Applied Machine Learning
Concepts, Clustering, Classification

JACEK MALEC, RSS, CS

Introductory quiz

- who has NOT programmed in Java?

Plan for today

- concept learning
- hypothesis space
- classification and clustering
- k-means (clustering)
- k-nearest neighbour (classification)
- (SOM - self organising maps /clustering/)

What is learning?

Elin's definition:

ML is an area of (AI) research providing *powerful tools* that enable machines (computers) to find models describing data and the correlations between them, to

- *predict* future outcomes or developments given previous data (weather, stock market)
- *classify* unknown input given known, classified data (scene contains pedestrian or not)
- *identify structures* in unseen, unlabeled data (grouping people according to different attributes)
- *decide* upon next *steps or actions* to take to maximise reward (new measurement, robot action)

Anything missing?

- Acquiring new knowledge

Consider the following:

- How do we learn what a *chair* is?

- by being shown sufficiently many examples of chairs (and non-chairs)

- Do we build a classifier for chairs? In a sense, yes.
- Do we cluster all our experiences in “chairs” vs “not chairs”? Maybe.
- But we do something more: we learn a **concept.** We can explain it. We can use it in some other context (e.g., solving a problem requiring reaching higher than you can do from the floor). We can **reason** about it.

Concept learning

- a classical example: (Will you) **enjoy sport** (on a particular day)?

Sky	Temp	Humid	Wind	Water	Forecst	EnjoySpt
Sunny	Warm	Normal	Strong	Warm	Same	Yes
Sunny	Warm	High	Strong	Warm	Same	Yes
Rainy	Cold	High	Strong	Warm	Change	No
Sunny	Warm	High	Strong	Cool	Change	Yes

- so what is the general concept?

Language to express our concepts

- Sunny/Rainy (sky)
- Warm/Cold (temperature)
- Normal/High (humidity)
- Strong/Weak (wind)
- Warm/Cool (water)
- Same/Change (forecast)

- Yes/No (Enjoying Sport)

Language to express our concepts

- While learning, we will construct and modify **hypotheses**
- Hypothesis space - all possible hypotheses - many possible representations
- In our case: a hypothesis ***h*** is a conjunction of constraints on attributes, e.g. Water=Warm (specific value), Water=? (don't care) or Water=NOT_ALLOWED

- For example:

Sky	Air Temperature	Humidity	Wind	Water Temperature	Forecast
Sunny	?	?	Strong	?	Same

Our problem

Given

- Instances X: Possible days, each described by the attributes Sky, AirTemp, Humidity, Wind, Water, Forecast,
- Target function c: EnjoySport: X -> {Yes,No}
- Hypotheses H: Conjunctions of literals, e.g., < ?, Cold, High, ?, ?, ?>
- Training examples D: Positive and negative examples of the target function <x1, c(x1)>, <x2, c(x2)>, …, <xn, c(xn)>

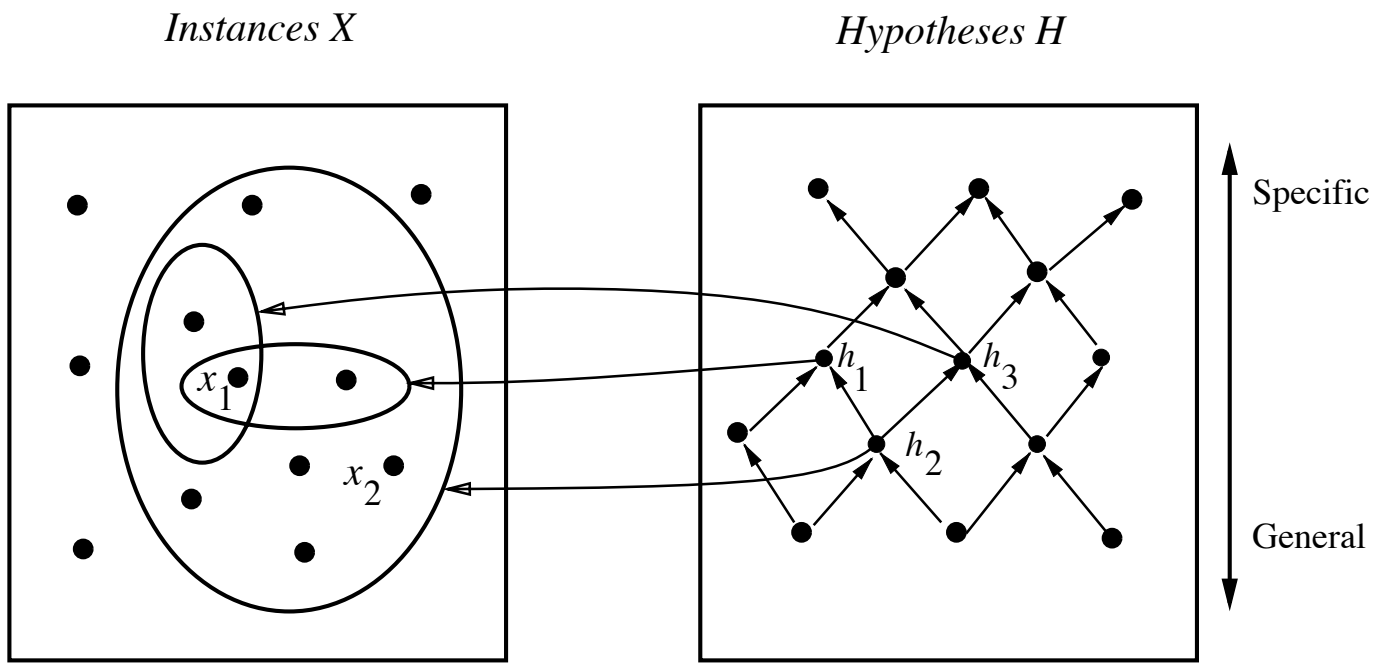
Determine

A hypothesis *h* in H such that
h(x) = c(x) for all x in D

Assumption (the inductive learning hypothesis):

Any hypothesis found to approximate the target function well over *sufficiently large* set of training examples will also *approximate* the target function *well* over other unobserved examples.

Instances and hypotheses



x_1= <Sunny, Warm, High, Strong, Cool, Same>
x_2= <Sunny, Warm, High, Light, Warm, Same>

h_1= <Sunny, ?, ?, Strong, ?, ?>
h_2= <Sunny, ?, ?, ?, ?, ?>
h_3= <Sunny, ?, ?, ?, Cool, ?>

Version Space

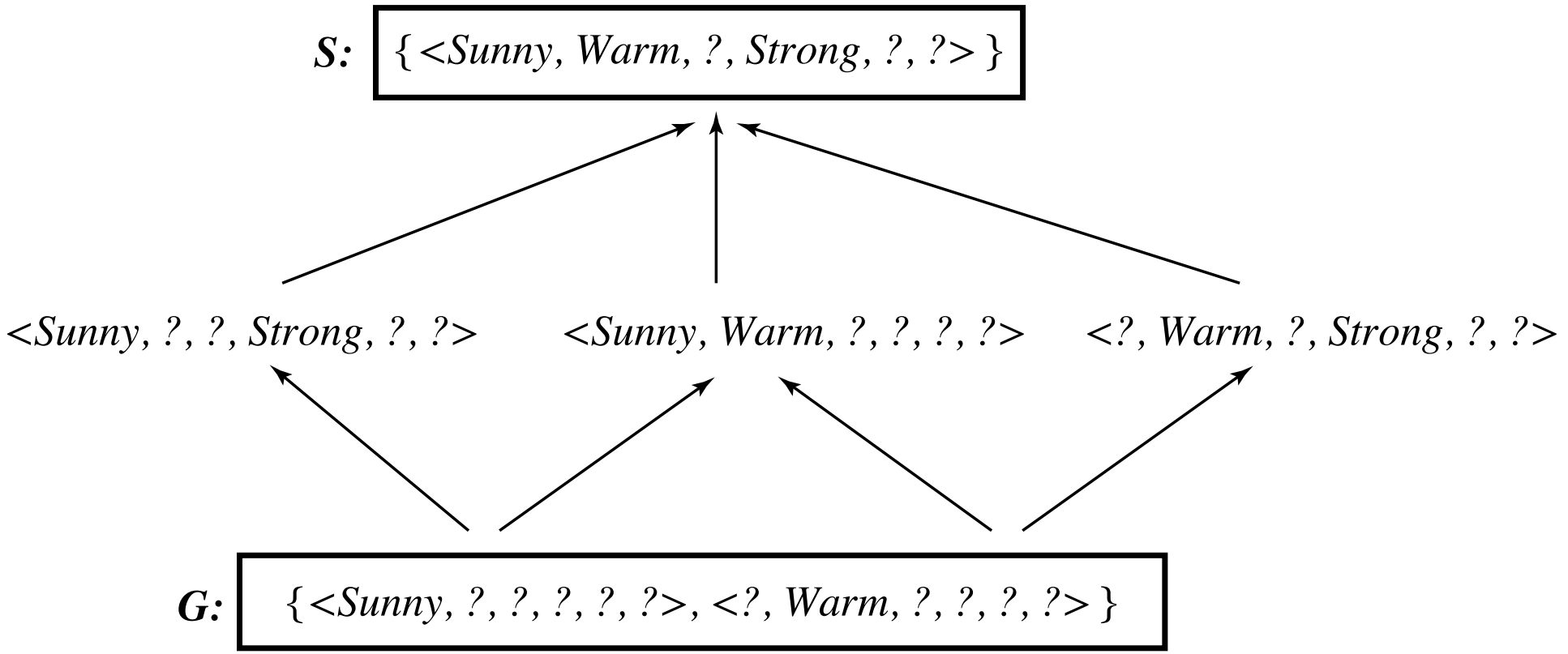
The Version Space with respect to a hypothesis space H and training examples D is the subset of hypotheses from H **consistent** with all training examples in D.

A (trivial) LIST-THEN-ELIMINATE algorithm:

1. Create a list of all hypotheses in H;
2. For each training example <x, c(x)>

   remove from the list any hypothesis h for which h(x) != c(x)
3. Return the list

Version Space example



LUND UNIVERSITY

Candidate elimination algorithm

G <- maximally general hypotheses in H

S <- maximally specific hypotheses in H

For each training example d, do

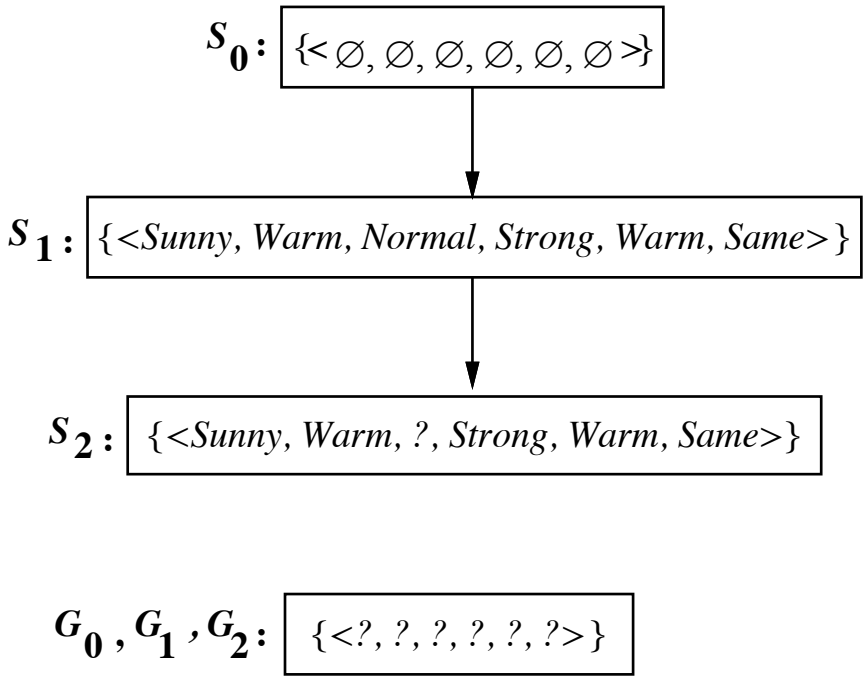
- If d is a positive example
    - Remove from G any hypothesis inconsistent with d
    - For each hypothesis s in S that is not consistent with d
        - Remove s from S
        - Add to S all minimal generalizations h of s such that
            - h is consistent with d, and
            - some member of G is more general than h
        - Remove from S any hypothesis that is more general than another hypothesis in S
- If d is a negative example
    - Remove from S any hypothesis inconsistent with d
    - For each hypothesis g in G that is not consistent with d
        - Remove g from G
        - Add to G all minimal specializations h of g such that
            - h is consistent with d, and
            - some member of S is more specific than h
        - Remove from G any hypothesis that is less general than another hypothesis in G

Example

S_0: {<Ø, Ø, Ø, Ø, Ø, Ø>}

G_0: {<?, ?, ?, ?, ?, ?>}

Example



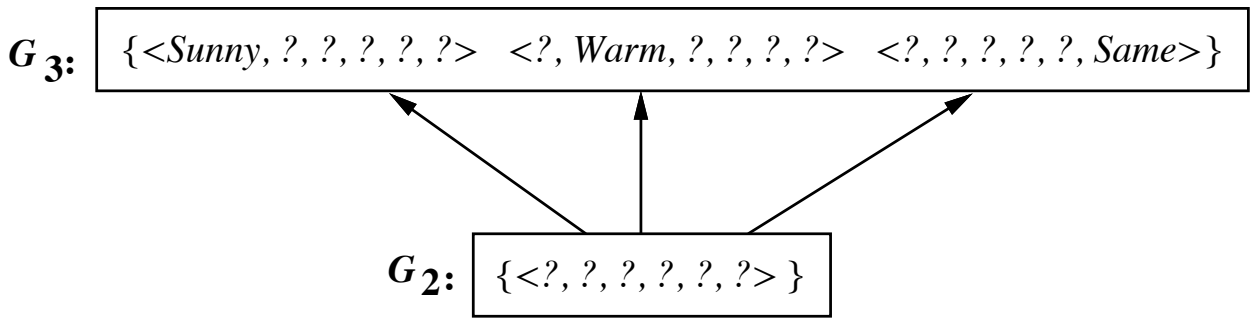
Training examples:

1 . <*Sunny, Warm, Normal, Strong, Warm, Same*>, *Enjoy Sport = Yes*

2 . <*Sunny, Warm, High, Strong, Warm, Same*>, *Enjoy Sport = Yes*

Example

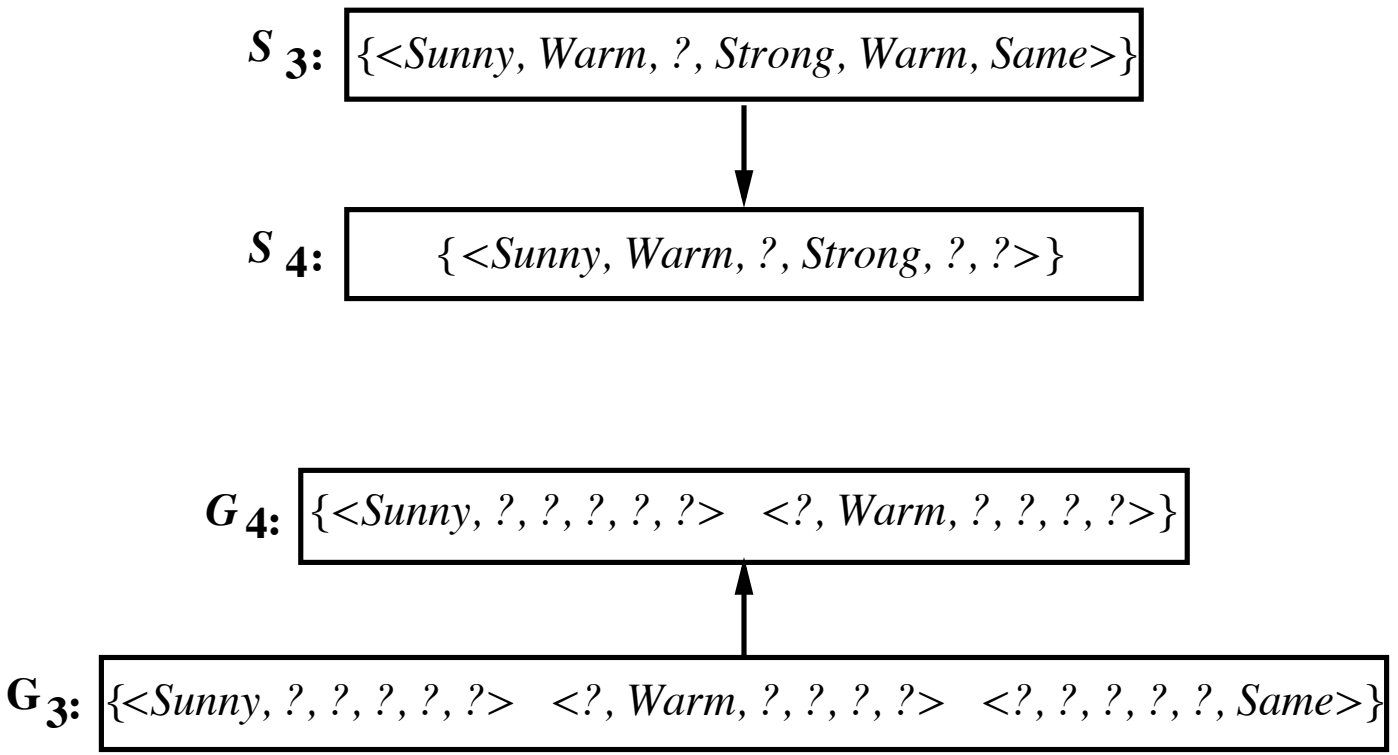
S_2, S_3: { <Sunny, Warm, ?, Strong, Warm, Same> }



Training Example:

3. <Rainy, Cold, High, Strong, Warm, Change>, EnjoySport=No

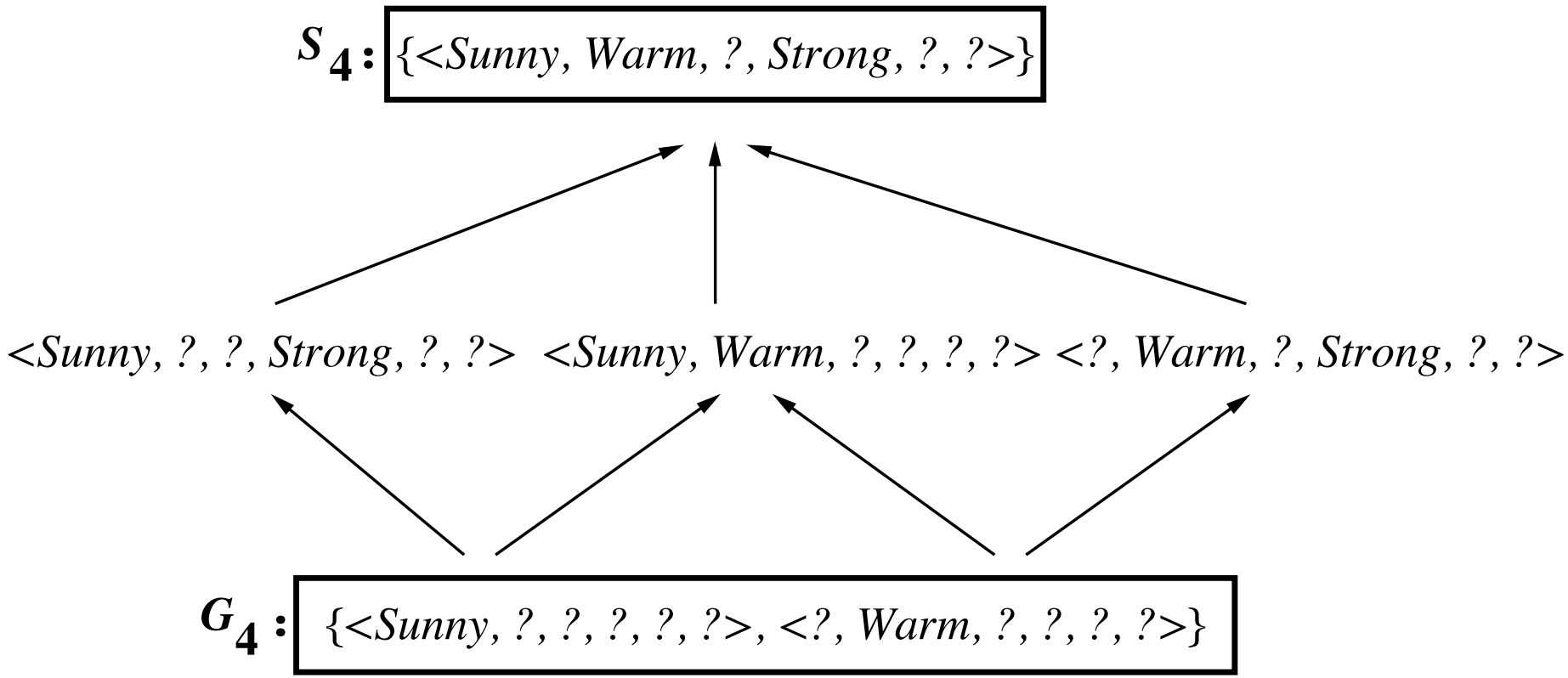
Example



Training Example:

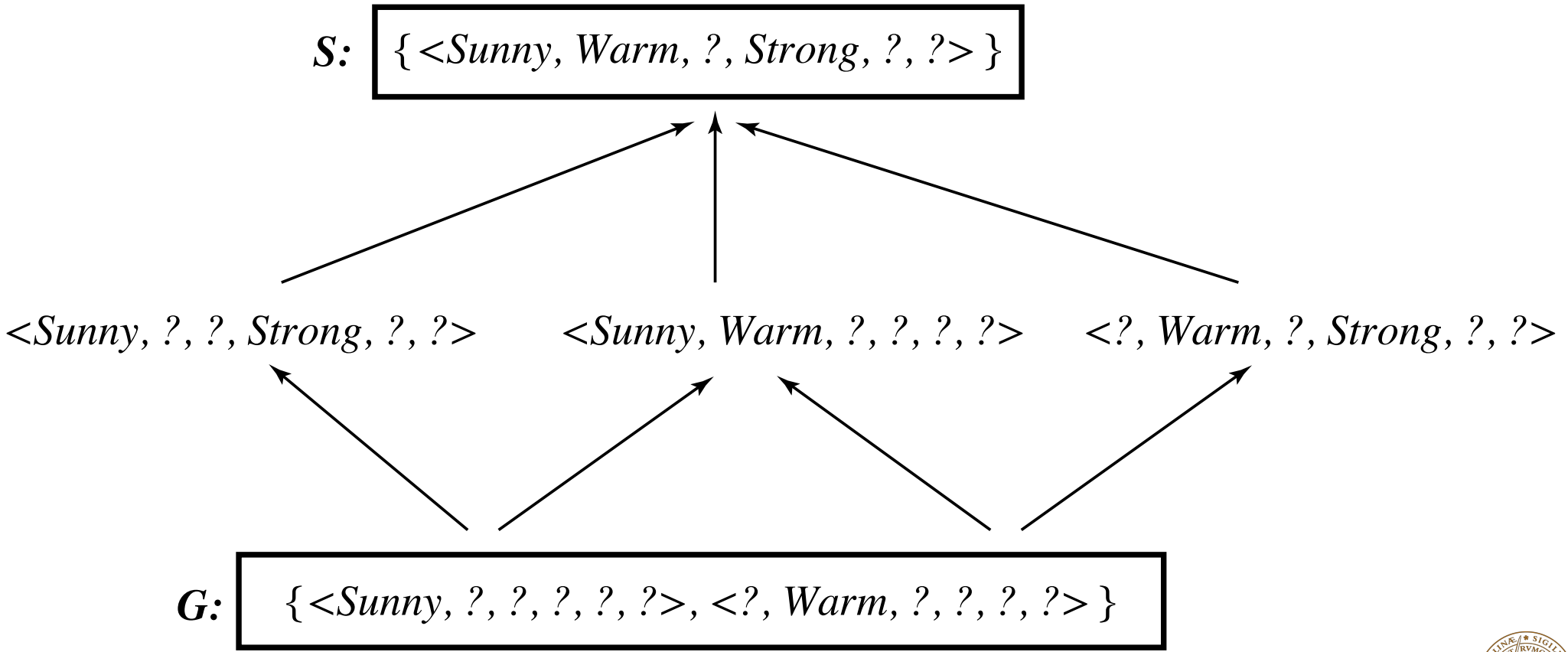
4.<*Sunny*, *Warm*, *High*, *Strong*, *Cool*, *Change*>, *EnjoySport* = *Yes*

Example



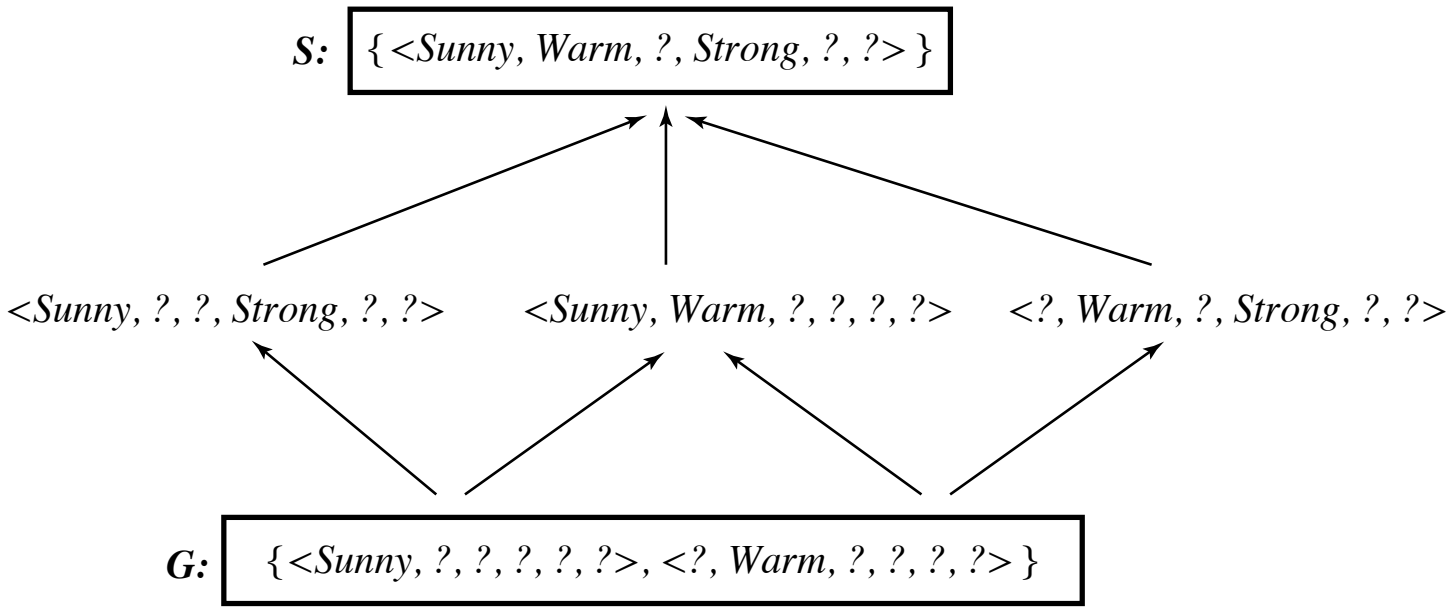
LUND UNIVERSITY

What should be the next example?



LUND UNIVERSITY

How to classify the following:



- < Sunny, Warm, Normal, Strong, Cool, Change>
- <Rainy, Cool, Normal, Light, Warm, Same>
- <Sunny, Warm, Normal, Light, Warm, Same>

What justifies the inductive leap?

From

- < Sunny, Warm, Normal, Strong, Cool, Change>
- <Sunny, Warm, Normal, Light, Warm, Same>

to

- < Sunny, Warm, Normal, ?, ?, ?>

Why believe we can classify the unseen

<Sunny, Warm, Normal, Strong, Warm, Same>?

Inductive bias

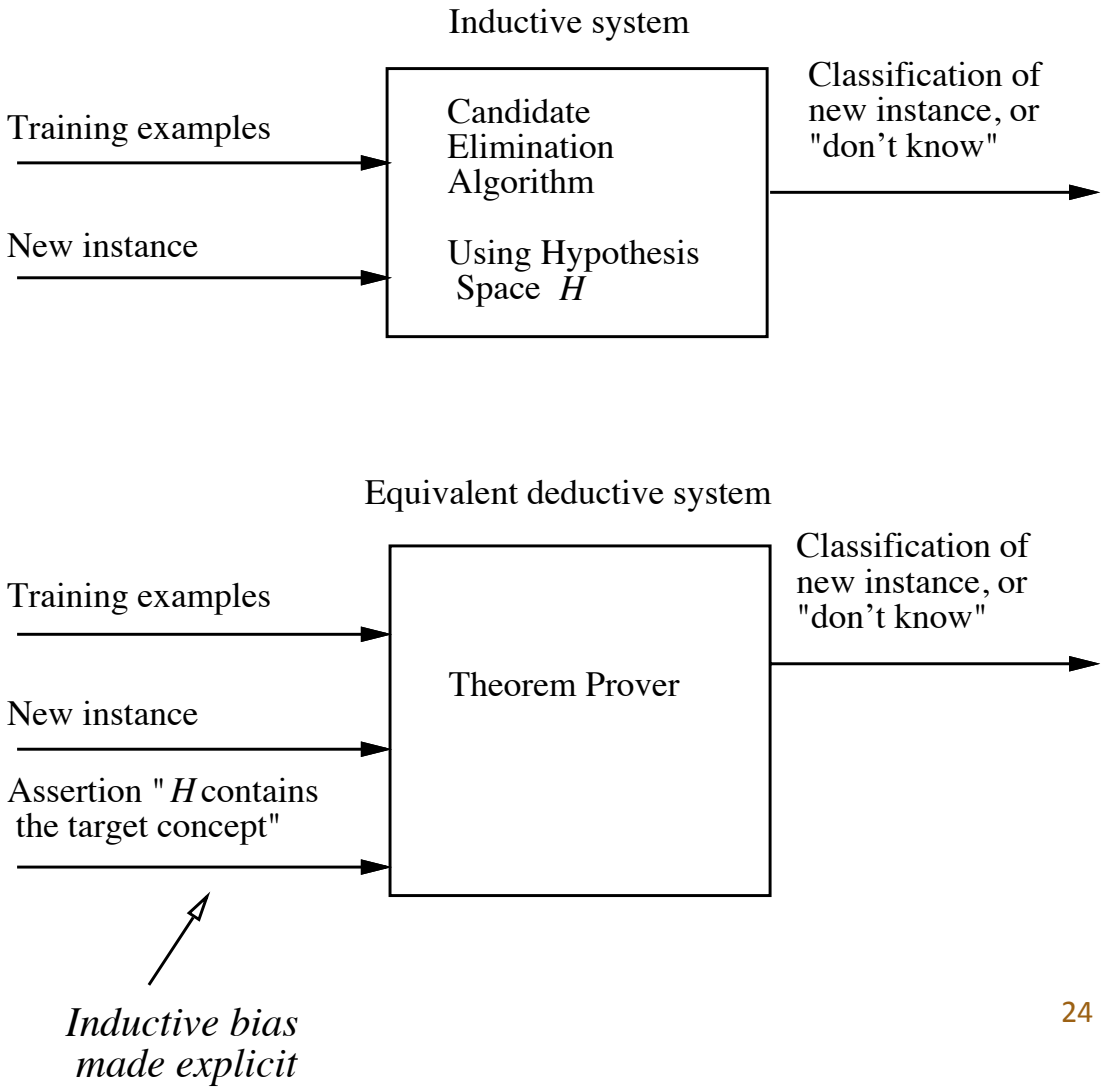
We prefer some hypotheses over others.

Bias - this preference. Depends on the language, but then may vary.

E.g.

1. Rote learning
2. Version spaces candidate elimination algorithm
3. List then eliminate algorithm

Inductive bias made explicit:



LUND UNIVERSITY

Clustering

Cluster analysis (clustering) is the task of grouping a set of objects in such a way that objects in the same group (called a **cluster**) are more similar (in some sense) to each other than to those in other groups (clusters).

Many methods for modelling clusters (Wikipedia):

- *Connectivity models*: for example, hierarchical clustering builds models based on distance connectivity.
- ***Centroid models***: for example, the k-means algorithm represents each cluster by a single mean vector.
- *Distribution models*: clusters are modeled using statistical distributions, such as multivariate normal distributions used by the expectation-maximization algorithm.
- *Density models*: for example, DBSCAN and OPTICS defines clusters as connected dense regions in the data space.
- *Subspace models*: in biclustering (also known as co-clustering or two-mode-clustering), clusters are modeled with both cluster members and relevant attributes.
- *Group models*: some algorithms do not provide a refined model for their results and just provide the grouping information.
- *Graph-based models*: a clique, that is, a subset of nodes in a graph such that every two nodes in the subset are connected by an edge can be considered as a prototypical form of cluster. Relaxations of the complete connectivity requirement (a fraction of the edges can be missing) are known as quasi-cliques, as in the HCS clustering algorithm.
- ***Neural models***: the most well known unsupervised neural network is the self-organizing map and these models can usually be characterized as similar to one or more of the above models, and including subspace models when neural networks implement a form of Principal Component Analysis or Independent Component Analysis.

k-means

Given a set of observations

x1, **x2**,…, **xn**

find *k* sets

S1, S2,…, Sk

such that the *Within-Cluster-Sum-of-Squares* is minimal.

What is WCSS then?

k-means

Algorithm (Lloyd's):

1. randomly initialise k values
2. assign data to clusters by choosing the minimal distance (nearest) mean
3. compute centroids taking data in current clusters into account

Repeat 2. and 3. until convergence

Often repeated with several initialisations.

Only local minimum is found!

k-means

https://commons.wikimedia.org/wiki/File:K-means_convergence.gif

Expectation Maximisation (EM)

Given samples generated by randomly chosen probability distributions belonging to a predefined set of distributions (e.g., 3 different gaussian distributions) estimate parameters of those distributions (e.g., the 3 means and variances for the gaussians).

Given data D find the maximum likelihood hypothesis h such that p(D|h) is maximal.

EM algorithm:

0. randomly generate parameters

1. estimate the probability of choosing the distributions, given data D

2. replace the parameters with new hypothesis so that probability estimation is maximised

Repeat 1 and 2.

Self-organising maps (Kohonen maps)

or: Self-Organising Feature Map

A neural network, trained in an unsupervised way, providing a low-dimensionality (e.g. 2) representation of the training samples.

Uses *competitive learning* (as opposed to error-correction learning).

Find a BMU (best matching unit = closest node to the example) and adjust it and its neighbours towards the example.

https://upload.wikimedia.org/wikipedia/commons/thumb/9/91/Somtraining.svg/1280px-Somtraining.svg.png

Notes: toroidal grids, similar to k-means for small number of nodes, initialisation matters

Classification

Given a number of classes (categories) C1, C2,…,Cn, determine which class the instance (observation) **x** belongs to.

Variants:

- binary (yes/no)
- multi-valued
- fuzzy
- non-deterministic
- …

k-nearest neighbour

- A lazy classification algorithm

- Teaching: just store the instances, together with their class membership

- Classification:
    1. find k nearest instances
    2. take majority vote
- Regression:
    A. find k nearest examples
    B. compute the value by taking some kind of mean, weighed by the distance

k-nearest neighbour

Some problems:

- euclidean distance only useful in continuous case
- better distance function can be learnt with specialised algorithms
- influenced by number of class elements (more frequent ones tend to take over)
- feature scales are important
- redundant features makes the approach inefficient, eliminate (e.g., PCA)
- large training sets make the algorithm computationally intensive, approximations exist
- useful for outlier detection (the larger the distance to the NN, the larger the probability that a point is an outlier)