K*: An Instance-based Learner Using an Entropic Distance Measure

John G. Cleary,

Leonard E. Trigg,

Dept. of Computer Science,

University of Waikato,

New Zealand.

e-mail:{jcleary,trigg}@waikato.ac.nz

Abstract

The use of entropy as a distance measure has several benefits. Amongst other things it provides a consistent approach to handling of symbolic attributes, real valued attributes and missing values. The approach of taking all possible transformation paths is discussed. We describe K*, an instance-based learner which uses such a measure, and results are presented which compare favourably with several machine learning algorithms.

Introduction

The task of classifying objects is one to which researchers in artificial intelligence have devoted much time and effort. The classification problem is hard because often the data available may be noisy or have irrelevant attributes, there may be few examples to learn from or simply because the domain is inherently difficult. Many different approaches have been tried with varying success. Some well known schemes and their representations include: ID3 which uses decision trees (Quinlan 1986), FOIL which uses rules (Quinlan 1990), PROTOS which is a case-based classifier (Porter, Bareiss, & Holte 1990), and the instance-based learners IB1-IB5 (Aha, Kibler, & Albert 1991; Aha 1992). These schemes have demonstrated excellent classification accuracy over a large range of domains.

What many instance-based algorithms lack is an integrated theoretical background. Often schemes which are primarily designed to handle symbolic attributes have difficulty in domains where features may be real valued. Many schemes which handle real feature values are extended to cope with symbolic attributes in an ad-hoc manner. A similar problem is apparent in how classifiers handle missing values. Common approaches are to treat missing values as a separate value, treat them as maximally different, replace them with the average value, or to simply ignore them. In this paper we discuss the use of entropy as a distance measure which provides a unified approach to dealing with these problems. We also present K*, an instance-based learner which uses such a measure and examine its performance on a range of problems.

Instance-Based Learners

Instance-based learners classify an instance by comparing it to a database of pre-classified examples. The fundamental assumption is that similar instances will have similar classifications. The question lies in how to define “similar instance” and “similar classification”. The corresponding components of an instance-based learner are the distance function which determines how similar two instances are, and the classification function which specifies how instance similarities yield a final classification for the new instance. In addition to these two components, IBL algorithms have a concept description updater which determines whether new instances should be added to the instance database and which instances from the database should be used in classification. For simple IBL algorithms, after an instance has been classified it is always moved to the instance database along with the correct classification. More complex algorithms may filter which instances are added to the instance database to reduce storage requirements and improve tolerance to noisy data.

Nearest neighbour algorithms (Cover & Hart 1967) are the simplest of instance-based learners. They use some domain specific distance function to retrieve the single most similar instance from the training set. The classification of the retrieved instance is given as the classification for the new instance. Edited nearest neighbour algorithms (Hart 1968; Gates 1972) are selective in which instances are stored in the database and used in classification. *k*-nearest neighbour algorithms are only slightly more complex. The *k* nearest neighbours of the new instance are retrieved and whichever class is predominant amongst them is given as the new instance's classification. A standard nearest neighbour classification is the same as a *k*-nearest neighbour classifier for which *k*=1.

Aha, Kibler & Albert (1991) describe three instance-based learners of increasing sophistication. IB1 is an implementation of a nearest neighbour algorithm with a specific distance function. Real valued attributes are normalised to a common scale so all attributes have equal weighting, and missing values are assumed to be maximally different to the present value. IB2 contains extensions to reduce storage requirements; only misclassified instances are saved. IB3 is a further extension to improve tolerance to noisy data; instances that have a sufficiently bad classification history are forgotten, only instances that have a good classification history are used for classification. Aha (1992) described IB4 and IB5, which handle irrelevant and novel attributes.

Cost & Salzberg (1993) use a modification of Stanfill & Waltz's (1986) value difference metric in conjunction with an instance weighting scheme in their system PEBLS. This scheme was designed for classifying objects where feature values are symbolic. Numeric distances between symbolic values are calculated based on the degree to which the relative frequencies of the symbols across classifications remains the same. Instances in memory are also weighted according to their past classification accuracy.

Some Desirable Properties for a Classification Algorithm

A good classification algorithm should exhibit the following characteristics:

The surface defined by a similarity measure over the hyperspace of predictor attributes must be smooth. In his work on evaluation functions for computer backgammon, Berliner (1980) argues that the surface defined by a good evaluation function will be smooth—if there are ridges or discontinuities in the surface, the program utilising the evaluation function may make decisions to its detriment. As it applies to instance-based learning, there should not be large jumps in the distance measure for small changes in the instance database or the test instance.

There should be a natural way of adjusting the relevances of predictor attributes.

There should be an intuitive way of dealing with partial data. The datasets for many real world domains contain missing values, and to ignore instances containing missing values is to throw away useful information contained in the other values.

A good algorithm should not be restricted to prediction of symbolic values—it should be able to predict real values as well.

Entropy as a Distance Measure

The approach we take here to computing the distance between two instances is motivated by information theory. The intuition is that the distance between instances be defined as the complexity of transforming one instance into another. The calculation of the complexity is done in two steps. First a finite set of transformations which map instances to instances is defined. A "program" to transform one instance (*a*) to another (*b*) is a finite sequence of transformations starting at *a* and terminating at *b*.

Following the usual development of complexity theory such programs (sequences) are made "prefix free" by appending a termination symbol to each string. The usual definition of the (Kolmogorov) complexity of a program is the length of the shortest string representing the program (Li & Vitanyi 1993). Using this approach a Kolmogorov distance between two instances can be defined as the length of the shortest string connecting the two instances. This approach focuses on a single transformation (the shortest one), out of many possible transformations. The result is a distance measure which is very sensitive to small changes in the instance space and which does not solve the smoothness problem well. The K* distance defined below tries to deal with this problem by summing over all possible transformations between two instances.

It is not entirely clear what it is that should be summed over when doing this. Adding the lengths of the different transformations clearly is not the correct thing to do. The key point here is to note that it is possible to associate a probability with each sequence. If the complexity (length) of program measured in bits is c then the appropriate probability is 2^{-c}. In particular, it is true that in any well defined distance based on Kolmogorov complexity the sum of this probability over all transformations will satisfy the Kraft inequality: $\sum 2^{-c} \leq 1$. One way of interpreting this is that it is the probability that a program will be generated by a random selection of transformations. In terms of the distance between instances it is the probability that an instance will be arrived at by doing a random walk away from the original instance. After summing over all paths this probability can be transformed into units of complexity by taking the logarithm.

This approach of summing over all possible transforming paths has been used successfully in Yee & Allison's (1993) r-theory which measures the distance between sequences of DNA. There is empirical evidence that using all mutational transformations between two strings rather than the single shortest path gives a more robust and realistic measure of the relatedness of two DNA sequences.

Specification of K*

Let **I** be a (possibly infinite) set of instances and **T** a finite set of transformations on **I**. Each $t \in$ **T** maps instances to instances: t:**I**→**I**. **T** contains a distinguished member σ (the stop symbol) which for completeness maps instances to themselves ($\sigma(a)=a$). Let **P** be the set of all prefix codes from **T*** which are terminated by σ. Members of **T*** (and so of **P**) uniquely define a transformation on **I**:

$$\bar{t}(a) = t_n(t_{n-1}(\ldots t_1(a)\ldots)) \text{ where } \bar{t} = t_1,\ldots t_n$$

A probability function p is defined on **T***. It satisfies the following properties:

$$0 \le \frac{p(\bar{t}u)}{p(\bar{t})} \le 1$$

$$\sum_u p(\bar{t}u) = p(\bar{t})$$

$$p(\Lambda) = 1$$

As a consequence it satisfies the following:

$$\sum_{\bar{t} \in P} p(\bar{t}) = 1$$

The probability function P* is defined as the probability of all paths from instance a to instance b:

$$P^*(b|a) = \sum_{\bar{t} \in P : \bar{t}(a) = b} p(\bar{t})$$

It is easily proven that P* satisfies the following properties:

$$\sum_b \mathrm{P}^*(b|a) = 1$$

$$0 \le \mathrm{P}^*(b|a) \le 1$$

The K* function is then defined as:

$$\mathrm{K}^*(b|a) = -\log_2 \mathrm{P}^*(b|a)$$

K* is not strictly a distance function. For example, $\mathrm{K}^*(a|a)$ is in general non-zero and the function (as emphasised by the | notation) is not symmetric. Although possibly counter-intuitive the lack of these properties does not interfere with the development of the K* algorithm below. The following properties are provable:

$$\mathrm{K}^*(b|a) \ge 0$$

$$\mathrm{K}^*(c|b) + \mathrm{K}^*(b|a) \ge \mathrm{K}^*(c|a)$$

Real Numbers

As the first example of how to use this theory we compute a distance between real numbers. This is done in two steps. First a distance function between integers is developed and then this is modified to deal with real numbers.

Let the set of instances **I** be the integers (positive and negative). There are three transformations in the set **T**: σ the end of string marker; and **left** and **right** which respectively add 1 and subtract one. The probability of a string of transformations is determined by the product of the probability of the individual transformations:

$$p(\bar{\mathrm{t}}) = \prod_i p(\mathrm{t}_i) \text{ where } \bar{\mathrm{t}} = \mathrm{t}_1,\ldots\mathrm{t}_n$$

The probability of the stop symbol σ is set to the (arbitrary) value *s* and *p*(**left**) = *p*(**right**) = (1-*s*)/2.

It can be shown (after significant effort for which we do not have space here) that $\mathrm{P}^*(b|a)$ depends only on the absolute difference between *a* and *b*, so abusing the notation for P* slightly we can write:

$$\mathrm{P}^*(b|a) = \mathrm{P}^*(i) = \frac{s}{\sqrt{2s - s^2}}\left(\frac{1-\sqrt{2s-s^2}}{1-s}\right)^i \text{ where } i = |a - b|$$

and

$$\mathrm{K}^*(b|a) = \mathrm{K}^*(i) =$$
$$\tfrac{1}{2}\log_2(2s - s^2) - \log_2(s) + i[\log_2(1-s) - \log_2(1-\sqrt{2s-s^2})]$$

That is, the distance is proportional to the absolute difference between two instances.

Note that the set of transformations chosen above is not determined by the theory. Other transformation sets (with different distance functions) could be envisaged but this formulation seems to capture the idea that all points are equivalent and that space is "invariant" under left and right shifts as well as giving results that work well in practice.

To reformulate this for real numbers the assumption is made that the real space is underlain by a discrete space with the distance between the discrete instances being very small. The first step is to take the expressions above in the limit as s approaches 0 (that is the transformation strings will be very long). This gives:

$$\mathrm{P}^*(i) = \sqrt{s/2}\,\mathrm{e}^{-i\sqrt{2s}}$$

This can then be reformulated as a probability density function where the probability that an integer between i and $i + \Delta i$ will be generated is:

$$\mathrm{P}^*(i) = \sqrt{s/2}\,\mathrm{e}^{-i\sqrt{2s}}\,\Delta i$$

This can then be rescaled in terms of a real value x where $\dfrac{x}{x_0} = i\sqrt{2s}$. This results in the probability density function P* over the reals:

$$\mathrm{P}^*(x) = \frac{1}{2x_0}\mathrm{e}^{-x/x_0}\,dx$$

In this formulation x_0 functions as a scale length, for example, it is the mean expected value for x over the distribution P*. For different applications it is necessary to choose a reasonable value for

x_0. There are some rough guidelines about how to do this. For example, if the instances have a measurement error then x_0 should probably not be smaller than the standard deviation of the errors on the measurement. The next section specifies a technique to choose x_0 values.

Symbolic Probabilities

One of the advantages of the approach we are following here is that both real attributes and symbolic attributes can be dealt with together within the same framework. To deal with symbolic attributes consider a set of *n* instance values which occur with probabilities p_i, $1 \le i \le n$. The transformations that we allow on the instances are the transmutation of any instance to any other instance. The probability of a symbol staying the same (the end of string transformation) is taken to be the probability *s* and the probability of a transformation to a symbol *j* to be $(1-s)p_j$. Summing over all possible transformations gives:

$$\mathrm{P}^*(j|i) = \begin{cases} (1-s)p_j & \text{if } i \neq j \\ s + (1-s)p_i & \text{if } i = j \end{cases}$$

Note that the probability *s* here is analogous to the probability *s* (and the equivalent x_0) in the development above for real numbers. That is, some reasonable value must be chosen for *s* depending on the data being modelled.

Combining Attributes

To compute a distance between instances with more than one attribute is straightforward. The set of transformations on the combined attributes can be taken as the union of the transformations for the individual attributes. The transformation strings can then be modelled by sequentially transforming the first attribute, then the second attribute and so on until all the attributes are transformed. The result is that the probability for the total string is the product of the probabilities of the individual strings and thus the distance function is the sum of the distances for the individual attributes.

This simple additive approach is the one taken for all the results given below. It is worth noting, however, that it is not the only possible way of combining attributes. For example, if an instance is modelled as a two dimensional space the set of transformations are naturally **left**, **right**, **up** and **down**. Summing over all such transformations the resulting distance is $\sqrt{d_1^2 + d_2^2}$ rather than the simple additive $d_1 + d_2$ (where d_1 and d_2 are the individual distances on the two co-ordinates).

Missing Values

One issue that must be dealt with in many datasets is instances where one or more of the attributes are missing. The approaches in the literature vary widely on how to deal with this problem. In some cases the distance to the missing attribute is taken to be the maximum possible in others the entire instance is ignored.

If the values that are missing are in an instance which is being classified the attributes can simply be ignored and the predictions made on just the remaining attributes.

The more interesting case is when the missing values are in instances stored in the database. The way we have chosen to deal with this, is to assume that the missing values can be treated as if they were drawn at random from among the instances in the database. This is easily fitted into the probability based distance by setting the probability of transforming to the missing value as the mean of the probability of transforming to each of the (specified) attribute values in the data base, that is:

$$\mathrm{P}^*(?|a) = \sum_b \frac{\mathrm{P}^*(b|a)}{N}$$

where the sum is over all specified instances in the database and N is the number of such instances. The effective distance to a missing value is then (roughly) the expected distance to a random instance of that attribute.

K* Algorithm

For the implementation of an instance-based classifier which uses the entropic distance measure described above we need a way of selecting values for the parameters x_0 and s, and a way of using the values returned by the distance measure to give a prediction.

<u>Choosing values for the arbitrary parameters</u>

For each dimension we have to choose values for the parameters x_0 (for real attributes) and s (for symbolic attributes). The behaviour of the distance measure as these parameters change is interesting. Consider the probability function for symbolic attributes as s changes. With a value of s close to 1, instances with a symbol different to the current one will have a very low transformation probability, while instances with the same symbol will have a high transformation probability. Thus the distance function will exhibit nearest neighbour behaviour. As s approaches 0, the transformation probability directly reflects the probability distribution of the symbols, thus favouring symbols which occur more frequently. This behaviour is similar to the default rule for many learning schemes which is simply to take whichever classification is most likely (regardless of the new instance's attribute values). As s changes, the behaviour of the function varies smoothly between these two extremes. The distance measure for real valued attributes exhibits the same properties. Thus when x_0 is small the probability to instances drops very quickly with increasing distance thus functioning like a nearest neighbour measure. On the other hand if x_0 is very large almost all the instances will have the same transformation and will be weighted equally.

In both these cases we can think of the number of instances that are included within the probability distribution as varying from an extreme of 1 (when the distribution is nearest neighbour) to the other extreme of N when all instances are weighted equally. (If there is more than one nearest neighbour then the minimum will be greater than 1.)

The effective number of instances can be computed for any function P* using the following expression:

$$n_0 \le \frac{\left(\sum_b \mathrm{P}^*(b|a)\right)^2}{\sum_b \mathrm{P}^*(b|a)^2} \le N$$

where N is the total number of training instances and n_0 is the number of training instances at the smallest distance from a (for this attribute). The K* algorithm then chooses a value for x_0 (or s) by selecting a number between n_0 and N and inverting the expression above. Thus selecting n_0 gives a nearest neighbour algorithm and choosing N gives equally weighted instances. For convenience the number is specified by using the "blending parameter" b, which varies from b= 0% (for n_0) and b=100% for N, with intermediate values interpolated linearly.

We think of the selected number as a "sphere of influence", specifying how many of the neighbours of a should be considered important (although there is not a harsh cut off at the edge of the sphere—more of a gradual decreasing in importance).

To compute x_0 we use an iterative root finder, which is slow, but the results are cached so that whenever that instance value reappears the precalculated parameters can be used. In the symbolic cases s is simply set directly proportional to b.

The x_0 (or s) parameters are set for each dimension independently, but using the same blend parameter which gives equal weighting to each attribute. We can then use the combined attribute distance measure to compute the size of the final sphere of influence. This is usually much smaller than the size specified at the single attribute level.

Category prediction

We calculate the probability of an instance a being in category C by summing the probabilities from a to each instance that is a member of C.

$$\mathrm{P}^*(C|a) = \sum_{b \in C} \mathrm{P}^*(b|a)$$

The probabilities for each category are calculated. The relative probabilities obtained give an estimate of the category distribution at the point of the instance space represented by a. Most other techniques return a single category as the result of classification. For ease of comparison here we choose the category with the highest probability as the classification of the new instance. Alternatives to this include choosing a class at random using the relative probabilities or returning a normalised probability distribution as the answer.

Results

To get an idea for how well the K* algorithm performs in practise, classification has been carried out on several datasets commonly used in the machine learning literature. The datasets are the same as those used by Holte (1993), and were originally taken from the UCI Machine Learning Database Repository.

The datasets were partitioned into 2/3 training, 1/3 test. 25 different partitions were made for each dataset. The schemes were run on all 25 partitions and the results averaged.

The fraction of correct classifications are presented in Table 1. The best result(s) for each dataset are highlighted in bold face. All schemes were run using default settings. C4.5 results were obtained for both pruned trees and rules. The default blend setting for K* is 20%, a value which seems to work well for most datasets. Results were also obtained for the blend setting which gave the best accuracy for each dataset.

	C4.5	C4.5	FOIL	1R	IB1	K*	K*
Dataset	P-Tree	Rules				b=20	b=best
BC	**70.7**	68.8	54.3	67.5	66.1	68.6	**70.8**
CH	**99.2**	**99.2**	29.3	64.9	89.6	93.2	93.3
GL	66.0	64.8	50.0	52.1	67.8	**72.4**	**73.9**
G2	72.9	74.2	64.4	69.0	76.4	**82.3**	**82.7**
HD	75.7	**77.6**	64.2	73.8	75.5	75.0	**82.2**
HE	68.7	79.5	66.6	78.4	**80.8**	80.4	**83.8**
HO	76.1	**81.7**	62.5	**81.7**	77.4	76.2	79.2
HY	91.3	**99.2**	98.2	97.8	97.7	98.5	98.6
IR	94.3	94.3	89.8	92.3	**95.3**	94.9	**95.3**
LA	72.2	84.2	65.3	76.4	84.2	**90.9**	**92.0**
LY	74.8	75.8	66.2	72.7	80.9	**82.2**	**82.6**
SE	75.4	**97.8**	95.8	95.1	93.8	95.2	95.7
SO			96.3	79.2	**99.8**	**99.8**	**99.8**
VO	91.9	94.8	87.6	**95.4**	91.9	93.0	93.2
V1	83.4	89.8	77.4	87.3	87.3	**90.5**	**90.5**

Table 1. Classification accuracy for several datasets.

As can be seen K* performs well across a wide range of schemes. In almost all cases it was better than the other instance-based learner (IB1) and in 6 of the 15 cases it gave the best results out of all the schemes.

Allowing the blend parameter to vary gives better results and we are currently investigating ways to set it more intelligently, possibly varying it for different attributes.

It is interesting to note that the 1R algorithm (which classifies based on a single attribute) performs very well on some datasets, indicating that varying the importance of individual attributes could further improve K*'s performance.

Conclusions

An instance based learning scheme has been presented. On real datasets it performs well against a range of both rule based and instance based learning schemes. The underlying technique used of summing probabilities over all possible paths solves the smoothness problem and we believe contributes strongly to its good overall performance. The underlying theory also allows clean integration of both symbolic and real valued attributes and a principled way of dealing with missing values.

The implementation and results here are a first implementation and we want to further explore refinements to the basic algorithm and extensions to the range of its applicability. For example, it is possible to use K* to predict real valued attributes. We are also exploring applications to more complex domains such as the similarity of 2-d images such as weather maps.

On two data files the simple learning algorithm 1R (which does its classification based on only a single attribute) gets better performance K*. It seems then that lowering the blend parameter for important attributes and raising it for unimportant ones might pay off in these cases.

Acknowledgments

This work was done as part of the WEKA project, at the University of Waikato. We would like to thank the members of the WEKA team, in particular Steve Garner for supplying the test datasets and for doing many computer runs.

References

Aha, D.W., Kibler, D. & Albert, M. K. (1991) "Instance-based Learning Algorithms." Machine Learning 6, pp. 37-66.

Aha, D.W. (1992) “Tolerating Noisy, Irrelevant and Novel Attributes in Instance-based Learning Algorithms.” International Journal of Man Machine Studies 36, pp. 267-287.

Berliner, H. (1980) "Backgammon Computer Program Beats World Champion." Artificial Intelligence 14, pp. 205-220.

Cost, S. & Salzberg, S. (1993) "A Weighted Nearest Neighbour Algorithm for Learning with Symbolic Features." Machine Learning 10, pp. 57-78.

Cover, T.T. & Hart, P.E. (1967) "Nearest Neighbour Pattern Classification." IEEE Transactions on Information Theory 13, pp. 21-27.

Gates, G.W. (1972) "The Reduced Nearest Neighbour Rule." IEEE Transactions on Information Theory 18, pp. 431-433.

Hart, P.E. (1968) "The Condensed Nearest Neighbour Rule." IEEE Transactions on Information Theory 14, pp. 515-516.

Holte, R.C. (1993) “Very Simple Classification Rules Perform Well on Most Commonly Used Datasets.” Machine Learning 11, pp. 63-91.

Li, M., and Vitanyi, P. (1993) *An Introduction to Kolmogorov Complexity and its Applications*, Springer-Verlag, New York.

Porter, B.W., Bareiss, R. & Holte, R.C. (1990) "Concept Learning and Heuristic Classification in Weak-theory Domains." Artificial Intelligence 45, pp. 229-263.

Quinlan, J.R. (1986) "Induction of Decision Trees." Machine Learning 1, pp. 81-106.

Quinlan, J.R. (1990) "Learning Logical Definitions from Relations." Machine Learning 5, pp 239-266.

Stanfill, C. & Waltz, D. (1986) "Toward Memory-based Reasoning." Communications of the ACM 29, 12, pp. 1213-1228.

Yee, C.C. & Allison, L. (1993) "Reconstruction of Strings Past." Computer Applications in the Biosciences 9, 1, pp. 1-7.