AITA : Machine Learning

What is Machine Learning?

Any study of ***Machine Learning*** should begin with a formal definition of what is meant by ***Learning***. A definition due to Simon (1983) is one of the best:

> “Learning denotes changes in the system that are adaptive in the sense that they enable the system to do the same task (or tasks drawn from the same population) more effectively the next time.”

We can easily extend this definition easily to our AI systems:

> “Machine learning denotes automated changes in an AI system that are adaptive in the sense that they enable the system to do the same task (or tasks drawn from the same population) more effectively the next time.”

The details of Machine Learning depends on the underlying knowledge representations, e.g. learning in neural networks will be very different to learning in rule based systems.

Types of Learning

The strategies for learning can be classified according to the amount of inference the system has to perform on its training data. In increasing order we have

1. **Rote learning** – the new knowledge is implanted directly with no inference at all, e.g. simple memorisation of past events, or a knowledge engineer's direct programming of rules elicited from a human expert into an expert system.

2. **Supervised learning** – the system is supplied with a set of training examples consisting of inputs and corresponding outputs, and is required to discover the relation or mapping between then, e.g. as a series of rules, or a neural network.

3. **Unsupervised learning** – the system is supplied with a set of training examples consisting only of inputs and is required to discover for itself what appropriate outputs should be, e.g. a *Kohonen Network* or *Self Organizing Map*.

Early expert systems relied on rote learning, but for modern AI systems we are generally interested in the supervised learning of various levels of rules.

The Need for Learning

We have seen that extracting knowledge from human experts, and converting it into a form useable by the inference engine of an expert system, is an arduous and labour intensive process.

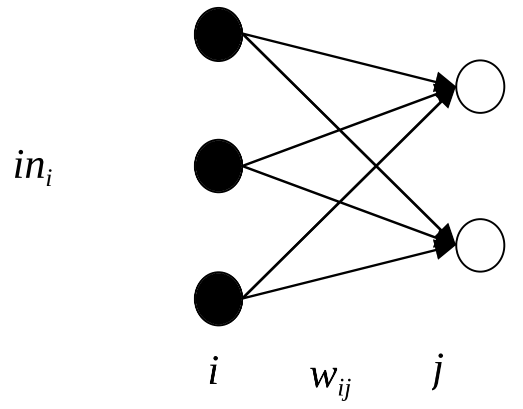
As with many other types of AI system, it is much more efficient to give the system enough knowledge to get it started, and then leave it to learn the rest for itself. We may even end up with a system that learns to be better than a human expert.

The ***general learning approach*** is to generate potential improvements, test them, and discard those which do not work. Naturally, there are many ways we might generate the potential improvements, and many ways we can test their usefulness. At one extreme, there are model driven (top-down) generators of potential improvements, guided by an understanding of how the problem domain works. At the other, there are data driven (bottom-up) generators, guided by patterns in some set of training data.

We shall now look in turn at learning in neural, evolutionary, and rule-based systems.

Learning in Neural Network Systems

Recall that ***neural networks*** consist of many simple processing units (that perform addition and smooth thresholding) with activation passing between them via weighted connections. Learning proceeds by iteratively updating the connection weights w_{ij} in such a way that the output errors on a set of training data are reduced.



$$out_j = \mathrm{Sigmoid}(\sum in_i w_{ij})$$

The standard procedure is to define an output error measure (such as the sum squared difference between the actual network outputs and the target outputs), and use ***gradient descent*** weight updates to reduce that error. The details are complex, but such an approach can learn from noisy training data and generalise well to new inputs.

Learning in Evolutionary Computation Systems

Evolutionary computation systems simulate the evolution by ***natural selection*** that is seen in biological systems. Typically one creates a whole population of AI agents defined by some ***genotypic representation***, and measures their individual performance levels (or fitnesses) on the given task or problem. The most fit individuals are chosen from each generation to survive and 'breed' to form the next generation.

The simulated breeding process involves ***cross-over*** and ***mutation*** of genetic material. This will map into the individuals' fitness and drive the selection process. In this way, good recombinations and mutations will proliferate in the population, and we will end up with generations of individuals which are increasingly good at their given tasks.

Often evolutionary improvements and lifetime learning are combined in the same system, and we end up with an approach that is superior to either on their own. We find the evolution of particularly good learners, and that learned behaviour can be assimilated into the genotype (the ***Baldwin Effect***).

Problems Facing Expert Systems

We can identify four major limitations facing conventional expert systems:

1. **Brittleness** – Expert systems generally only have access to highly specific domain knowledge, so they cannot fall back on more general knowledge when the need arises, e.g. to deal with missing information, or when information appears inconsistent.

2. **Lack of Meta-Knowledge** – Expert systems rarely have sophisticated knowledge about their own operation, and hence lack an appreciation of their own limitations.

3. **Knowledge Acquisition** – Despite an increasing number of automated tools, this remains a major bottleneck in applying expert system technology to new domains.

4. **Validation** – Measuring the performance of expert systems is difficult because it is not clear how to quantify the use of knowledge. The best we can do is compare their performance against that of human experts.

Progress on the first two points should follow once we have made progress on the third point. We shall now look at how machine learning techniques can help us here.

Types of Learning in Rule Based Systems

The principal kinds of learning appropriate for rule based systems are

1. The invention of new ***conditions and/or actions*** for rules.
2. The invention of new ***conflict resolution strategies*** (i.e. meta-rules).
3. The discovery and ***correction of errors*** in the existing system.

For learning new rules (including meta-rules) there are two basic approaches:

1. **Inductive rule learning** methods create new rules about a domain that are not derivable from any previous rules. We take some 'training data', e.g. examples of an expert performing the given task, and work out corresponding rules that also ***generalize*** to new situations.
2. **Deductive rule learning** enhances the efficiency of a system's performance by deducing new rules from previously known domain rules and facts. Having the new rules should not change the outputs of the system, but should make it perform more efficiently.

Meta-Rules and Meta-Knowledge

In order to learn effectively, we need to be able to reason about the rules, and to have an understanding of the state of the knowledge base. We need to have ***meta-rules***, i.e. rules about the rules, and ***meta-knowledge***, i.e. knowledge about the knowledge.

An example of a meta-rule is:

IF: there are rules which **do not** mention the current goal in their premise,
AND there are rules which **do** mention the current goal in their premise,
THEN: the former rule should be used in preference to the latter

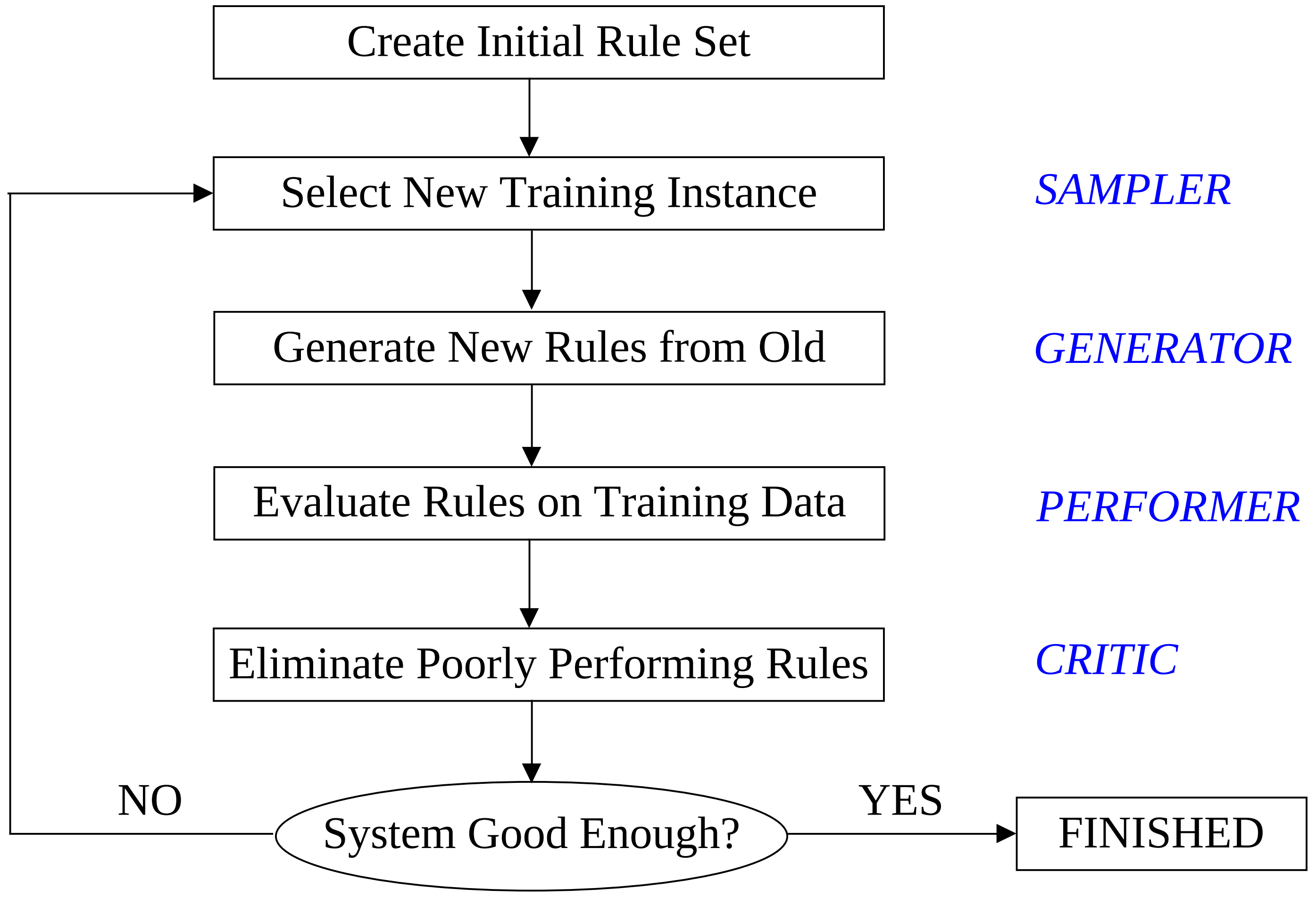
An example of a meta-knowledge is:

Knowledge/information that can be frequently used in strong rules is more important than knowledge/information that is rarely used and only appears in weak rules.

Using meta-rules and meta-knowledge involves ***meta-level inference***.

Rule Induction Systems

The simplest rule induction system can be represented by the following flowchart:



Create Initial Rule Set
Select New Training Instance
SAMPLER
Generate New Rules from Old
GENERATOR
Evaluate Rules on Training Data
PERFORMER
Eliminate Poorly Performing Rules
CRITIC
NO
System Good Enough?
YES
FINISHED

Rule Refinement Strategies

There are numerous approaches one can take to improve the rules in an existing rule based systems. A good rule refinement program should involve:

1. **Removing redundancy.** More than one rule may deal with essentially the same situation – unnecessary rules should be removed to increase efficiency.

2. **Merging rules.** Sometimes a set of rules can be merged into a single, more general, rule that has the same effect. Doing this will improve efficiency.

3. **Making rules more specific.** If a rule is too general it can make incorrect predictions. Such rules should be made more specific to reduce errors.

4. **Making rules more general.** If a rule can be made more general without introducing errors, it should be, as it is likely to improve generalization.

5. **Selecting the final rules.** The process of specialization and generalization my have introduced more redundancy, so step 1 is applied again to remove them.

Concept Learning and Classification

The above procedures for generating and testing and refining rules make good sense, but for large (i.e. useful) systems we need to formulate a more ***systematic procedure***.

The idea of ***concept learning*** and ***classification*** is that given a training set of positive and negative instances of some concept (which belongs to some pre-enumerated set of concepts), the task is to ***generate rules*** that classify the training set correctly, and that also ‘recognize’ unseen instances of that concept, i.e. ***generalize*** well.

To do this we work with a set of ***patterns*** that describe the concepts, i.e. patterns which state those properties which are common to all individual instances of each concept. The simplest patterns are nothing more than the descriptors that specify the rule conditions (i.e. the IF parts of the rules) that relate to the given concept.

We will clearly need to be able to ***match*** efficiently given instances in the training set against the hypothetical/potential descriptions of the concepts.

Version Spaces and Partial Ordering

The idea of a ***version space*** is simply a way of representing the space of all concept descriptions (rule conditions) consistent with the training instances seen so far.

Efficient representation and update of version spaces can be achieved by defining a ***partial order*** over the patterns generated by any concept definition language.

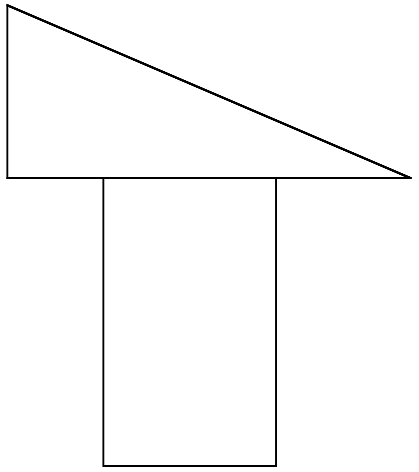
We can do this by defining the relation “more specific than or equal to” as follows:

> “Pattern *P1* ***is more specific than or equal to*** pattern *P2* (written as $P1 \leq P2$) if and only if *P1* matches a subset of all the instances that *P2* matches.”

For example, *P1* = “car” is a fairly general concept pattern, *P2* = “American car” is more specific, and *P3* = “yellow American cars with sun-roofs and alloy wheels” is an even more specific pattern. We can write $P3 \leq P2 \leq P1$. Note that we can order *P4* = “blue car” with respect to *P1* but not *P2* and *P3*, so the ordering is only partial.

Version Spaces – Blocks World Example

Consider the following simple example from Winston’s “blocks world”:



P1:

$$\begin{tikzcd}\boxed{\begin{array}{l}\text{STANDING}\\ \text{BRICK}\end{array}} \arrow[r, "\text{SUPPORTS}"] & \boxed{\begin{array}{l}\text{LYING}\\ \text{WEDGE or BRICK}\end{array}}\end{tikzcd}$$

P2:

$$\begin{tikzcd}\boxed{\begin{array}{l}\text{not LYING}\\ \text{any shape}\end{array}} \arrow[r, "\text{TOUCHES}"] & \boxed{\begin{array}{l}\text{any orientation}\\ \text{WEDGE or BRICK}\end{array}}\end{tikzcd}$$

Clearly, pattern *P1* is more specific than pattern *P2*, because the constraints imposed by *P1* are only satisfied if the weaker constraints imposed by *P2* are satisfied. So *P1* ≤ *P2*.

Note that, for a program to perform this partial ordering, it would need to “understand” the relevant concepts and relationships, e.g. that wedges and bricks are different shapes, that supporting implies touching, and so on.

Version Spaces – Boundary Sets

Once a system can grasp the relationship of specificity, the version space can be represented in terms of its ***maximally specific*** and ***maximally general*** patterns.

The system can consider the version space as containing:

1. The set $S = \{S_i\}$ of maximally specific patterns.
2. The set $G = \{G_i\}$ of maximally general patterns.
3. All concept descriptions which occur between these two sets in the partial ordering.

This is called the ***boundary sets representation*** for version spaces, which is both

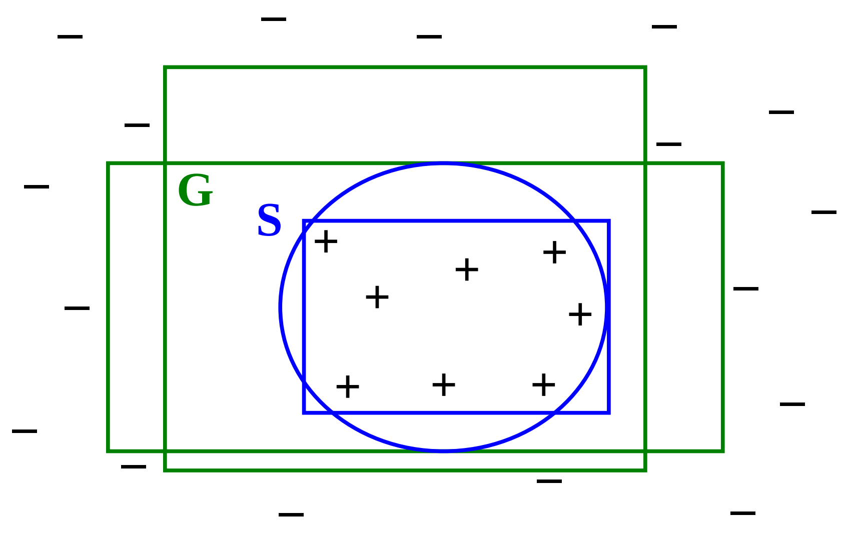
1. **Compact** –it is not explicitly storing every concept description in that space.
2. **Easy to update** – a new space simply corresponds to moving the boundaries.

With this convenient representation we can now apply machine learning techniques to it.

Version Spaces – Learning the Boundaries

A machine learning technique known as the ***candidate elimination algorithm*** can manipulate the boundaries in an extremely efficient manner.

This is best illustrated by thinking of a set of positive and negative training examples in some input space, and looking at where the decision boundaries can go:



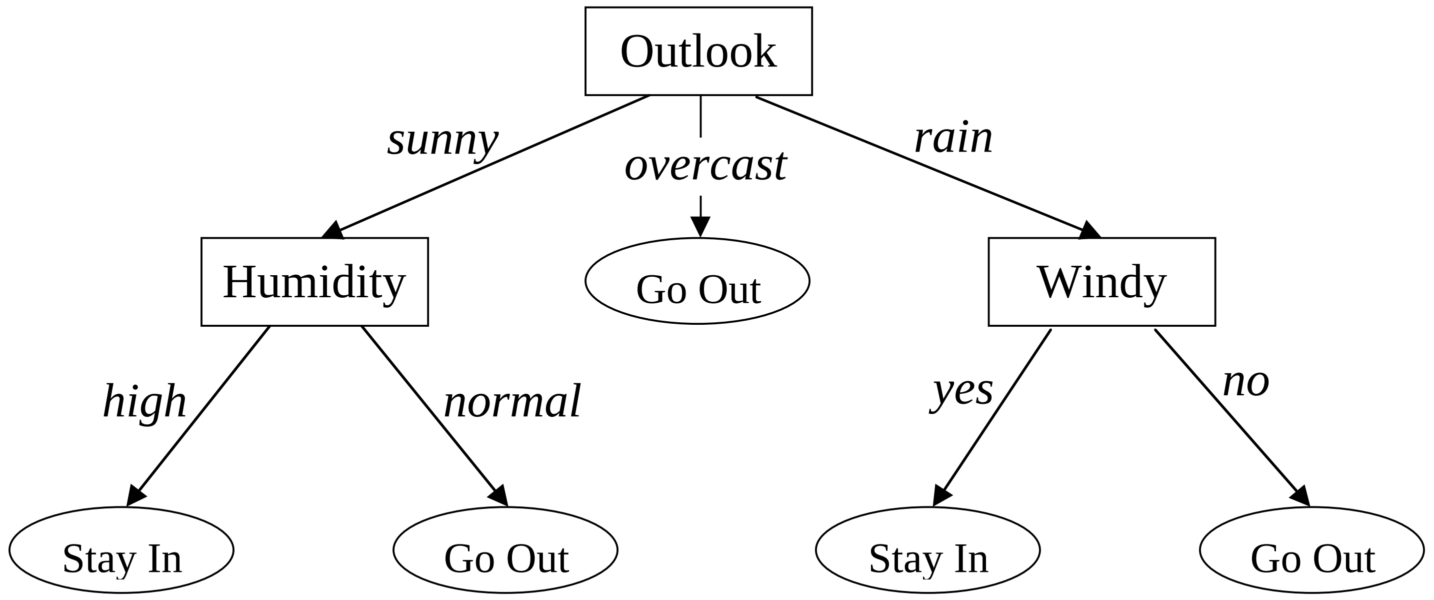
S = Most specific boundaries

G = Most general boundaries

It is easy to see how the boundaries can be refined as increasing numbers of data points become available, and how to extend the approach to more complex input spaces.

Decision Trees

Decision trees are a particularly convenient way of structuring information for classification systems. All the data to be classified enters at the root of the tree, while the leaf nodes represent the classifications. For example:



Intermediate nodes represent choice points, or tests upon attributes of the data, which serve to further sub-divide the data at that node.

Decision Trees versus Rules

Although ***decision trees*** look very different to ***rule based systems***, it is actually easy to convert a decision tree into a set of rules. From the above example we have:

R1:	IF:	Outlook = *overcast*
	THEN:	Go Out
R2:	IF:	Outlook = *sunny*
		Humidity = *normal*
	THEN:	Go Out
R3:	IF:	Outlook = *rain*
		Windy = *no*
	THEN:	Go Out
R4:	IF:	Outlook = *sunny*
		Humidity = *high*
	THEN:	Stay In
R5:	IF:	Outlook = *rain*
		Windy = *yes*
	THEN:	Stay In

The advantage of decision trees over rules is that comparatively simple algorithms can derive decision trees (from training data) that are good at generalizing (i.e. classifying unseen instances). Well known algorithms include CLS, ACLS, IND, ID3, and C4.5.

Decision Tree Algorithms - ID3, C4.5, Etc.

All ***decision tree algorithms*** set out to solve basically the same problem: Given a set of training data D, and a set of disjoint target classes $\{C_i\}$, the algorithm must use a series of tests T on data attributes with outcomes $\{O_i\}$ to partition D into subsets $\{D_i\}$ such that

$$D_i = \{ d \in D : T(d) = O_i \}$$

If we repeat this process for an appropriate sequence of tests T, we will end up with each resulting data subset D_i corresponding to a single class C_i, and we can draw the resultant decision tree, and if required, convert it to a set of rules.

The hard part is to determine the appropriate sequences of tests, and this is where the various decision tree algorithms differ. ID3 uses ideas from ***information theory*** and at each stage selects the test that gains the most information (or equivalently, results in the biggest reduction in entropy). C4.5 uses different ***heuristics*** which usually work better. Note that unlike the version space approach to concept learning, these algorithms are not incremental – if we get new data we need to start again.

Overview and Reading

1. We began by defining some general ideas about machine learning systems.
2. We first looked at learning in neural networks and evolutionary systems.
3. We then considered the need for learning in expert systems, and how we might set up simple rule induction systems and rule refinement strategies.
4. We then considered the version space approach to concept learning.
5. We ended by looking at decision trees, how they can be turned into rule sets, and how they can be generated by algorithms such as ID3 and C4.5.

Reading

1. Jackson: Chapter 20
2. Russell & Norvig: Chapters 18, 19, 20 & 21
3. Callan: Chapters 11 & 12
4. Rich & Knight: Chapter 17
5. Nilsson: Section 17.5