Learning with Distance

John Abela
Department of Computer Science
Faculty of Science
University of Malta
jabel@cs.um.edu.mt

ABSTRACT

The two main, competing, paradigms in Artificial Intelligence are the *numeric* (vector-space) and the *symbolic* approaches. The debate on which approach is the best for modelling intelligence has been called the 'central debate in AI'. ETS is an inductive learning model that unifies these two, competing, approaches to learning. ETS uses a distance function to define a class and also uses distance to direct the learning process. An ETS algorithm is applied to the Monk's Problems, a set of problems designed to evaluate the performance of modern learning algorithms - whether numeric and symbolic.

Keywords

Machine Learning, ETS, Monk's Problems

1. INTRODUCTION

Evolving Transformation System (ETS) is a new inductive learning model proposed by Goldfarb [3]. The main objective behind this learning model was the unification of the two major directions being pursued in Artificial Intelligence (AI), i.e. the *numeric* (or vector-space) and *symbolic* approaches. In Pattern Recognition (PR), analogously, the two main areas are the *decision-theoretic* and *syntactic/structural* approaches [2]. The debate on which of the two is the best approach to model intelligence has been going on for decades - in fact, it has been called the 'Central Debate' in AI [7]. It was McCulloch and Pitts who proposed simple neural models that manifested adaptive behaviour. Not much later, Newell and Simon proposed the *physical symbol systems* paradigm as a framework for developing intelligent agents. These two approaches more-or-less competed until Minsky and Papert published their now famous critique of the perceptron, exposing its limitations. This shifted attention, and perhaps more importantly funding, towards the symbolic approach until the 1980s when the discovery and the development of the *Error Back Propagation* algorithm together with the work of Rumelhart *et al* reignited interest in the connectionist approach.

One of the main ideas in the ETS model is that the concept of distance plays an important, even critical, role in the definition, specification, and learning of the class. This paper presents the results of applying an ETS learning algorithm to the task of using distance to learn the classes in the *Monk's Problems* [8], a set of problems designed for testing modern learning algorithms. The experiments on the Monk's Problems were carried out as part of the author's Ph.D. programme. A full exposition can be found in the author's Ph.D. thesis [1]. Unless otherwise stated, the definitions, tables and diagrams of this paper are reproduced from this work.

2. THE ETS MODEL

The main idea that characterizes the ETS model is that of using distance, i.e. metric or pre-metric function, for defining a class. Given a domain of discourse O, a class C in this domain can be specified by a non-empty finite subset of C which is called the set of *attractors*, and which we denote by A, and also by a distance function d_C. The set of all objects in O that belong to C is then defined to be:

$$\{o \in O \mid d_C(a, o) < \delta,\ a \in A\}.$$

In other words, the class consists precisely of those objects that are a distance of δ or less from some attractor. We illustrate with a simple example. Suppose we want to describe (i.e. specify) the class (or concept) *Cat*. Let O be the set of all animals, C a finite set of (prototypical) cats, δ as non-negative real number, and d_{Cat} a distance function defined on the set of all animals. Provided that C, δ, and d_{Cat} are chosen appropriately, the set of all cats is then taken to be the set of all animals that are a distance of δ or less from any of the attractors, i.e. the set of prototypical cats. This is depicted below in Figure 1. Here the set C contains just one prototypical cat. All animals that are in the δ-neighbourhood of this cat are classified at cats.

This idea borrows somewhat from the theory of concepts and categories in psychology. The reader is also referred to [6] for a discussion of Eleanor Rosch's theory of concept learning known as *Exemplar Theory*. Objects are classified together if they are, in some way, *similar*. In our example, all the animals that are cats are grouped together since the distance between any cat and the prototype is less than the threshold δ. In other words, an animal is a cat if it is similar

CSAW 2006 Dec 5-6, Kalkara, Malta

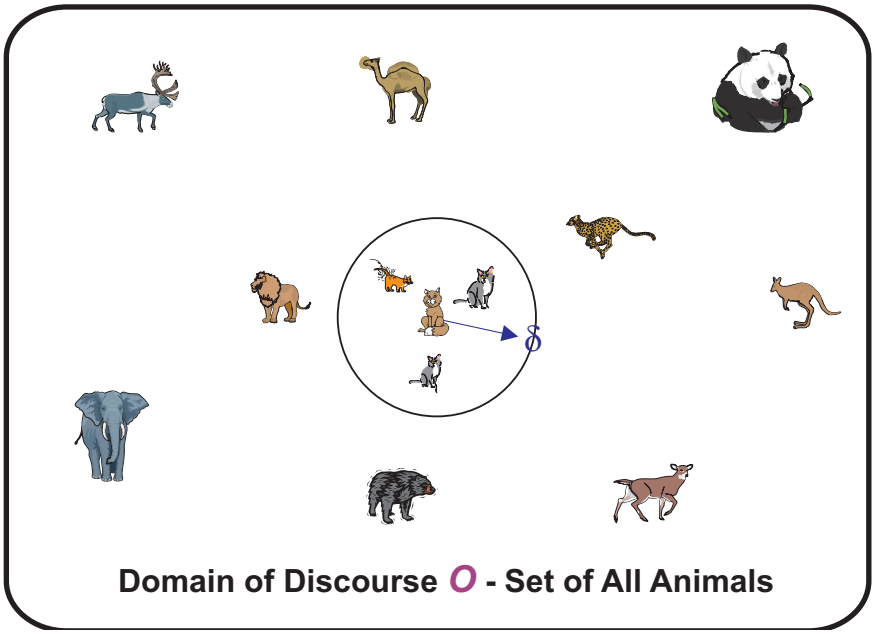

Figure 1: Class Description in the ETS Model.

to the cat prototype. The less distance there is between two animals, the more similar they are - i.e. distance is a measure of dissimilarity.

The ETS model is not just about class description, but also about learning class descriptions of classes from finite samples to obtain an *inductive class description*. Let O be a domain of discourse and let $\mathcal{C}$ be a, possibly infinite, set of related classes in O. Let C be a class in $\mathcal{C}$ and let C^+ be a finite subset of C and C^- be a finite subset of O whose members do not belong to C. We call C^+ the *positive* training set and C^- the *negative* training set. The learning problem is then to find, using C^+ and C^-, a class description for C. Of course, in practice, this might be, for all intents and purposes, impossible since if the number of classes in $\mathcal{C}$ is infinite, then C^+ may be a subset of infinitely many classes in $\mathcal{C}$. In other words, no finite subset, on its own, can characterize an infinite set. We therefore insist only on finding a class description for some class $C' \in \mathcal{C}$ such that C' *approximates* C. This depends, of course, on having a satisfactory definition of what it means for a class to approximate another.

In essence, learning in the ETS model involves finding a distance function that achieves *class separation*, i.e. a distance function such that the distance between objects in C^+ is zero or close to zero while the distance between an object in C^+ and an object in C^- is appropriately greater than zero.

An ETS algorithm achieves this by iteratively modifying a (parametrized) distance function such that the objects in C^+ start moving towards each other while, at the same time, ensuring that the distance from any object in C^+ to any object in C^- is always greater than some given threshold.

3. KERNEL LANGUAGES

Kernel Languages is an interesting subclass of regular languages. A kernel language over a finite alphabet Σ is specified by the pair $\langle K, F\rangle$ where $K \subset \Sigma^*$ is a finite, non-empty, set of strings called the set of *kernels* and $F \subset \Sigma^+$ is a finite, non-empty, and factor-free set of strings called the set of *features*. Informally, the strings in the kernel language specified by $\langle K, F\rangle$ are precisely those strings that can be obtained (generated) by inserting features from F anywhere, in any order, and any number of times, into the kernel strings of K. We only require that;

1. features are not inserted inside other features,
2. no feature is a factor[1] of any other feature, i.e. F is factor-free, and
3. no kernel contains a feature as a factor.

We illustrate with an example. Consider the set of kernels $K = \{bb, bc\}$ and the set of features $F = \{ab, ba\}$. The following strings in $L\langle K, F\rangle$, the language generated from K and F, are obtained by successive insertions of features in the kernels: (kernels letters are shown in bold)

bb	***bc***
*ba**bb***	***bc**ab*
*ba**bb**ab*	*ab**bc**ab*
*ba**b**ab**b**ab*	*ab**b**ba**c**ab*
*ba**b**abba**b**ab*	*ab**b**baba**c**ab*
*abba**b**abba**b**ab*	*baab**b**baba**c**ab*
*abba**b**abba**b**abab*	*baab**b**baba**c**abba*

[1]i.e. a substring

Note that features can be inserted anywhere in a kernel but not inside another feature. We must point out, however, that this does not necessarily mean that a string in $L\langle K, F\rangle$ cannot contain factors such as $aabb$ which can be formed by the insertion of the feature ab inside another occurrence of the same feature. The reason for this is because this feature can also be formed from the features ab and ba as follows: $\boldsymbol{bb} \rightarrow_{ab} ab\ \boldsymbol{bb} \rightarrow_{ab} baab\ \boldsymbol{bb}$ to obtain the string $baabbb$ which, of course, contains $aabb$ as a factor. Testing for membership in a kernel language is achieved either by checking if a given unknown string x can be generated from one of the kernels by a sequence of feature insertions or, alternatively, by (nondeterministically) deleting features from x to obtain a kernel. Note that the latter procedure is equivalent to computing the normal forms of x modulo the special semi-Thue system, R_F, that consists exactly of $|F|$ rules of the form (f, ε), $f \in F$. The set of rewrite rules of R_F is therefore indexed by F. To determine if x belongs to $L\langle K, F\rangle$ we then need only check whether one of the normal forms belongs to K.

There are various types of kernel languages. These include confluent and non-confluent kernel languages, trivial kernel languages, non-congruential kernel languages, and kernel languages with single or multiple kernels. It turns out that kernel languages have a number of real-world applications. The Monk's Problems instances can be encoded as strings from a kernel language. So can the parity problem and other interesting real-world problems [1].

4. THE MONK'S PROBLEMS

In the summer of 1991 at the 2^{nd} European Summer School on Machine Learning held at the Corsendonk Priory in Belgium, a number or researchers proposed a set of problems for testing the various machine learning algorithms that existed at the time. This set of problems was called 'The Monk's Problems' . The idea was that the main machine learning algorithms would be tested and compared using the same dataset.

The Monk's Problems are set in an artificial robot world where a robot can be described by six different attributes as follows (see Figure 2):

x_1 : head_shape	$\in$	{round, square, octagon}
x_2: body_shape	$\in$	{round, square, octagon}
x_3 : is_smiling	$\in$	{yes, no}
x_4: holding	$\in$	{sword, balloon, flag}
x_5: jacket_colour	$\in$	{red, yellow, green, blue}
x_6: has_tie	$\in$	{yes, no}

There were three problems in the set. Each problem was a binary classification task, i.e. that of determining whether or not a robot belongs to one of three classes. Each problem consists of a description of the class and a training set that is a proper subset of the 432 possible robots in the artificial world. The task of the machine learning algorithm is to generalize from the training examples and, if possible, to output a class description of each of the three classes. The three classes were:

1. **Monk1**:
   (head_shape = body_shape) or (jacket_colour = red)
   124 labelled examples were randomly selected from 432 possible robots. No misclassifications.

2. **Monk2**:
   exactly two of the six attributes have their *first* value
   169 labelled examples were randomly selected from 432 possible robots. No misclassifications.

3. **Monk3**:
   (jacket_colour is green and holding sword) or (jacket_colour is not blue and body_shape is not octagon)
   122 labelled examples were randomly selected from 432 possible robots. 5% misclassifications.

The only problem that contained noise was Problem 3. The intention here was to test the performance of the algorithms in the presence of noise. Problem 2 is very similar to parity problems. Problems 1 and 3 are in DNF form and are therefore assumed to be solvable by symbolic learning algorithms such as ID3, AQ, etc. Problem 2 combines attributes in a way which makes it awkward to express in DNF or CNF.

Table 1, reproduced from [8], lists the results obtained by the different learning algorithms on the Monk's Problems datasets. The experiments were performed by leading researchers in Machine Learning, each of whom was an advocate of the algorithm he or she tested and, in many cases, the creator of the algorithm itself.

The algorithms tested included the various decision tree learning algorithms such as ID3 and its variations, mFOIL - a rather interesting inductive learning system that learns Horn clauses using a beam search technique, and various neural networks such as Backpropagation and Cascade Correlation. No algorithm managed to correctly learn all three classes, although some came very close. In spite of the fact that the Monk's Problems are defined in a symbolic rather than a numeric domain, the best performing algorithms were, perhaps surprisingly, the neural networks.

5. THE *VALLETTA* ALGORITHM

The main objective of the *Valletta* algorithm is to investigate the feasibility or otherwise of applying the ETS model to a grammatical inference problem. The aim is to see if and how distance could be used to direct the learning process and also how such an algorithm would perform in the presence of noise. Valletta's learning strategy is based on the observation that the set of features that partially specify an unknown kernel language K must necessarily be a subset of the set of all repeated substrings in C^+ - assuming, of course, that the strings in C^+ were drawn at random from K and that every feature occurs at least twice in C^+. Valletta was designed from the beginning to learn multiple kernel languages and not just single kernel languages. This is because all examples of naturally occurring, i.e. real-world, kernel languages that we came across were all multiple-kernel. It turns out that learning multiple-kernel languages is much more difficult than learning single kernel languages. With

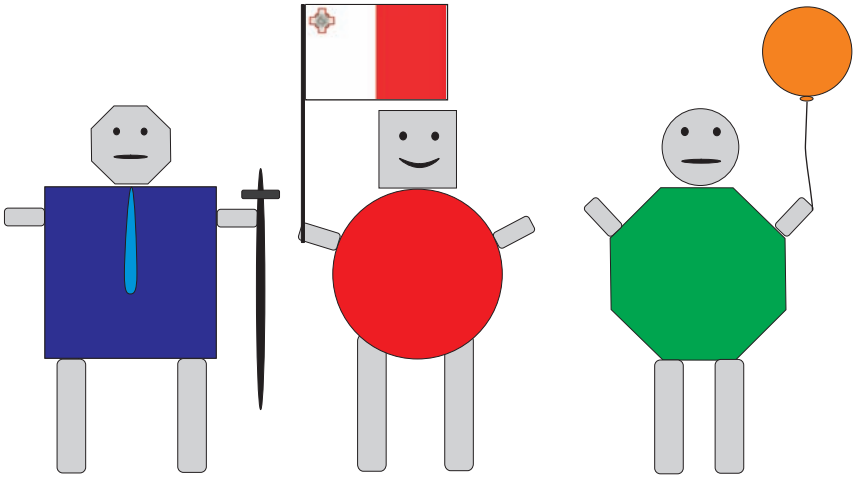

Figure 2: Some of the robots in the Monk's Problems.

Learning Algorithm	#1	#2	#3
AQ17-DCI	**100%**	**100%**	94.2%
AQ17-HCI	**100%**	93.1%	**100%**
AQ17-FCLS		92.6%	97.2%
AQ17-NT			**100%**
AQ17-GA	**100%**	86.8%	**100%**
Assistant Professional	**100%**	81.3%	**100%**
mFoil	**100%**	69.2%	**100%**
ID5R	81.7%	61.8%	
IDL	97.2%	66.2%	
ID5R-Hat	90.3%	65.7%	
TDIDT	75.7%	66.7%	
ID3	98.6%	67.9%	94.4%
ID3, no windowing	83.2%	69.1%	95.6%
ID5R	79.7%	69.2%	95.2%
AQR	95.9%	79.7%	87.0%
CN2	**100%**	69.0%	89.1%
CLASSWEB 0.10	71.8%	64.8%	80.8%
CLASSWEB 0.15	65.7%	61.6%	85.4%
CLASSWEB 0.20	63.0%	57.2%	75.2%
PRISM	86.3%	72.7%	90.3%
ECOWEB leaf prediction	71.8%	67.4%	68.2%
ECOWEB l.p. & information utility	82.7%	71.3%	68.0%
Backpropagation	**100%**	**100%**	93.1%
Backpropagation with weight decay	**100%**	**100%**	97.2%
Cascade Correlation	**100%**	**100%**	97.2%

Table 1: The published results of the Monk's Problems.

single-kernel languages one need only find a set of features. On the other hand, with multiple-kernel languages one also have to find the kernels without knowing beforehand the the number of kernels in the unknown language. Besides the obvious computational complexity this problem also poses an interesting question. Should one find a TS description that minimizes the number of features or the number of kernels? Valletta can be instructed to find TS descriptions that minimize either the number of features or the number of kernels. Valletta has what is called a *variable inductive preference bias*. This means that Valletta allows the user to choose which hypotheses (i.e. TS descriptions) are preferred over others. This is an important advantage over other learning algorithms.

Valletta has two main stages. The *pre-processing stage* searches for all repeated substrings in C^+ and stores them in a repeated substring list R_{C^+}. The *learning stage* then finds a set of features from R_{C^+} that gives class separation, i.e. a set of features that optimizes the function

$$f = \frac{f_1}{\epsilon + f_2},$$

where f_1 is the minimum EvD distance (over all pairs) between C^+ and C^-, f_2 is the average pair-wise *intra-set* EvD distance in C^+, and ϵ is a small positive real constant to avoid divide-by-zero errors. Valletta's learning stage builds a structure, the *search tree*, in which each node represents a feature set. Valletta expands this tree only on the basis of f_2. This means that Valletta's search for the set of features that describes the unknown kernel language K is completely directed by f_2. No other criteria are used to direct the learning process.

Valletta uses a new string-edit distance function called *Evoluntionary Distance* (*EvD*) [1]. EvD is suitable for describing kernel languages since it can detect features inserted inside other features. The idea behind EvD is that, given two strings and a set of features F, the distance between two strings can be taken to be the weighted Levensthein distance (WLD) between the normal forms (modulo R_F) of the two strings [1]. One important advantage of this technique is that normal forms are usually much shorter than the actual strings and this this results in significantly shorter computation times. The main problem is, of course, how to efficiently reduce the strings to their normal form modulo F. This was accomplished using a data structure called a *parse graph*. EvD works by first building the parse graphs for the two strings and then extracting the normal forms from the parse graphs. The EvD procedure then computes the weighted Levensthein distance between the normal forms and the set of kernels that is passed as a parameter.

An explanation of the inner workings of Valletta is beyond the scope of this paper. The reader is refered to [1] for a full exposition.

6. RESULTS AND CONCLUSIONS

The Monk's problems can quite easily be posed as GI [2] problems. In theory, every learning problem can be posed as a GI problem. In particular, each of the Monk's three classes of robots can be represented by a confluent kernel language. For the experiments, a special version of Valletta was developed. This is simply Valletta but with a much narrower inductive preference bias. The new version of Valletta was called Mdina, after Malta's old capital. Mdina considers only trivial kernel languages [1]. Mdina successfully learned problems 1 and 2 but did not learn problem 3. Investigation showed that this was because the training set was not *structurally complete* [4]. The addition of one string made the training set structurally complete and Mdina was then able to learn the class.

Mdina served to show that distance can indeed be used to direct the learning process. The experiments also highlighted the fact that learning algorithms converge to the correct class if the inductive bias of the algorithm is correct. In his technical report Thrun describes the Monk's problems and the results obtained by the various algorithms does not attempt to analyse or explain the results. We feel the whole exercise served more to determine whether each algorithm had the correct inductive bias to learn each of the Monk's problem than to determine the actual learning ability of the various algorithms. Each of the algorithms listed in the report have successfully been used for other learning tasks. We believe that the apparent inability of some of the algorithms to learn the Monk's problems is due more to their type of inductive bias rather than to anything else. Wolpert [10] and others have shown that no inductive bias can achieve a higher generalization accuracy than any other bias when when considered over all classes in a given domain. In spite of this, it has been documented that certain bias do perform better than average on many real-world problems [9]. This strongly suggests that many real-world problems are homogenous in nature in that they require very similar inductive biases. This explains why certain learning algorithms such as ID3 do well on most applications. When learning algorithms do badly it is very often a case of an incorrect inductive bias.

[2] Grammatical Inference

7. REFERENCES

[1] Abela, John. *ETS Learning of Kernel Languages.* Ph.D. Thesis, University of New Brunswick, Canada, 2002.

[2] Bunke, H. and Sanfeliu, A., (eds). *Syntactic and Structural Pattern Recognition - Theory and Applications.* World Scientific series in Computer Science, Vol. 7. 1990.

[3] Goldfarb, Lev. *On the Foundations of Intelligent Processes - 1: An Evolving Model for Pattern Learning.* Pattern Recognition, 23, pp. 595-616, 1990.

[4] Michalski, R., Carbonel, J., Mitchell, T., (eds). *Machine Learning - An Artificial Intelligence Approach.* Morgan Kaufmann Publishers Inc. 1983.

[5] Nigam, Sandeep, *Metric Model Based Generalization and The Generalization Capabilities of Connectionist Models.* Masters Thesis, Faculty of Computer Science, University of New Brunswick, Canada. 1992.

[6] Rosch, Eleanor, H., *On the Internal of Perceptual and Semantic Categories.* in Timothy E. Morre, ed., *Cognitive Development and the Acquisition of Language*, Academic Press. 1973.

[7] J. Stender, and T. Addis (eds). *Symbols vs Neurons.* IOS Press, Amsterdam, 1990.

[8] S. Thrun et al. *The Monk's Problems: A Performance Comparision of Different Learning Algorithms.* Carnegie Mellon University CMU-CS-91-197, December 1991.

[9] Thornton, Chris *There is No Free Lunch but the Starter is Cheap: Generalisation from First Principles* Cognitive and Computing Sciences, University of Sussex, Brighton, UK, 1999.

[10] Wolpert, D., and Macready, W. *No Free Lunch Theorems for Search*, Unpublished MS, 1995.