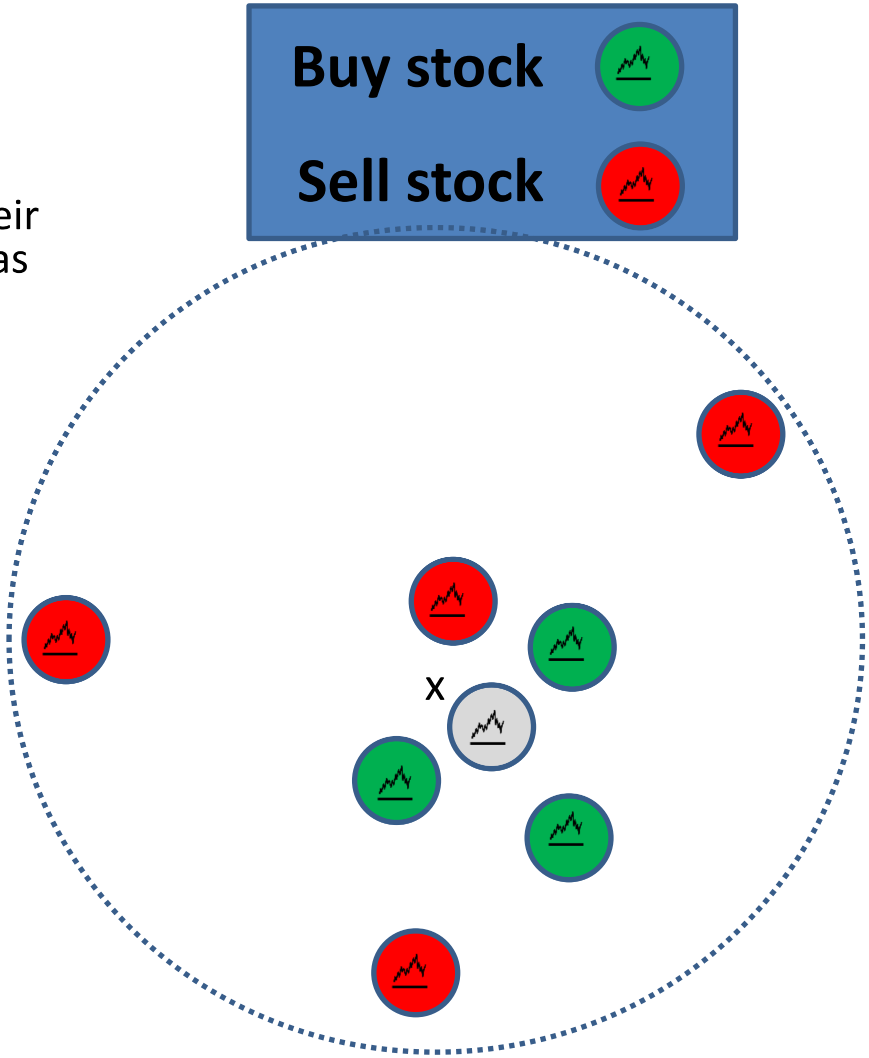
k-NN algorithm

- Given:
  - n time series $x_1,..,x_n$ along with their labels (classes) $y_1,..,y_n$ to be used as training examples
  - a time series x with an unknown label
- Goal:
  - classify x: find the class label of x
- *Intuition:*
  - assign x to the class most common among its *k* nearest neighbours
- Considerations:
  - selection of k
  - weigh neighbours



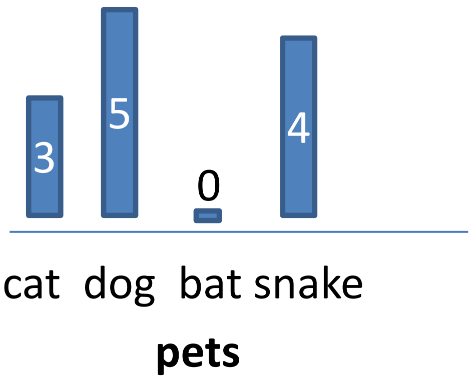
k-NN algorithm

- Given:
  - n time series $x_1,.., x_n$ along with their labels (classes) $y_1,.., y_n$ to be used as training examples
  - a time series x with an unknown label
- Goal:
  - classify x: find the class label of x
- *Intuition:*
  - assign x to the class most common among its *k* nearest neighbours
- Considerations:
  - selection of k
  - weigh neighbours



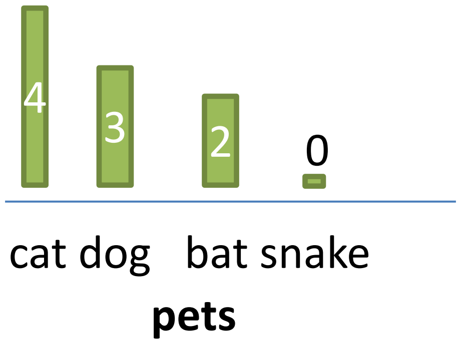
Comparing Distributions (1): Convert to vectors

Pet-shop A



vs

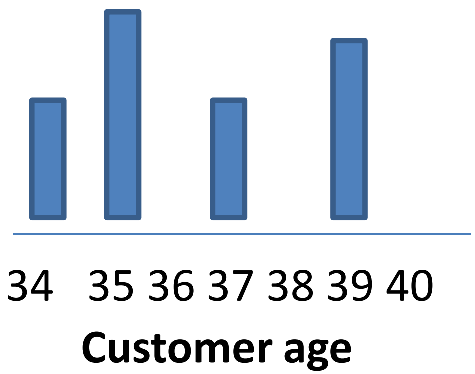
Pet-shop B



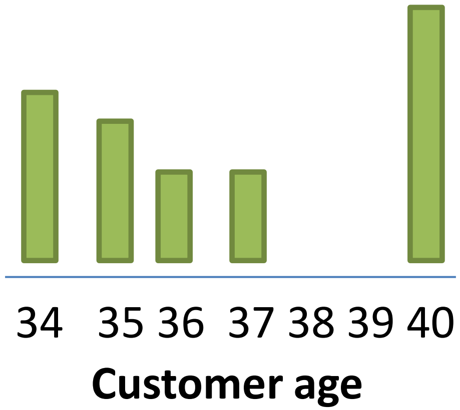
<3,5,0,4> vs. <4,3,2,0>

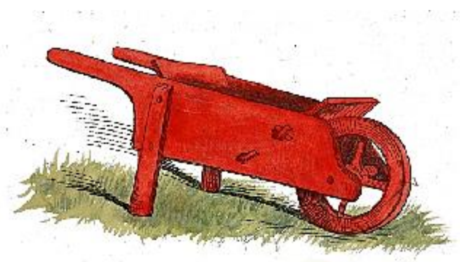
- Makes sense for categorical domains

Comparing Distributions (2): Earth Movers Distance



vs





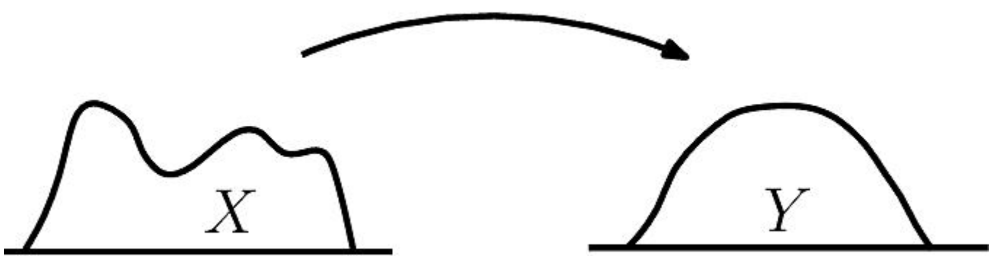
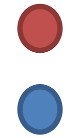


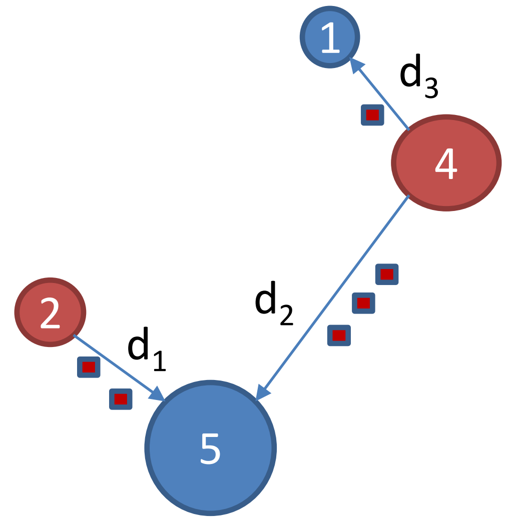
Image source: https://sbl.inria.fr/doc/Earth_mover_distance-user-manual.html

EMD Intuition



- Compute minimum amount of work required to change one distribution into the other.
  - Unit of work: the amount of work necessary to move one unit of weight by one unit of ground distance.
    - Informally: work = amount of dirt moved x distance travelled
  - Ground distance: the distance measure between weight locations.

$$EMD(X,Y) = \frac{\min(Work(X,Y))}{\min(Weight(X), Weight(Y))}$$



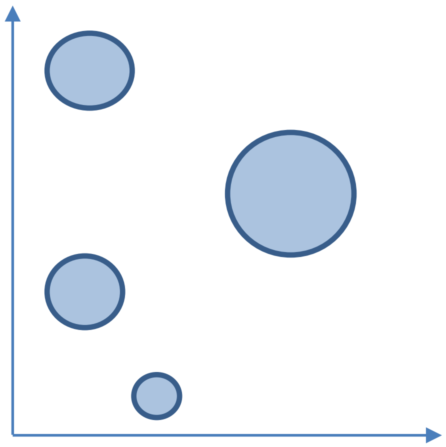
Work = $2^*d_1+3^*d_2+1^*d_3$

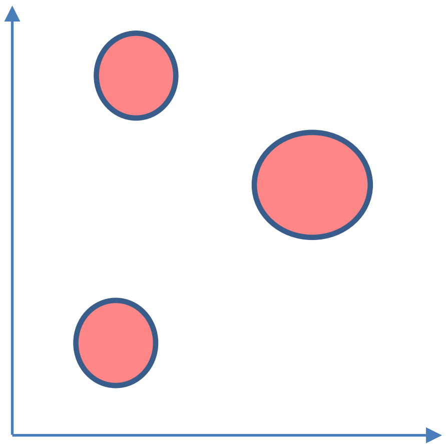
EMD = WORK / 6

- EMD allows partial matching (when cumulative weights don't match): Weight(X)<>Weight(Y)
  - all the weight in the lighter distribution should be matched to weight in the heavier distribution
  - In this case EMD(x,y) is not a distance metric

Compare results of Clustering

- Clusters: $\{(x_i,n_i), i=1,..n\}$
  - x_i is the cluster centroid
  - n_i is the size of the cluster





Compare Features Exported from dataset

- Features: $\{(f_i,n_i), i=1,..n\}$
  - f_i : feature i
  - n_i : number of times f_i appears in dataset
  - Ground distance: $dist(f_i,f_j)$

Neat Application:
Word Movers Distance (Kusner et. al.)