Chapter IX
Programming Drills with a Decision Trees Workbench

Dimitris Kalles
Hellenic Open University, Greece

Athanasios Papagelis
University of Patras, Greece

ABSTRACT

Decision trees are one of the most successful machine learning paradigms. This chapter presents a library of decision tree algorithms in Java that was eventually used as a programming laboratory workbench. The initial design focus was, as regards the non-expert user, to conduct experiments with decision trees using components and visual tools that facilitate tree construction and manipulation and as regards the expert user, to be able to focus on algorithm design and comparison with few implementation details. The system has been built over a number of years and over various development contexts and has been successfully used as a workbench in a programming laboratory for junior computer science students. The underlying philosophy was to achieve a solid introduction to object-oriented concepts and practices based on a fundamental machine learning paradigm.

INTRODUCTION

A decision tree is a graphical representation of a procedure for classifying or evaluating an item of interest. It represents a function that maps each element of a domain to a value from a set of values; this value is typically a symbolic class label or a numerical value. Decision trees are excellent tools for supporting decisions, when a lot of complex information must be taken into account and the reasoning must be supplied for alternative paths (Mitchell, 1997).

Decision trees have two key merits when compared to other concept learners. First, they

can manipulate a large amount of information due to the small computational power that is needed for the creation of the model of the underlying hypothesis (furthermore, the representation of the model does not demand excessive memory). Second, by providing classifications and predictions that can be argued about, they advance our insight in the problem domain.

Their success has motivated many researchers to attempt to improve decision tree learners. Efforts have mostly focused on pre-processing data (Musick et al., 1993; Quinlan, 1993), selecting splitting attributes (Breiman, 1984; Mingers, 1989) and tree pruning (Breiman, 1984; Quinlan, 1987).

Mundane but important tasks take up, usually, a large portion of the programming effort, when trying out a new idea. Such tasks include parsing input, creating data structures and statistics, printing and classifying. However, most of these typical components remain unchanged, even when a researcher wants to create a new tree algorithm. This observation necessitates the effort towards reusing the most flexible components, which can be easily adapted to each researcher's requirements.

The library described in this chapter addresses directly the problem of focusing research effort where it is mostly needed when one designs or implements a decision tree algorithm, which is the algorithm itself. The library is organized in components, each one corresponding to a clearly distinct stage of the tree building process. This architecture reduces the time of creating a tree algorithm by providing building blocks of the algorithm that do not need to be changed.

Two popular similar libraries are MLC++ (Kohavi et al., 1994) and WEKA (Witten & Frank, 2000), with WEKA being today the *de facto* choice and MLC++ being effectively sidelined. Both of them contain common induction algorithms (i.e., C4.5 (Quinlan; 1993), Naïve Bayes, ID3 (Quinlan; 1986)) under a unified framework. Moreover, they contain wrappers to wrap around algorithms including feature selection, discretization filters and bagging/combining classifiers and they provide a means for testing classifier accuracy.

This work will not replace those established and global machine-learning tools. Moreover, it does not compete with focused applications (Quinlan; 1993). Our library has a more limited scope: it focuses on providing the necessary infrastructure for creating and manipulating binary decision trees. Reducing the scope provides a more solid framework for the specific problem and results in a more attractive learning curve. Moreover, by limiting our attention to one domain, we can use the standard steps of decision tree learning as a pre-defined backbone, to which all new components must conform. Specificity, in this sense, allows for a tighter definition of which interface criteria components must satisfy and, eventually, results in a more structured (and easier) way of designing new algorithms.

The library is an open growing system (Christodoulou, 2001; Christodoulou et al., 2004; Christodoulopoulou, 2006; Drossos et al., 2000; Mpekou, 2006; Papagelis & Drosos, 1999) that supports the addition of algorithms and components, yet it is component and not algorithm oriented. The added architectural complexity it creates, from the software engineering perspective, is efficiently managed through a GUI, which provides an easy way to interchange "building blocks" between different tree implementations and to compare competing designs. In this respect, even if MLC++ and WEKA provide a good base of existing algorithms, they squarely trail our approach when the focus is on capability and usability to enhance the repertoire of algorithms.

The rest of the chapter is organized in six sections. In the next section, the basic characteristics of a decision tree algorithm are presented and, then, the library is described and the mapping of each component of a decision tree algorithm to each component of the library is explained. We then describe the shell used for creating and manipulating new objects and follow with