Flexible Decision Trees in a General Data-Mining Environment

Joshua W. Comley, Lloyd Allison, and Leigh J. Fitzgibbon

School of Computer Science and Software Engineering
Monash University, Clayton 3800, Australia
{joshc,lloyd,leighf}@bruce.csse.monash.edu.au

Abstract. We describe a new data-mining platform, CDMS, aimed at the streamlined development, comparison and application of machine learning tools. We discuss its type system, focussing on the treatment of statistical models as first-class values.
This allows rapid construction of composite models - complex models built from simpler ones - such as mixture models, Bayesian networks and decision trees. We illustrate this with a flexible decision tree tool for CDMS which rather than being limited to discrete target attributes, can model any kind of data using arbitrary probability distributions.

1 Introduction

This paper introduces the 'Core Data-Mining Software' (CDMS) system and a flexible decision tree inference program implemented as a CDMS 'plug-in'. CDMS is a general data-mining platform providing a tool-box of common operations such as data input and output, manipulation, and visualization. CDMS is being developed at Monash University, and although only in its early stages is already proving to be a useful environment for the streamlined implementation of a variety of machine learning and statistical analysis tools.

Many other machine learning and data-mining platforms exist (e.g. S-plus [7], R [5], and Weka [12]), but the unique handling and definition of statistical models by CDMS makes it a particularly interesting and powerful system. Section 2 describes CDMS and discusses its representation of values and data types. In section 3 we examine the characteristics of a CDMS model.

A statistical model is treated as a value by CDMS, meaning that it may be the subject or result of functions, or even a parameter for another model. By including models as parameters for other models, we can build powerful composite models. We illustrate this with a rather general decision tree 'plug-in', described in section 4, and give examples in section 5.

2 The Core Data Mining Software (CDMS) Platform

CDMS provides a framework in which to implement data-mining and machine learning tools, referred to in this paper as 'plug-ins'. It has a library of inbuilt functions, models and other values which plug-ins can utilize and add to. CDMS is capable of performing input/output of data in various file formats, including standard delimited text files, Weka [12] '.arff' files, GenBank files and C5's [6] '.data' and '.names' files. It is implemented in Java and offers an intuitive graphical user interface. It is currently under development, but it is anticipated that a prototype will be available soon.

Data is represented in CDMS as values. CDMS defines various types of values using a class hierarchy. This includes simple types such as string, scalar - which has the sub-types of discrete and continuous - and 'Triv' (the null type of CDMS). In addition, the structured and vector types provide support for heterogeneous and homogeneous collections of values respectively. Functions are also treated as values in CDMS. They are characterized by a parameter type and result type. A function can be applied to any value which matches the parameter type, giving a new value which will match the result type. Because functions are values, one could build a vector of them, or even write a function which can be applied to a function, giving another function as a result.

3 CDMS Models

This section discusses another type of CDMS value - the model, which we define here to be a family of distributions (e.g. normal distributions). When predicting or generating data, or when obtaining a likelihood, the distribution to be used is determined by the parameters supplied to the model. For a discussion of closely related schemes for model types and operations, see [1], and [4].

Models are characterized by the kind of values they work with. This is broken down into the parameter space, data space, input space, and sufficient space.

The parameter space refers to the type of parameters the model requires. For example, a Gaussian model requires the mean and variance as parameters - which can be represented in CDMS using a structure of two continuous values.

The data space[1] is the type of data that is modelled - i.e.the data over which the model can give a probability density. This density is typically a function of the model's parameters, and often of some 'input' data as well.

The input space is the type of any 'input' or 'explanatory' data that the model requires (in addition to its parameters) to provide a probability of the observed target data. Examples include the explanatory attributes of a decision tree, or the 'independent' attributes in a polynomial regression.

The sufficient space refers to the type of data that can capture all the information needed by the model about a vector of input data, and a corresponding vector of output data. Such data are commonly referred to as a 'sufficient statistic'. For a more detailed discussion of sufficient statistics in relation to models, data types and efficient computations, see [4].

Any CDMS model should be able to perform the following operations: log-probability, prediction, generation, and 'get-sufficient'.

[1] referred to in this paper also as the 'target data' type, or 'output data' type

The 'log-probability' operation requires a model to compute the log-probability of some given output data, when also supplied with parameters and a vector of input data. Models are also required to perform this operation when given sufficient statistics in place of the vector of input and output data.

The prediction operation involves the model selecting the 'most likely' output vector, when presented with parameters and a corresponding input vector. Generation is similar to the prediction operation, but instead of returning the most likely output value, the model pseudo-randomly selects an output value by sampling from the predictive distribution.

When presented with a vector of input and output data, the 'get-sufficient' operation requires models to return the corresponding sufficient statistic.

4 Decision Trees

Decision trees model the correlation between the input and output attributes in order to predict likely output values for future 'test' data-sets where only the input values are known. Decision tree tools such as C5 [6], CART [3], and DTree [11] generally do this well, but are often only applicable to problems where the output attribute takes discrete values. Furthermore, rather than modelling the output attribute probabilistically some decision trees simply state the predicted value for each test case. Here we are concerned with a more general situation where the output attribute is not necessarily discrete, and where we wish to model it with a probability distribution.

The tree plugin has been designed to accept any CDMS model in the leaves, allowing us to use arbitrary distributions to fit the output data. A tree class is constructed from modular elements (described below) which are largely independent, making it easy to assemble decision tree classes to suit a range of data sets.

The **leaf model** of a decision tree class defines how the target attribute is to be modelled. Each leaf is likely to contain different parameters for this model, which are estimated using the tree's **leaf estimator function**, based on the training data pertaining to the leaf. Examples of simpler leaf models include Gaussian and multi-state distributions, but any CDMS model could be used. By choosing a leaf model with a (non-Triv) input space, e.g. a polynomial regression, interesting tree classes may be constructed, whose leaves model the target attribute as a function of one or more input attributes.

A tree class has a **branch method** for each attribute. This defines how the attribute will be tested at a branch node. Possible branch methods include 'hard' or 'soft' cut points for continuous attributes, 'pie-slice' cuts for cyclic attributes, or n-way branches for discrete n-valued attributes. The tree tool can accept any branch method so long as it can provide a fractional assignment over the branch's children when given the attribute to test.

The **cost function** is used by a search to cost each candidate tree. It is given a structure comprising the tree model, its parameters, and the data, and returns a penalty value which the **search function**[2] attempts to minimise.

It is interesting to note that the decision tree class, as well as using CDMS models in its leaves, can itself be seen as a CDMS model. The data space of the decision tree model is simply the target attribute, while the input space is a structured value of the explanatory attributes. The parameters of a decision tree model would include the topology of the tree, the input attribute to be tested at each branch node and any parameters pertaining to these tests, and the parameters for the model in each leaf. The search function, leaf estimator function, and tree costing function are not actually part of the decision tree model. Rather, they constitute an estimator function for it. Given the training data and this estimator function, one can obtain an estimate of the optimal parameters for the decision tree model.

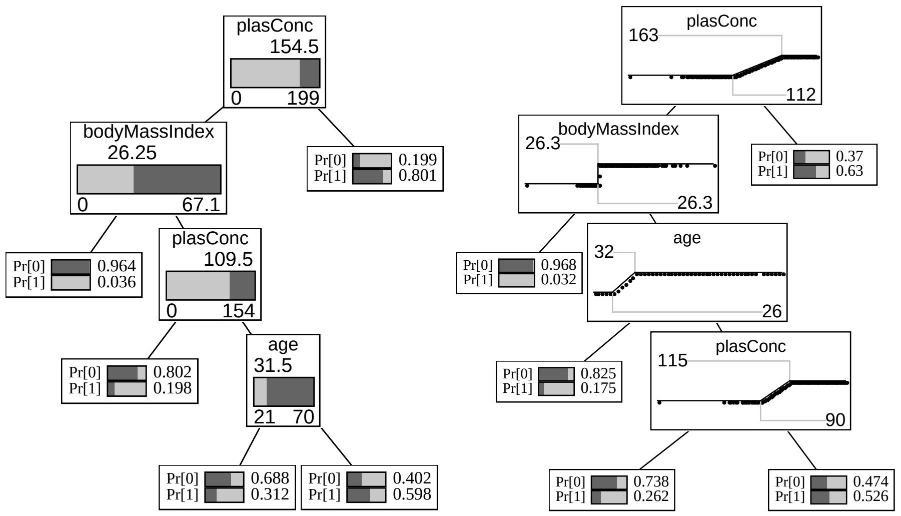

Fig. 1. Trees from two decision tree classes. The tree on the left uses 'hard' cut-points to test input attributes, while the tree on the right uses the 'soft' cuts.

5 Examples

This section briefly shows example trees from three CDMS decision tree classes. The first two trees were learnt from the "Pima Indians Diabetes Database", available from [2] and are depicted in figure 1. Both classes used a 2-state distribution as the leaf model, and a lookahead-0 search based on a Minimum Message

[2] Search functions provide a way to traverse the parameter space of a decision tree.

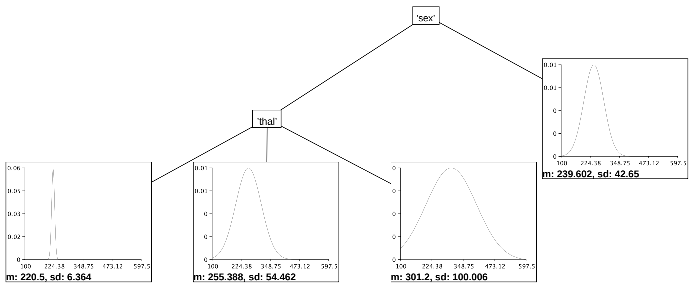

Fig. 2. This tree has been learnt from a cholesterol data set, obtained from the Weka website *http://www.cs.waikato.ac.nz/ml/weka/*. The data-set is a modified form of the Cleveland Heart Disease database collected by Dr Robert Detrano, available from [2].

Length (MML) [8–10] tree cost function, similar to that proposed in [11]. The difference is in the choice of branch methods. The tree on the left used the familiar 'hard' cut-point to test the continuous input attributes, while the tree on the right used 'soft' tests. The soft tests define a 'ramp' between two values a and b. Any item with a value less than a is assigned totally to the left-hand sub-tree, while any value greater than b is assigned totally to the right. Any value x between a and b is fractionally assigned to the right hand sub-tree with weight $(x - a)/(b - a)$.

Figure 2 illustrates a third class of decision tree. This tree uses discrete branch methods to test input attributes, and uses a Gaussian density function as the leaf model, supplying a probability density over the continuous target attribute.

6 Results

We now compare the performance of the two decision tree classes shown in figure 1 and that of C5 [6]. Five data sets were analysed - wine, iris, E-coli, pima, and glass - and are all available from [2]. Table 1 summarizes their characteristics. The data-sets chosen each had a categorical (discrete) target attribute, allowing easy comparison with C5 - which, unlike the tree classes presented here, is not able to model continuous data.

Table 1. A summary of the nature of the five data-sets.

Data-set	Target Attribute	Other Attributes
Wine	3-valued discrete	12 continuous attributes
Iris	3-valued discrete	4 continuous attributes
E-coli	8-valued discrete	7 continuous attributes
Pima	2-valued discrete	8 continuous attributes
Glass	7-valued discrete	9 continuous attributes

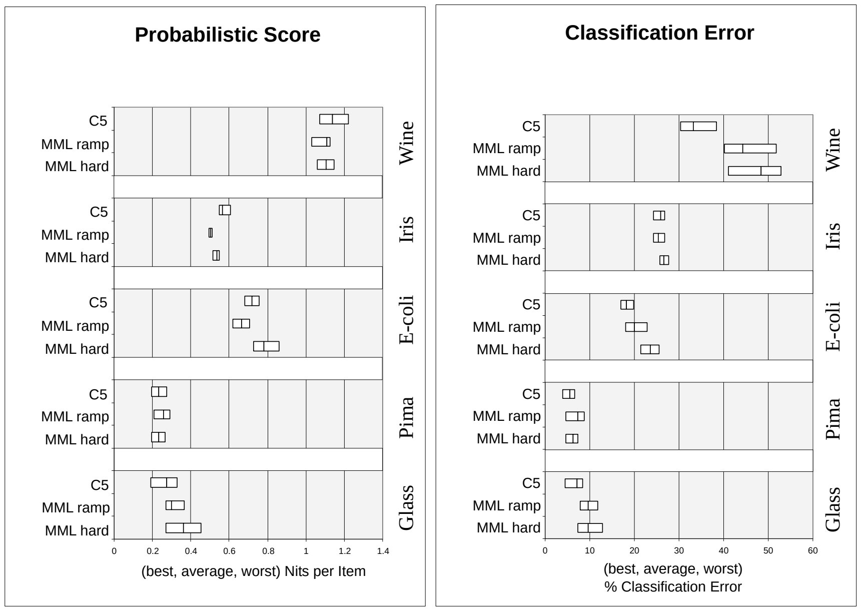

Fig. 3. A comparison of three decision tree methods. 'MML-ramp' refers to the tree class on the right in figure 1, using soft cut-points to test continuous input attributes. 'MML-hard' refers to the tree class on the left in figure 1, which like C5, uses hard cut-points to test continuous attributes.

We have assessed each method using both classification error and probabilistic score. Classification error simply measures the percentage of misclassified test cases (which only makes sense when dealing with discrete target attributes). Probabilistic score is a more general measure of performance, which penalises a method according to the negative logarithm of the probability the method assigns to the test data. This score rewards accurate estimates of probabilities, taking into account the certainty with which a prediction is made, rather than simply treating each prediction as 'right' or 'wrong'. Probabilistic score is thus more informative than classification error, and reflects our interest in *probabilistic* prediction. For many real-world instances, we would like to know not just the most likely outcome, but how likely this outcome is. For example, in a medical scenario, if we believed that a radical new treatment was 99% likely to succeed, the patient may feel differently than if the chance of success was believed to be only 51%.

For each data-set, we performed a 10-fold cross-validation experiment (with 90% training data, 10% testing data). The performance of each method was averaged over the 10 test-sets, resulting in a score, s. We then repeated this 10 times, recording for each method the best, worst, and average s. This is shown in figure 3.

Although C5 often yields superior classification error, we can see from the probabilistic score that it does not do so well at inferring accurate probabilities. An explanation for this is that it tends to estimate more extreme distributions, and may therefore make predictions with more certainty than is warranted. While assigning a near-zero probability to a test item does not impact greatly on classification error, it can have a disastrous effect on probabilistic score.

The results for the $iris$, $E - coli$ and $wine$ data-sets demonstrate the merits of a more general class of decision tree. In the probabilistic score especially, we can see that the 'soft ramp' branch tests significantly improve performance, providing a more accurate probabilistic model of the target attribute.

7 Conclusion

We have given an overview of a general data-mining platform that provides in-built support for statistical models in its type system. Using a decision tree 'plug-in' as an example, we have discussed how complex models can easily be constructed using simpler models as building blocks. This approach lets us quickly build a versatile library of machine learning tools that are tightly integrated with the system, rather than being stand-alone algorithms.

References

1. L. Allison. Types and classes of machine learning and data mining. In *Proceedings of the Twenty-Sixth Australasian Computer Science Conference (ACSC2003)*, pages 207–215, Adelaide, South Australia, 2003.
2. C. Blake and C. Merz. UCI repository of machine learning databases, 1998. http://www.ics.uci.edu/~mlearn/MLRepository.html.
3. L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees.* Statistics/Probability Series. Wadsworth Publishing Company, Belmont, California, U.S.A., 1984.
4. L. Fitzgibbon, L. Allison, and J. Comley. Probability model type sufficiency. In *Proc. 4th International Conference on Intelligent Data Engineering and Automated Learning*, 2003.
5. R. Ihaka and R. Gentleman. R: A language for data analysis and graphics. *Journal of Computational and Graphical Statistics*, 5(3):299–314, 1996.
6. J. R. Quinlan. C5.0. http://www.rulequest.com.
7. W. N. Venables and B. D. Ripley. *Modern Applied Statistics with S-PLUS.* Springer, 3 edition, 1999.
8. C. S. Wallace and D. M. Boulton. An information measure for classification. *Computer Journal*, 11:185–194, 1968.
9. C. S. Wallace and D. M. Boulton. An invariant Bayes method for point estimation. *Classification Society Bulletin*, 3(3):11–34, 1975.
10. C. S. Wallace and P. R. Freeman. Estimation and inference by compact coding. *J. Royal Statistical Society (Series B)*, 49:240–252, 1987.
11. C. S. Wallace and J. D. Patrick. Coding decision trees. *Machine Learning*, 11:7–22, 1993.
12. I. H. Witten and E. Frank. *Nuts and bolts: Machine learning algorithms in Java.* Morgan Kaufmann, 1999.