LEARNING

Outline

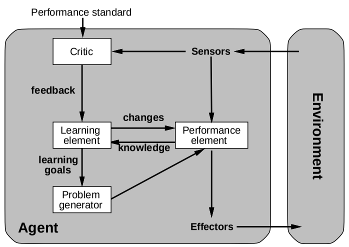
- General Model of Learning Agents
- Inductive Learning
- Learning Decision Trees
- Using Information Theory
- Learning Logical Descriptions
- Learning Belief Networks

Based on the textbook by S. Russell & P. Norvig:

Artificial Intelligence, A Modern Approach,
Chapters 18.1-5, 18.7, and 19.6

GENERAL MODEL OF LEARNING AGENTS

- A **learning agent** consists of four conceptual components:



1. A **performance element** (a conventional agent) which is responsible for choosing external actions.
2. A **learning element** which aims to improve the agent.
3. A **critic** which evaluates the performance of the agent.
4. A **problem generator** which suggests new courses of action.

Example. Consider dividing an automated taxi-driving agent into four components mentioned above.

Design of the Learning Element

The design of the learning element if affected by four major choices:

1. Which *components* of the performance element are improved.
2. What kind of internal *representation* is used for those components.
3. What kind of *feedback* is available to the agent.
4. What *prior knowledge* on the environment is available.

Components of the performance element

- The components may include the following:
  1. A direct mapping from the current state to actions.
  2. Means to infer relevant properties of the world from percepts.
  3. Information about the way the world evolves.
  4. Information about the possible outcomes of the agent's actions.
  5. Utility information indicating the desirability of (performing particular actions in) particular world states.
  6. Goals describing states that maximize the agent's utility.
- Each of these can be learned – given appropriate feedback.
- Various kinds of internal representations can be used for the components: polynomials, logical rules, belief networks, etc.

Available Feedback

Different kinds of learning situations can be distinguished:

- **Supervised learning**: the outputs that a component generates for particular inputs can be compared with the correct outputs (which are provided by an external *teacher*).
- **Unsupervised learning**: the correct outputs are not known.

  Example. An unsupervised learner may learn to predict its future percepts given its percept history so far.
- **Reinforcement learning**: the outputs get evaluated somehow (for instance, the agent receives a *reward* or a *punishment*), but the correct outputs remain unknown.

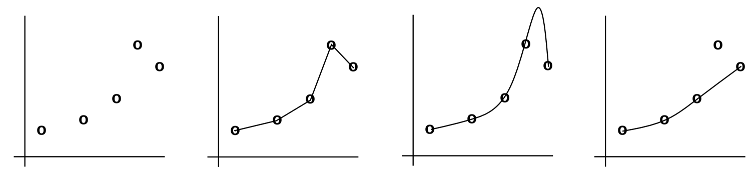
Any *prior knowledge* on the environment helps enormously in learning!

INDUCTIVE LEARNING

- In general, learning can be understood as a process of determining a representation for some function f of interest.
- An **example** is a pair $\langle x, f(x)\rangle$ where x is the input and $f(x)$ is the output of the function f applied to x.
- The task of **pure inductive inference** (or **induction**) is:

  *Given a collection of examples of f, return a function h (called a **hypothesis**) that approximates f.*
- Typically, there are many hypotheses conforming to the examples.
- In **incremental learning**, the collection of examples grows gradually, and the agent updates its hypothesis accordingly.

Example. Consider a set of points $\langle x, y\rangle$ in xy-plane such that $y = f(x)$. The task is to find $h(x)$ that fits the points well.



- As f is unknown, there are many choices for h, but without further knowledge there is no way to prefer (b), (c), or (d).
- Any preference for one hypothesis over another beyond mere consistency with examples is called a **bias**.
- All learning algorithms exhibit some sort of bias.

A Reflex Agent Taught by a Teacher

- The agent maintains a collection of pairs of percepts and actions:

```
global examples ← {}

function REFLEX-PERFORMANCE-ELEMENT(percept) returns an action
   if (percept, a) in examples then return a
   else
      h ← INDUCE(examples)
      return h(percept)

procedure REFLEX-LEARNING-ELEMENT(percept, action)
   inputs: percept, feedback percept
           action, feedback action
   examples ← examples ∪ {(percept, action)}
```

- There is no commitment to how the hypothesis is represented.
- Currently, there exist algorithms (cf. INDUCE above) for learning logical rules, nonlinear numerical functions, belief networks, etc.
- There is a clear trade-off between *expressiveness* and *efficiency*.

LEARNING DECISION TREES

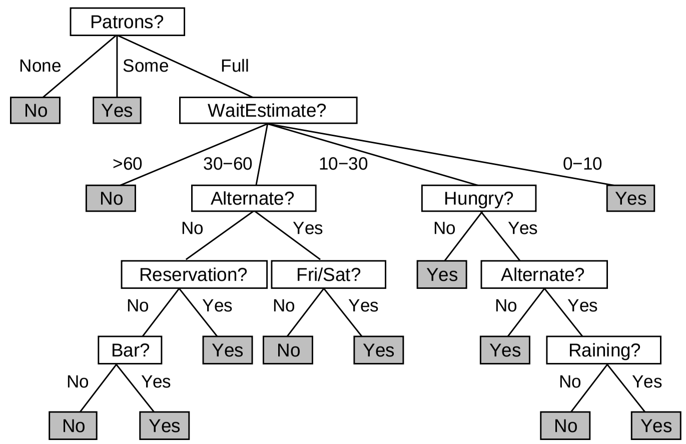
- A **decision tree** is a representation of a function f from an n-dimensional attribute space to the set $\{Yes, No\}$.

  Thus f can be understood as a Boolean-*valued* function.
- Decision trees are structured as follows:
  1. Each internal node tests the value of an attribute and the branches are labeled by the values of the attribute.
  2. Leaf nodes contain the Yes/No answer for the **goal predicate** the values of which are represented by the decision tree.
- Arbitrary Boolean functions can be represented as decision trees.
- Functions with larger range of outputs can also be represented.

Example. Consider the problem of deciding whether to wait for a table at a restaurant. The aim is to learn a decision tree for the goal predicate $WillWait$ using the following attributes:

1. $Alternate$: is there a suitable alternative restaurant nearby?
2. Bar: is there a comfortable bar area to wait in?
3. Fri/Sat: is it Friday or Saturday?
4. $Hungry$: are we hungry?
5. $Patrons$: the number of people ($None$, $Some$, $Full$) in the restaurant.
6. $Price$: the price range of the restaurant (\$, \$\$, \$\$\$).
7. $Raining$: is it raining outside?
8. $Reservation$: has a reservation been made beforehand?
9. $Type$: the type ($French$, $Italian$, $Thai$, $Burger$) of the restaurant.
10. $WaitEstimate$: the estimate in minutes ($0\text{-}10$, $10\text{-}30$, $30\text{-}60$, >60).

Example. Mr. Russell makes decisions for this domain as follows:



☞ $Price$ and $Type$ attributes are considered irrelevant.

Expressiveness of Decision Trees

- Paths of decision trees can be expressed as logical implications:

$$\forall r(Patrons(r, Full) \wedge WaitEstimate(r, 0\text{-}10) \wedge Hungry(r) \rightarrow WillWait(r)).$$

- Full first order logic is not easily covered.

  ☞ Decision trees are effectively propositional.
- Any boolean function can be encoded as a decision tree, but such a representation may require an exponential space.

  Example. The sizes of decision trees for **parity** and **majority functions** grow exponentially in the number of variables.
- There are 2^{2^n} different Boolean functions (with n Boolean attributes). When $n = 6$, this number is about 1.8×10^{19}.

Inducing Decision Trees from Examples

➤ An **example** is described by a combination of values for the attributes and the corresponding value of the goal predicate.

Example. Consider the following set of **positive** and **negative** examples for the goal predicate $WillWait$.

Example	Attributes										Goal
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
X_1	*Yes*	*No*	*No*	*Yes*	*Some*	*$$$*	*No*	*Yes*	*French*	*0–10*	*Yes*
X_2	*Yes*	*No*	*No*	*Yes*	*Full*	*$*	*No*	*No*	*Thai*	*30–60*	*No*
X_3	*No*	*Yes*	*No*	*No*	*Some*	*$*	*No*	*No*	*Burger*	*0–10*	*Yes*
X_4	*Yes*	*No*	*Yes*	*Yes*	*Full*	*$*	*No*	*No*	*Thai*	*10–30*	*Yes*
X_5	*Yes*	*No*	*Yes*	*No*	*Full*	*$$$*	*No*	*Yes*	*French*	*>60*	*No*
X_6	*No*	*Yes*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Italian*	*0–10*	*Yes*
X_7	*No*	*Yes*	*No*	*No*	*None*	*$*	*Yes*	*No*	*Burger*	*0–10*	*No*
X_8	*No*	*No*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Thai*	*0–10*	*Yes*
X_9	*No*	*Yes*	*Yes*	*No*	*Full*	*$*	*Yes*	*No*	*Burger*	*>60*	*No*
X_{10}	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$$$*	*No*	*Yes*	*Italian*	*10–30*	*No*
X_{11}	*No*	*No*	*No*	*No*	*None*	*$*	*No*	*No*	*Thai*	*0–10*	*No*
X_{12}	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$*	*No*	*No*	*Burger*	*30–60*	*Yes*

➤ The task is to construct a decision tree for $WillWait$ using this set of examples as a **training set**.

➤ A trivial solution encodes each example as a path leading to a leaf:

1. Along the path, all the attributes are tested in turn.
2. The leaf node holds the correct classification for the example.

➤ Such a decision tree produces correct classifications for the examples in the training set, but does not cover other cases.

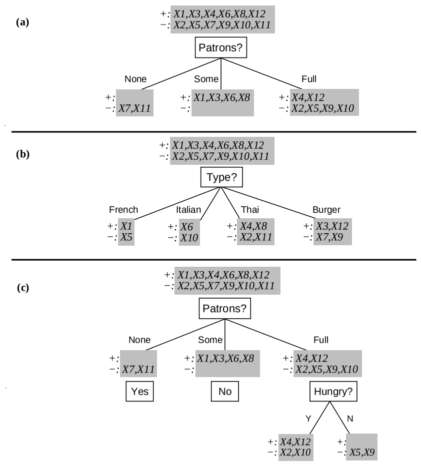
➤ A central principle of inductive learning is called **Ockham's razor**:

"*The most likely hypothesis is the simplest one that is consistent with all observations.*"

➤ It is intractable to find the smallest decision tree for a training set, but relatively small ones can be found using a heuristics.

➤ The basic idea is to test the most important attribute first, i.e., the one that best classifies examples in the training set.

Example. In the restaurant example, the attribute $Patrons$ yields a much better classification than the attribute $Type$.



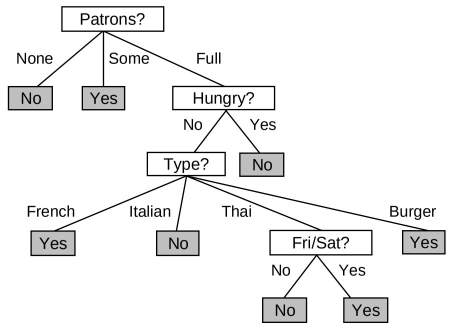
An Algorithm for Learning Decision Trees

➤ The process can be formalized as a concrete learning algorithm:

```
function DECISION-TREE-LEARNING(examples, attributes, default) returns a decision tree
  inputs: examples, set of examples
          attributes, set of attributes
          default, default value for the goal predicate

  if examples is empty then return default
  else if all examples have the same classification then return the classification
  else if attributes is empty then return MAJORITY-VALUE(examples)
  else
    best ← CHOOSE-ATTRIBUTE(attributes, examples)
    tree ← a new decision tree with root test best
    for each value v_i of best do
      examples_i ← {elements of examples with best = v_i}
      subtree ← DECISION-TREE-LEARNING(examples_i, attributes − best,
                                       MAJORITY-VALUE(examples))
      add a branch to tree with label v_i and subtree subtree
    end
    return tree
```

- The training set is split into smaller sets of examples that are solved as recursive instances of the decision tree learning problem.
- The recursive problems fall into four different categories:
  1. If there are both positive and negative examples, then one of the best attributes is chosen to split the examples.
  2. If all the remaining examples are positive (or all negative), then the answer is Yes (or No).
  3. If there are no examples left, the majority classification at the node's parent is returned as a default value.
  4. If there are no attributes left, but both positive and negative examples, there is **noise** in the data *or* the set of attributes is insufficient to fully determine the goal predicate.
- A way to handle the last category is to use a majority vote.

Example. The following tree is obtained for the earlier training set:



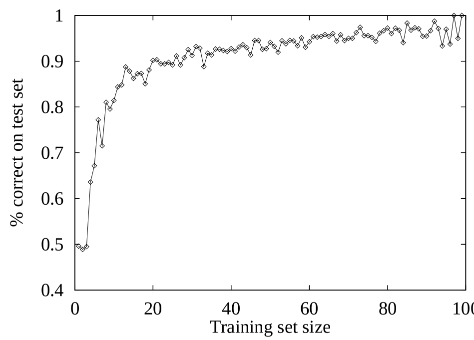
- The resulting decision tree is much simpler than the original tree (which was actually used for generating the training set).
- Despite simplicity, the decision tree produces a correct classification for every example in the training set.

Assessing the Learning Element Performance

- A learning algorithm is good if it produces hypotheses which yield correct classifications for as many unseen examples as possible.
- A way to evaluate the performance of a learning algorithm is to
  1. Collect a large set of examples and divide it into a **training set** and a separate **test set**.
  2. Apply the learning algorithm to the examples in the training set in order to generate a hypothesis H.
  3. Measure the percentage of examples in the test set that are correctly classified by the hypothesis H.
  4. Repeat steps 1-3 for random training sets of increasing size.

- The performance of a specific learning algorithm can be depicted as a **learning curve** that gives the percentage of correct classifications on the test set as a function of the training set size.

Example. The learning curve below shows how the decision tree learning algorithm performs in the restaurant example:



Case Study: Learning to Fly

- Decision tree learning has been applied to flying a Cessna airplane on a flight simulator [Sammut et al., 1992].
- The data was generated by watching three skilled human pilots performing an assigned flight plan 30 times each.
- In all, 90000 examples were obtained – each described by 20 state variables and labelled by the action taken by the pilot.
- The decision tree that resulted from these was converted into C code and inserted to the flight simulator's control loop.
- Surprisingly, the program was able to fly *better* than its teachers.

USING INFORMATION THEORY

- A perfect attribute divides the set of examples into subsets in which examples are all positive or all negative.
- One suitable measure for comparing attributes is the expected amount of **information** (in the sense proposed by Shannon) obtained by learning the exact values of attributes.

Example. Suppose you are going to bet 1€ on the flip of a coin.

1. If $P(Heads) = 0.99$, then $\mathrm{EMV} = 0.99 \times 1€ - 0.01 \times 1€ = 0.98€$ and $\mathrm{VPI}(Heads) = 1€ - 0.98€ = 0.02€$.
2. If $P(Heads) = 0.5$, then $\mathrm{EMV} = 0.5 \times 1€ - 0.5 \times 1€ = 0€$ and $\mathrm{VPI}(Heads) = 1€ - 0€ = 1€$.

☞ The less you know, the more valuable the information.

Measuring Information Content

- Information theory uses the same intuition, but it measures information content in **bits** rather than value of information.
- In general one bit of information is enough to answer a yes/no question about which one has no idea.
- In general, the information content I of the actual value of V is

$$I(P(v_1), \ldots, P(v_n)) = \sum_{i=1}^{n} -P(v_i) \log_2 P(v_i) \text{ (bits)}.$$

where $P(v_1), \ldots, P(v_n)$ are the probabilities for the possible values $v_1, \ldots, v_n$ of the variable V.

Example. The information content $I(0.5, 0.5) = -0.5 \log_2 0.5$ $-0.5 \log_2 0.5 = 1$ bits, but $I(0.99, 0.01) \approx 0.08$ bits.

Information Gain

- In case of decision trees, the **information gain** from getting to know the exact value of a v-valued attribute A is given by

$$\mathrm{Gain}(A) = I(\frac{p}{p+n}, \frac{n}{p+n}) - \mathrm{Remainder}(A)$$

where the *remaining information content*

$$\mathrm{Remainder}(A) = \sum_{i=1}^{v} (\frac{p_i + n_i}{p+n} \times I(\frac{p_i}{p_i + n_i}, \frac{n_i}{p_i + n_i}))$$

and p (p_i) and n (n_i) are the numbers of positive and negative examples (that have the i^{th} value of A in common).

Example. More information is gained from $Patrons$ than from $Type$:

$\mathrm{Gain}(Patrons) = 1 - [\frac{2}{12}I(0,1) + \frac{4}{12}I(1,0) + \frac{6}{12}I(\frac{2}{6}, \frac{4}{6})] \approx 0.541$ and

$\mathrm{Gain}(Type) = 1 - [\frac{2}{12}I(\frac{1}{2}, \frac{1}{2}) + \frac{2}{12}I(\frac{1}{2}, \frac{1}{2}) + \frac{4}{12}I(\frac{2}{4}, \frac{2}{4}) + \frac{4}{12}I(\frac{2}{4}, \frac{2}{4})] = 0.$

Noise and Overfitting

- Recall the possibility of noise in the training set (there are two examples with identical attribute values, but classifications differ).
- **Overfitting** means that a (decision tree) learning algorithm forms a consistent hypothesis using *irrelevant attributes* for classification even when *relevant attributes* are missing.
- The information gain is close to zero for irrelevant attributes.
- The relevance of attributes can be tested: the total deviation

$$D = \sum_{i=1}^{v} \left(\frac{(p_i - \hat{p}_i)^2}{\hat{p}_i} + \frac{(n_i - \hat{n}_i)^2}{\hat{n}_i} \right)$$

  where $\hat{p}_i = p \times \frac{p_i + n_i}{p+n}$ and $\hat{n}_i = n \times \frac{p_i + n_i}{p+n}$ distributes according to the χ^2 distribution with $v - 1$ degrees of freedom.
- Decision trees can be **pruned** by neglecting irrelevant attributes.

Broadening the Applicability of Decision Trees

In order to extend decision tree induction to a wider variety of problems, several problems have to be addressed.

1. **Missing values:** an example X lacking the value of an attribute A is given the majority classification among those obtained by assuming that X has each value of A in turn.
2. **Multivalued attributes:** when an attribute has a large number of possible values (e.g. $RestaurantName$), the information gain gives a misleading indication on the usefulness of the attribute.

   A solution is to use **gain ratio** instead of plain information gain.
3. **Continuous-valued attributes** (e.g. $Price$) are not well suited for decision-tree learning, and have to be **discretized** somehow.

LEARNING

GENERAL LOGICAL DESCRIPTIONS

- Inductive learning can be viewed as a process of searching for a good hypothesis in a large **hypothesis space** which is determined by the representation language chosen for the task.
- In the sequel, the aim is to describe the interconnections of examples, hypotheses, and the goal in logical terms.
- This helps understanding inductive learning in more general/complex forms compared to learning decision trees.

Hypotheses

- The goal is a predicate $Q(x)$ for which **candidate definitions** $C_i(x)$ are formed as hypotheses $H_i = \forall x(Q(x) \leftrightarrow C_i(x))$.

Example. For the decision tree learned in the restaurant example:

$$\begin{aligned}\forall r(\, WillWait(r) \leftrightarrow & Patrons(r, Some) \lor \\ & (Patrons(r, Full) \land \neg Hungry(r) \land Type(r, French)) \lor \\ & (Patrons(r, Full) \land \neg Hungry(r) \land Type(r, Thai) \land Fri/Sat(r)) \lor \\ & (Patrons(r, Full) \land \neg Hungry(r) \land Type(r, Burger))\ )\end{aligned}$$

- The **extension** of a hypothesis $H_i = \forall x(Q(x) \leftrightarrow C_i(x))$ is the set of examples X for which $Q(X)$ evaluates to true.

  ☞ Logically equivalent hypotheses have equal extensions.
- The hypothesis space $\{H_1, \ldots, H_n\}$ of a learning algorithm is denoted by $\mathbf{H}$ and it is usually believed that one of the hypotheses in the space $\mathbf{H}$ is correct, i.e., $H_1 \lor \ldots \lor H_n$ is true.

Classifying Examples with Hypotheses

- Given a hypothesis $H_i = \forall x(Q(x) \leftrightarrow C_i(x))$, an example X is **positive/negative** if $Q(X)/\neg Q(X)$ evaluates to true.

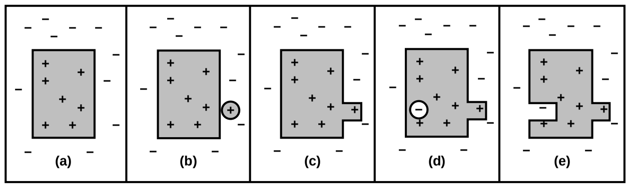
Example. The first example X_1 in the training set of the restaurant example is a positive one, as $WillWait(X_1)$ evaluates to true.

- An example X corresponds to a conjunction of literals which define the values of attributes and the goal predicate for X.
- A **false** positive/negative example X for a hypothesis $H_i = \forall x(Q(x) \leftrightarrow C_i(x))$ gets an incorrect classification by H_i.

  ☞ X (as a conjunction of literals) is inconsistent with H_i.
- Inductive learning can be understood as a process of gradually eliminating hypotheses that are inconsistent with examples.

Current-Best-Hypothesis Search

- The idea is to maintain a single hypothesis H, and to adjust it if new false positive/negative examples w.r.t. H are encountered.
- The current hypothesis H is illustrated in the figure (a) below.
- A false negative example (b) can be removed by a **generalization** (c) that extends the extension of the current hypothesis H_i.
- A false positive example (d) can be removed by a **specialization** (e) that narrows the extension of the current hypothesis H_i.



Skeletal Algorithm

Current-best-hypothesis search is captured by the following algorithm:

```
function CURRENT-BEST-LEARNING(examples) returns a hypothesis
  H ← any hypothesis consistent with the first example in examples
  for each remaining example in examples do
     if e is false positive for H then
        H ← choose a specialization of H consistent with examples
     else if e is false negative for H then
        H ← choose a generalization of H consistent with examples
     if no consistent specialization/generalization can be found then fail
  end
  return H
```

- Generalizations and specializations imply *logical relationships*:

  E.g., if $H_1 = \forall x(Q(x) \leftrightarrow C_1(x))$ is a generalization of $H_2 = \forall x(Q(x) \leftrightarrow C_2(x))$, then $\forall x(C_2(x) \rightarrow C_1(x))$ holds.
- Note that H_2 is a specialization of H_1 in the setting above.

Example. A way to generalize is to **drop conditions** from definitions. For instance, $C_1(x) \leftrightarrow Patrons(x, Some)$ generalizes the definition $C_1(x) \leftrightarrow Alternate(x) \wedge Patrons(x, Some)$.

Example. Hypotheses are formed in the restaurant example as follows:

H_1: $\forall x(WillWait(x) \leftrightarrow Alternate(x))$

H_2: $\forall x(WillWait(x) \leftrightarrow Alternate(x) \wedge Patrons(x, Some))$

H_3: $\forall x(WillWait(x) \leftrightarrow Patrons(x, Some))$

H_4: $\forall x(WillWait(x) \leftrightarrow Patrons(x, Some) \vee (Patrons(x, Full) \wedge Fri/Sat(x)))$

There are also other hypotheses conforming to the first four examples:

H'_4: $\forall x(WillWait(x) \leftrightarrow \neg WaitEstimate(x, 30\text{-}60))$

H''_4: $\forall x(WillWait(x) \leftrightarrow Patrons(x, Some) \vee (Patrons(x, Full) \wedge WaitEstimate(x, 10\text{-}30)))$

Least-Commitment Search

- The original hypothesis space can be viewed as a disjunction

$$H_1 \vee \cdots \vee H_n.$$

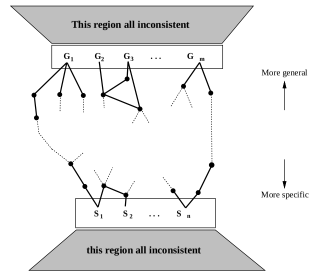
- Hypotheses which are consistent with all examples encountered so far form a set of hypotheses called the **version space** V.
- Version space is shrunk by the **candidate elimination** algorithm:

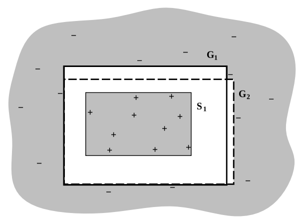
```
function VERSION-SPACE-LEARNING(examples) returns a version space
   local variables: V, the version space: the set of all hypotheses

   V ← the set of all hypotheses
   for each example e in examples do
      if V is not empty then V ← VERSION-SPACE-UPDATE(V, e)
   end
   return V
---------------------------------------------------------------
function VERSION-SPACE-UPDATE(V, e) returns an updated version space
   V ← {h ∈ V : h is consistent with e}
```


Boundary Sets

- The algorithm finds a subset of the version space V that is consistent with all examples in an *incremental* way.
- Candidate elimination is an example of a **least-commitment** algorithm, as no arbitrary choices are made among hypotheses.
- Since the hypothesis space V is possibly enormous, it cannot be represented directly as a set of hypotheses or a disjunction.
- The problem can be alleviated by **boundary sets** $\{S_1, \ldots, S_n\}$ (**S-set**) and $\{G_1, \ldots, G_m\}$ (**G-set**) and a partial ordering among hypotheses induced by specialization/generalization.
- Any hypothesis H between a most specific boundary S_i and a most general boundary G_j is consistent with the examples seen.

- Boundary sets for the version space are illustrated below:





- Initially, the S-set contains a single hypothesis $\forall x(Q(x) \leftrightarrow \mathit{False})$ while the G-set contains $\forall x(Q(x) \leftrightarrow \mathit{True})$ only.
- Upon a false negative/positive example, a most specific boundary S is replaced by all its immediate generalizations / deleted.
- Upon a false positive/negative example, a most general boundary G is replaced by all its immediate specializations / deleted.

These operations on S-sets and G-sets are continued until:

1. There is exactly one hypothesis left in the version space.
2. The version space *collapses* (i.e., the S-set or G-set becomes empty): there are no consistent hypotheses for the training set.
3. We run out of examples with several hypotheses remaining in the version space: a solution is to take the majority vote.

Discussion

- If the domain contains noise or insufficient attributes for exact classification, the version space will always collapse.
- If unlimited disjunction is allowed when hypotheses are formed, the S-set/G-set will always contain a single boundary.
- A solution is to allow only limited forms of disjunction.

BAYESIAN LEARNING

- The aim is to make a prediction concerning an unknown quantity X given some data D and hypotheses $H_1, H_2, \ldots$.
- Assuming that each H_i specifies a complete distribution for X, **full Bayesian learning** is characterized by

$$\mathbf{P}(X \mid D) = \sum_i \mathbf{P}(X \mid H_i)\mathbf{P}(H_i \mid D).$$

- In most cases, computing $\mathbf{P}(H_i \mid D)$ is intractable.
- A common approximation is to use **maximum a posteriori** (MAP) hypothesis H_{MAP} – a hypothesis H_i that maximizes $\mathbf{P}(H_i \mid D)$:

$$\mathbf{P}(X \mid D) \approx \mathbf{P}(X \mid H_{\mathrm{MAP}}).$$

- Since $\mathbf{P}(H_i \mid D) = \frac{\mathbf{P}(D|H_i)\mathbf{P}(H_i)}{\mathbf{P}(D)}$ and $P(D)$ is fixed, it is sufficient to maximize $\mathbf{P}(D \mid H_i)\mathbf{P}(H_i)$ in order to determine H_{MAP}.
- This maximization process involves determining the prior probabilities $P(H_i)$ for the possible hypotheses H_i.
- The relation of MAP hypotheses to preferring simpler hypotheses (like Ockham's razor principle) is not yet fully understood.
- The only reasonable policy is to assign prior probabilities $P(H_i)$ based on some simplicity measure on hypotheses.
- In some cases, the prior probabilities $P(H_i)$ can be assumed to be **uniformly** distributed.
- Then maximizing $\mathbf{P}(D \mid H_i)$ produces a **maximum-likelihood** (ML) hypothesis H_{ML} – a special case of H_{MAP}.

Belief Network Learning Problems

The learning problem for belief networks comes in several varieties:

1. **Known structure, fully observable:** only CPTs are learned and the statistics of the set of examples can be used.
2. **Unknown structure, fully observable:** this involves heuristic search through the space of structures – guided by the ability of modeling data correctly (MAP or ML probability value).
3. **Known structure, hidden variables:** analogy to neural networks.
4. **Unknown structure, hidden variables:** no good/general algorithms are known for learning in this setting.

SUMMARY

- Learning is essential for dealing with unknown environments.
- Learning may take several forms depending on the chosen representation, available feedback, and prior knowledge.
- The aim of **inductive learning** is to learn a **function** from examples of its inputs and outputs.
- **Ockham's razor** principle suggests choosing the simplest hypothesis that matches the examples observed.
- The performance of inductive learning algorithms is measured by their prediction accuracy as a function of the training set size.
- **Bayesian learning** methods can be used to learn representations of probabilistic functions, particularly belief networks.