Journal of Statistical Software

August 2018, Volume 86, Issue 6. doi: 10.18637/jss.v086.i06

mbonsai: Application Package for Sequence Classification by Tree Methodology

Yukinobu Hamuro
Kwansei Gakuin University

Masakazu Nakamoto
Kwansei Gakuin University

Stephane Cheung
Kwansei Gakuin University

Edward H. Ip
Wake Forest University

Abstract

In many applications such as transaction data analysis, the classification of long chains of sequences is required. For example, brand purchase history in customer transaction data is in a form like AABCABAA, where A, B, and C are brands of a consumer product. The decision tree-based package **mbonsai** is designed to handle sequence data of varying lengths using one or multiple variables of interest as predictor variables. This software package uses tree growing and pruning strategies adopted from C4.5 and CART algorithms, and includes new features for handling sequence data and indexing for classification purpose. The software uses a simple command line program for learning and predicting processes, and has the ability to generate user-friendly graphics depicting decision trees. The underlying C++ codes are designed to efficiently process large data sets in ASCII files. Two examples from transaction data sets are used to illustrate the application of **mbonsai**.

Keywords: decision tree, sequence, classification, alphabet indexing.

1. Introduction to mbonsai

mbonsai is a tree-based classification program that can delineate patterns in sequence data – an ordered collection of categorical, numerical, or ordinal observations – and provide rules for splitting the sequence variable space such that the classification of individual cases is possible. A prototypical application of **mbonsai** is to classify customers' purchase of different brands in sequence order in terms of their possibilities to churn – i.e., stop purchasing from the target brand at a later period. To illustrate the key features of **mbonsai**, consider the following

BrandSequence	Churn
cab	yes
ab	yes
bbab	yes
acca	yes
aaab	yes
aacc	no
bacbc	no
abca	no
abca	no

Table 1: Example of sequence data. Each line corresponds to one customer, with the target variable indicating the churn status of the customer. The alphabetic characters a, b, and c, shown in the "BrandSequence" column, represent the brand purchased by the customer in the order shown.

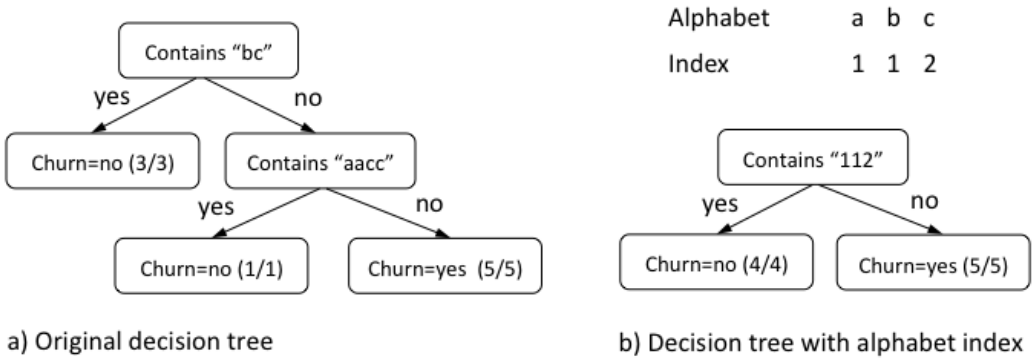

Figure 1: Example of a customer churn model: (a) the decision tree for the original brand sequence data; (b) the decision tree for the index pattern data.

example. A diaper manufacturer of a brand is interested in using previous diaper purchasing pattern (e.g., when using size M) to predict churning (i.e., switching to a different brand when using size L as the baby grows). Table 1 shows the purchase pattern, or sequence, of 9 customers' purchases of 3 brands of diaper of size M and the churn status of the customer. Churn=yes indicates that the customer switches to another brand for size L diaper. Churn status is determined separately using transaction data on size L diapers. The first record, for example, shows a customer purchased brand c, then a, and then b. This customer eventually churned.

mbonsai first uses an alphabet index set to map the brands, or alphabets, into a smaller number of indexes, in this case $\{1, 2\}$. The mapping is such that brands a and b map into index 1, and brand c maps to index 2. When the number of alphabets is large, alphabet indexing simplifies a model and, when done properly, preserves information in the sequence patterns for classification purposes. Figure 1 shows decision trees for respective classification based on the original alphabets and the index. When multiple predictor variables are present, it is possible to apply alphabet indexing to all of the predictor variables together or to a set of selected predictors. Details on subsequence and candidate patterns matching is illustrated in Section 3.

The software concept underlying **mbonsai** was originally initiated as a machine learning system called **BONSAI Garden** (Shimozono, Shinohara, Shinohara, Miyano, Kuhara, and Arikawa 1994), which was motivated by pattern discovery from amino acid sequences of proteins. The word BONSAI symbolizes knowledge, as a small bonsai tree comprises various regular patterns that are in harmony with alphabet indexing to reduce the size of the tree.

In our implementation of **mbonsai**, we have extended the algorithms in **BONSAI** for analyzing large-scale business data, especially for the prediction of consumer behavior (Kawata, Hamuro, Kato, and Yada 2001; Katoh, Yada, and Hamuro 2003; Yada, Hamuro, and Katoh 2007a; Yada, Ip, and Katoh 2007b). The key features in this new implementation are as follows:

- Transform patterns from numerical and categorical sequence data into classification conditions.
- Use alphabet indexing as a data reduction technique for sequence data.
- Process multiple variables of sequence data, using numerical and categorical variables as predictors.
- Allow a cost sensitive learning approach to optimize misclassification costs by the integration of a cost matrix.
- Allow separate training and testing of decision tree models.
- Allow two or more classes of the target variable for classification.
- Allow cross-validation for assessing the performance of the predictive model.

mbonsai is a data mining tool within the **NYSOL** software package (**NYSOL** Corporation 2014). **NYSOL** is an integrated framework of knowledge discovery which contains a collection of command driven tools known as **m-commands** designed for large-scale data processing and data mining. The underlying data processing methodology for the set of **m-commands** was developed by Yasuyuki Matsuda. Command names within the **NYSOL** package including **mbonsai** are preceded by the letter "m" in honor of the developer. The **NYSOL** package provides a wrapper command underpinned by existing algorithms and data processing programs, for simple execution of data transformation, data aggregation, data mining, and visualization at command line. This methodology allows users to integrate and manage all text-based information in one system throughout the knowledge discovery process. Both **mbonsai** and **NYSOL m-commands** were developed in C++ for scalable implementation in the UNIX environment by the JST ERATO Minato Discrete Structure Manipulation System Project hosted by Hokkaido University, Japan, and was distributed under the terms of the GNU Affero General Public License Version 3 (`https://www.gnu.org/licenses/agpl-3.0.html`). The **mbonsai** decision tree application can be executed as a simple UNIX command directly on the command line and is customizable with user-defined parameters. **mbonsai** can be installed as a standalone package on LINUX and Mac OS X platform at the terminal emulator. More details are described in Section 5.

The UNIX-based, command-driven **mbonsai** was designed for the direct processing of large-scale, text-based data. The default input format is CSV (comma-separated values), which has the advantage of being highly accessible and can be processed with great efficiency.

This paper is organized as follows. Section 2 provides a brief description of the background and algorithms underlying **mbonsai**. Section 3 describes the strategy of growing a tree using sequence data. Section 4 describes the required data structure and preparation for data input for the software. Section 5 describes the functions and parameters of **mbonsai**. The application of **mbonsai** is then illustrated through two classification examples with real customer purchase transaction data in Section 6. Finally, we provide brief concluding remarks.

2. Tree algorithms

mbonsai is built upon the C4.5 (Quinlan 1993) and CART (Olshen, Breiman, Friedman, and Stone 1984) algorithms for the classification of sequence data. The core algorithm of tree building in **mbonsai** is based on information entropy in the **C4.5/C5.0** implementation by Rulequest (2013), whereas the algorithm of tree pruning is similar to that used in **CART** which is implemented by Salford (2015). See Kuhn and Johnson (2013) for a recent review of tree-based classification methods. Briefly, at each node of the tree, **mbonsai** selects the predictor variable for the split in terms of information entropy. Not unlike **CART**, **mbonsai** employs a greedy algorithm for splitting rules, grows a full tree until a terminal node contains a small sample (e.g., $n < 5$), and then performs cost complexity pruning on the full tree to prevent overfitting. Readers are referred to Quinlan (1993) and Olshen *et al.* (1984) for technical details for C4.5 and CART algorithms respectively. The differences between C4.5 and CART algorithms are summarized in Wu *et al.* (2008). The recent development of tree algorithms, and the associated software programs are described in conjunction with unique features in **mbonsai**.

The first recent development is the use of a non-greedy algorithm to circumvent possible biases introduced in model selection from a greedy algorithm. One example is the evolutionary method for learning globally optimal trees and related software (Grubinger, Zeileis, and Pfeiffer 2011). An alternative approach is to split each node into as many children nodes as the number of classes and use F tests to rank the predictor variables, as implemented in **CRUISE** (Kim and Loh 2001). Because of the potential large search space in sequences, **mbonsai**, however, relies on a greedy search algorithm.

The second recent development is the emergence of ensemble-based classifiers. It represents another important development for improving the accuracy of predictions of tree-based classifiers. Bagging (Breiman 1996) and boosting (Freund and Schapire 1997) are two noted examples of this class of classifiers. Open source programs for ensemble-based classifiers include **randomForest** (Liaw and Wiener 2002) for bagged trees, and **gbm** (Ridgeway 2017) for generalized boosted models including boosted trees. Some implementations of ensemble-based classifiers such as the R-based package **bst** (Wang 2018), allow the plugging-in of loss functions. **mbonsai** also allows specification of loss functions but does not use ensemble-based methods. Because ensemble-based methods are general and can be applied to almost any type of classifier, it is possible to further enhance the accuracy of **mbonsai** through ensemble-based methods.

A third recent development are the multivariate and longitudinal extensions of C4.5 and CART algorithms, which is especially relevant to **mbonsai**. Examples of such extensions include Segal (1992) for longitudinal recursive partitioning, Zhang and Singer (2010) for multivariate binary tree classification, and Yu and Lambert (1999) for a functional curve

approach. Unlike Yu and Lambert (1999), **mbonsai** focuses on response vectors that are categorical, not continuous. Yet another recent approach for analyzing longitudinal data is the RE-EM method by Sela and Simonoff (2012), which treats longitudinal data as repeated measurements and the temporal information is viewed more or less as a nuisance factor. In contrast, **mbonsai** considers temporal information as central to classification and such information is embedded in a sequence.

Two other recent work that are most closely related to the approach in **mbonsai** are the temporal decision tree (Console, Picardi, and Theseider 2003) and the sequence decision tree (Rokach, Romano, and Maimon 2008). Incidentally, both tree-based models have been inspired by applications in production and manufacturing. Both approaches use some form of C4.5, but are different from **mbonsai**. For example, the sequence tree algorithm proposed by Rokach *et al.* (2008) treats temporal data as sequences. The sequence of operational setting of a product (e.g., an automobile) is first coded as a string of token. For example, the sequence B 1–5–9–3–2 F represents assembly procedure 1, 5, 9, 3, 2 (and in that order). B and F respectively denotes begin and finish. The goal of sequence tree analysis is to delineate effects of the operation sequence on the quality of the product (e.g., fail/pass). The algorithm consists of several steps which include (1) using regular expression to represent relevant patterns in pairs of sequences – e.g., $(1, 5), (5, 9), (9, 3), \ldots$ in the above example, and (2) using C4.5 to classify useful patterns derived from regular expressions. In other words, preprocessing data plays an important role in this approach, and the original decision tree algorithm C4.5 is applied to the derived pattern data. The approach thus is different from **mbonsai** within which the C4.5 tree algorithm is extended to *directly* analyze sequence data. Earlier applications of **mbonsai** to business transaction data with discrete and continuous variables can be found in Yada *et al.* (2007b).

3. The mbonsai algorithm

The process flow of **mbonsai** is shown in Figure 2, and Figure 3 shows the overall algorithm of **mbonsai**. The algorithm takes a sequence dataset D and pruning parameter α as input, and it returns a decision tree T with alphabet-index f. Alphabet indexing is a notable

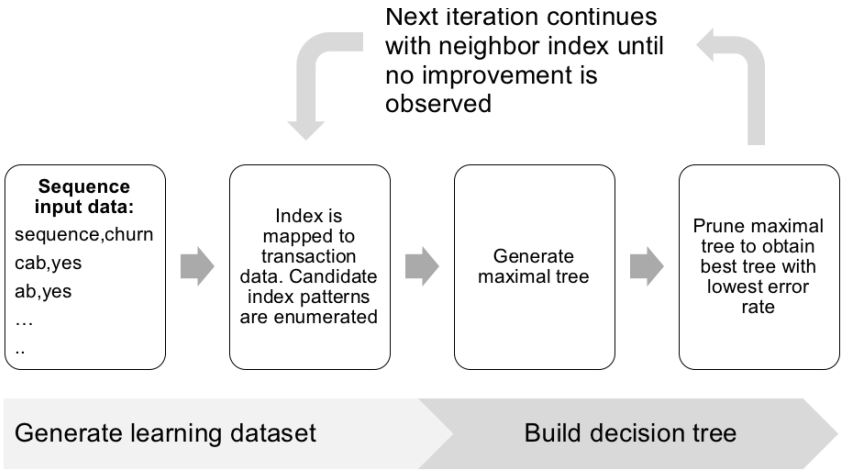

Figure 2: Workflow of **mbonsai**.

```
 1: Procedure bonsai(D, α)
 2:   D: sequence dataset
 3:   α: pruning parameter
 4: f = initIndex(D)                  f is a mapping f : Σ → I
 5: e = 1.0                           initialize an error rate
 6: while
 7:   F = neighborIndex(f)
 8:   e' = 1.0
 9:   foreach f'' ∈ F                 find the best mapping in F
11:     S = mkDataset(D, f'')
12:     T'' = mkTree(S, α)
13:     e'' = error(T)
14:     if e'' < e'
15:       e' = e''; T' = T''; f' = f''
16:     end
17:   end
18:   if e' < e                        update the best tree T and mapping f
19:     e = e'; T = T'; f = f'
20:   else
21:     break
22:   end
23: end
24: return T, f
```

Figure 3: Main procedure of **mbonsai**

feature of **mbonsai**, which is a mapping $f : \Sigma \to \mathcal{I}$, where $\Sigma = \{a_1, a_2, \ldots, a_n\}$ is a alphabet set corresponding to the elements of the sequence, and $\mathcal{I} = \{b_1, b_2, \ldots, b_m\}$ is an index set. Usually m is set at a much smaller number than n, so indexing works as grouping of elements of a sequence. The algorithm explores the best mapping which minimizes error (misclassification rate) of the model.

If n is small, it is possible to exhaustively search the entire space of possible partitions for optimal alphabet indexing. When either n is large or a relatively large number of indexes m is required, a local search technique is used. The local search starts by generating a mapping at random (line 4 in Figure 3). In the examplary sequence shown in Table 1, $\Sigma = \{a, b, c\}$ and $\mathcal{I} = \{1, 2\}$, applying randomized mapping generates a mapping such as $f = \{(a, 1), (b, 1), (c, 2)\}$.

Afterwards, all mapping combinations similar to mapping f are enumerated and stored to a mapping set F. In the above example, close mappings of f are $f_0 = \{(a, 2), (b, 1), (c, 2)\}$, $f_1 = \{(a, 1), (b, 2), (c, 2)\}$, $f_2 = \{(a, 1), (b, 1), (c, 1)\}$, but the last mapping is eliminated because $m = 1$, so $F = \{f_0, f_1\}$.

With respect to each element f'' of the mapping set F, it calculates the best tree T' and mapping f' with lowest error (line 14–15). Then the best mapping f is updated. The process is repeated until no improvement is observed.

We describe the generation of a learning dataset (line 11) and the construction of a decision tree (line 12) below.

BrandSequence	Churn
211	yes
11	yes
1111	yes
1221	yes
1111	yes
1122	no
11212	no
1121	no
1121	no

Table 2: Indexed sequence converted from Table 1 based on the mapping $f = \{(a, 1), (b, 1), (c, 2)\}$.

3.1. Generating a learning dataset

First of all, the original sequence data on dataset D is converted to an indexed data based on the given mapping f''. For example, the sequence data in Table 1 is converted to the one shown in Table 2, based on the mapping $f'' = \{(a, 1), (b, 1), (c, 2)\}$.

Subsequently, a learning dataset for building a decision tree will be generated from the indexed data sequence. Input variables are patterns of the sequence and they take the Boolean value of 0 or 1. **mbonsai** uses "regular pattern" as pattern.

Regular patterns

"Regular patterns" refer to the collection of generic sequence patterns designed for matching sequences seen in the data. Apparently, the maximum length of a regular pattern of interest equals the maximum length of observed sequence data. However, in practice one requires to limit computation costs by specifying the maximum length of regular patterns (e.g., capped at 5), so the matching procedure will only consider subsequences in the data that have a maximum length of 5. Define the alphabet set Σ as a collection of characters. This could be, for example, a collection of brands as indicated by letters `a`, `b`, `c` and so on. Regular patterns can either be a string or a sequence. In string matching, no other alphabets are allowed between alphabets. For example, if the string `aab` is to be matched, then the data must appear exactly as `aab`. However, if `aab` is treated as sequence, then an observed data of the form `acaccb` is still considered a match. Thus, as sequence, any data of the form `*a*a*b*` is considered a match, where `*` is a wildcard. Note that here we distinguish between the terms "sequence" and "string", whereas the broader term "sequence data" used earlier refers to generic data that contain a chain of alphabets. Formally, define n substrings $\pi_1, \pi_2, \ldots, \pi_n$ on alphabet Σ, and $n + 1$ substrings $x_0, x_1, \ldots, x_n$ that are used as wild cards. A regular pattern takes the form $x_0\pi_1x_1\pi_2x_2 \cdots \pi_nx_n$. In **mbonsai**, the pattern-matching algorithm allows both string and sequence patterns. For string, **mbonsai** takes a substring pattern of the form $x_0\pi_1x_1$ unless otherwise specified (see also the discussion on begin / end match) and subsequence of the form $x_0\pi_1x_1\pi_2x_2 \ldots$. The use of string and sequence patterns can be specified in the second parameter of the **mbonsai** command at `p=`. An example of usage is given in Section 5.

a	b	c	aa	ab	cc	⋯	Churn
1	1	1	0	1	0		yes
1	1	0	0	1	0		yes
1	1	0	0	1	0		yes
1	0	1	0	0	1	⋯	yes
1	1	0	1	1	0		yes
1	0	1	1	0	1		no
1	1	1	0	0	0		no
1	1	1	0	1	0		no
1	1	1	0	1	0		no

Table 3: Each candidate pattern is extracted from the original sequence data as shown in the "BrandSequence" column from Table 1. Each candidate pattern is matched with the original sequence, and the presence of the matching candidate pattern is converted to 0–1 values. A value of 0 indicates "not matched" and 1 indicates "matched".

Generation of candidate patterns

Candidate patterns are enumerated for each node in a decision tree for classification accuracy, and a selected number of candidate patterns (e.g., the top 30) for each variable will be considered in splits when growing a tree. The enumeration is based on the following heuristic. First, regular patterns generated from the index with length 1 are stored in a priority queue. Regular patterns in the priority queue are ordered by an entropy measure (defined below). At the next step, the regular pattern with the lowest entropy in the priority queue is selected, and a second index is added to the selected regular pattern, resulting in an updated regular pattern with length of 2. The updated regular pattern is stored in the priority queue again and evaluated. The above steps are then repeated. If a regular pattern with length k is selected, an updated regular pattern with length $k+1$ is stored in the priority queue. In **mbonsai**, the default length of index is set at 5, and the upper limit of the regular pattern k can be specified in the sixth parameter of `p=`. The iteration process terminates when the size of the regular pattern exceeds the number of candidates, which can be specified in `cand=`. The default is set at `cand=30`. The following measure of information entropy is used to both evaluate regular patterns and compute splitting rules at the nodes of the decision tree:

$$\mathrm{ent}(\pi) = -q^{\mathrm{m}(\pi)} \sum_{i=1}^{c} p_i^{\mathrm{m}(\pi)} \log p_i^{\mathrm{m}(\pi)} - q^{\mathrm{u}(\pi)} \sum_{i=1}^{c} p_i^{\mathrm{u}(\pi)} \log p_i^{\mathrm{u}(\pi)}, \tag{1}$$

where c denotes the number of classes, and $p_i^{\mathrm{m}(\pi)}(p_i^{\mathrm{u}(\pi)})$ represents the relative proportion of class i that matches (not matches) with the sample in the regular pattern π, with $\sum_i^c p_i^{\mathrm{m}(\pi)} = 1$. Additionally, $q^{\mathrm{m}(\pi)}(q^{\mathrm{u}(\pi)})$ represents the composition ratio of matching (not matching) with regular patterns π of all samples, with $q^{\mathrm{m}(\pi)} + q^{\mathrm{u}(\pi)} = 1$.

As an example, Table 3 shows candidate patterns, which are matched against the brand sequence data shown in Table 1. A value of 0 indicates "not matched" and 1 indicates "matched".

3.2. Building a decision tree

Selection of splitting rule

Like the implementation in CART, **mbonsai** adopts a top-down greedy algorithm for splitting branches of a tree; the splitting rules at the node of the tree are determined by the information change in the node due to the split. Specifically, the branching rule is such that the split maximizes the entropy gain. Given a specific node, the probability of belonging to class i, as estimated by the empirical proportion, is represented by p_i. Entropy is computed by the equation $\text{ent} = -\sum_{i=1}^{c} p_i \log p_i$. For a given splitting rule at a given node, denote the number of samples classified into class i by n_i. Accordingly, the ratio of the classified samples to class i is given by n_i/n, where n is the total sample size at the node. Equation 1 is used to compute the entropy gain as the difference of entropy $\text{ent}(\pi)$ after splitting the regular pattern π. After entropy is calculated for each splitting point, the splitting point with the maximal information gain among all splitting point is selected. The procedure is repeated until the tree can no longer be grown. **mbonsai** uses a stopping rule that requires a minimum sample size of a terminal node (e.g., `leafSize = 100`). The resulting fully-grown decision tree is referred to as the maximal tree.

Pruning

A maximal tree often overfits the data. In order to avoid spurious branches that "fit to noise", **mbonsai** adopts a strategy similar to that of CART by iteratively pruning back sections of the maximal tree. Denote the set of nodes t in a decision tree by T and the nodes in the maximal tree by $T_{\max} = \{t_1, t_2, \ldots, t_k\}$, and the subtree with root node t by T_t, $t \in P$, where P is the set of nodes t. The resubstitution misclassification rate of decision tree T is denoted by $R(T)$. Pruning aims to select a subtree T^* that has the lowest misclassification rate $R(T)$ on unseen data.

A practical method of controlling the size of a tree is based on the cost-complexity pruning method, which is implemented in CART. In brief, the method penalizes the estimated error based on the subtree size. The degree of penalty is controlled by the pruning parameter α. An efficient search algorithm can be used to compute all the distinct α values that change the tree size, and the parameter is chosen to minimize the error on a holdout sample.

The implementation of pruning in **mbonsai** is completed by the direct specification of the tuning parameter α, by the use of a holdout test sample, or by cross-validation. If the pruning based on a holdout test sample or cross-validation is selected, then the subtree with the highest prediction accuracy is reported.

Specifically, the evaluation function or penalized resubstitution error rate of the decision tree T, which measures cost complexity, is defined as $R_\alpha(T) = R(T) + \alpha|\tilde{T}|$, where $\alpha(\geq 0)$ is the tuning parameter. A subtree model is considered a better model if its associated cost is smaller. The complexity equation represents a tradeoff between the misclassification rate of a decision tree $R(T)$ and tree complexity, as measured by the number of leaf nodes in T, $|\tilde{T}|$. In general, for a given value of α, we can always find the subtree $T(\alpha)$ that minimizes $R_\alpha(T)$. Because there are a finite number of subtrees, the minimizing subtree for any α always exists and can be denoted by T_α. Furthermore, because there are at most a finite number of subtrees of $T_{\max}$, $R_\alpha(T)$ yields different values for only a finite number of α's. The pruned tree $\hat{T}^*$ that has a minimum prediction misclassification rate when applied to holdout data or

when evaluated using cross-validation is selected as the optimal tree. Specific details about computational procedure for pruning and related theoretical issues can be found in Olshen *et al.* (1984).

Pruning parameters

In model building mode, **mbonsai** controls pruning through the following parameters: test sample `ts=`, cross validation `cv=`, and `alpha=`. When `cv=` or `ts=` is specified, test data is used for predictive accuracy; the model with the minimum misclassification rate is selected. **mbonsai** also allows the direct specification of the value of the tuning parameter α. The option `alpha=` could be used for promptly defining a specific value of the tuning parameter and creating a decision tree for special purposes such as exploration or testing the effect of changes in α. The final tree model is saved in `model.txt` and `model_info.txt`.

In prediction mode (`-predict`), the α value is calculated internally unless the option `alpha=` is specified. In that case, the specified value of α is used to construct the decision tree for prediction.

In the test sample method, training data D is partitioned into two sets D_1, D_2 at a ratio of 1 : 2. The maximal tree based on training data D_2 is first constructed. Based on the complexity parameter $\alpha_1, \alpha_2, \ldots, \alpha_k$ obtained for pruned subtrees, D_1 is used as the set of unknown data to predict the value of the misclassification rate, and the best subtree is selected. If the costs for the test sample exceed the costs for the learning sample, then this is an indication of a poor model fit.

In the cross-validation method, training data D is partitioned equally into $D_1, D_2, \ldots, D_p$, and D_1 is treated as unknown data for prediction. This method is similar to the test sample method where a percentage of training data is used for prediction. Cycling through the D_j, $j = 1, \ldots, p$ as unknown data with the other data as training data, an average misclassification rate can be obtained by this method. Subsequently, among the complexity parameters $\alpha_1, \alpha_2, \ldots, \alpha_k$ for corresponding decision trees $T_1, T_2, \ldots, T_k$, the optimal decision tree with the lowest average misclassification rate is selected. Additionally, the smallest tree among all subtrees of which the estimated mean error rate is within one standard error of the overall error rate is selected ("1 SE rule").

Learning cost considerations

In many applications, the cost for misclassification may not be uniform across different outcomes. Using a two-class (positive and negative) model as an example, the cost for false-positives and false-negatives could be very different, and cost consideration would affect the construction of a decision tree. The construction of the classification model that takes misclassification costs into account is often called cost sensitive learning. Various methods have

real	predict	cost
positive	negative	2
negative	positive	5

Table 4: Example of defining the cost file for a two-class (positive/negative) example. The column name can be customized by the user, but the columns must follow the order of actual class, predicted class, and cost.

ProfitPattern	Profit	Gender	Class
55552342	7969	F	Loyal
525	5379	M	Loyal
5	1538	F	Loyal

Table 5: Data sample with 3 predictor variables and 1 target variable.

been proposed in the literature. For example, the method proposed by Olshen *et al.* (1984) modified the probability p_i of class i in the sample by assigning weights based on cost. Specifically, the cost of a case that belongs to class j but predicted in class i is expressed as $c(i|j)$; thus, the total cost of class i is expressed as $\sum_j c(i|j)$. This weight is assigned to class i, and probability p_i is accordingly updated.

In **mbonsai**, the cost function is specified by the creation of a definition file in CSV format. A two-class (positive/negative) example is shown in Table 4, in which the actual class (`real`), corresponding predicted class (`predict`), and associated cost (`cost`) are displayed in the same row. When the combination of actual class and predicted class is not specified, or the cost file is not defined, all costs of misclassifications are set to unity.

4. Data requirements

4.1. Variable types

There are three basic data types accepted by **mbonsai**: a sequence made up of alphanumeric characters, numerical variables, or categorical variables. For the target class variable, which is defined in `c=` (see Section 5), it is assumed that the variable is categorical. Note that **mbonsai** accepts a multiclass definition of the target variable. Predictor variables can exist in the form of sequence or single-value. Continuing the customer churning example in Table 1, we use other predictor variables – a pattern of profit, total amount of profit, and gender – to illustrate the three data types. Table 5 shows the three respective variables "ProfitPattern", "Profit" and 'Gender" respectively of the type numeric sequence patterns (1–5), numeric, and categorical. The algorithm for branching rules based on single-value numerical and categorical is similar to that of the C4.5 algorithm. Within **mbonsai**, sequential, numerical, and categorical data types are respectively specified in the `p=,n=,d=` parameters.

5. Installation and parameters of mbonsai

5.1. Installation

We will describe installation of the **mbonsai** standalone package for the Linux and Mac OS platform. When the user is at the website (`http://www.nysol.jp/mbonsai`), the user can download the zip archive file. Note that **mbonsai** requires Ruby 2.0 (Thomas, Fowler, and Hunt 2013), **gcc** and **g++ complier**, **boost** C++ libraries, and **libxml2**. Details of the installation instructions of prerequisite software and **mbonsai** can be found in the website.

The user can first create a parent directory, which is named `parentdir` in this example, and save the software program `mbonsai.zip` in the directory. Afterwards, launch the terminal (in

Mac OS, the terminal emulator is located under `/Applications/Utilities`). Alternatively, create the directory in the terminal using the `mkdir` command:

```
~$ mkdir parentdir
~$ cd parentdir
~/parentdir$
```

At the parent directory, execute the following commands to unzip the zip archive `mbonsai.zip`, and use the `ls` command to show the extracted files and folders in the directory. Then go to the `mbonsai/cmd` directory to execute the `make` command to install **mbonsai**:

```
~/parentdir$ unzip mbonsai.zip
~/parentdir$ ls

data  license.txt  mbonsai  mbonsaiscript.sh mbonsai.zip

~/parentdir$ cd mbonsai/cmd
~/parentdir/mbonsai/cmd$ make
```

This package contains three commands in the **mbonsai** package subfolders. The `mbonsai` decision tree command and the `mcm` classifier command are located within the `mbonsai/cmd` directory, the visualization command `mdtree.rb` is located in the `mbonsai/view` directory. The input data for this tutorial is located inside the `data` directory.

The user needs to define the directory path for **mbonsai** to execute the command. The input data is defined at `i=`, the name of the desired directory where output files are stored at `O=` parameter. The `p=` parameter accepts the column name of the pattern, while `c=` parameter accepts the column name of the class. In this example, at the parent directory, named `parentdir`, at which the command prompt is shown as `~/parentdir$`, the user can execute the command with corresponding parameters at command line as shown in the following structure:

```
~/parentdir$ mbonsai/cmd/mbonsai p=seq c=cls i=data/input.csv O=output
```

From this point on, the user can proceed to use the tutorial.

5.2. Parameters used in model construction mode

mbonsai is a command-based program. Output model and statistics are saved in text files and PMML formats[1]. For data learning and tree constructing, the command line in model construction mode contains the following parameters:

```
mbonsai i= [p=] [n=] [d=] c= O= [delim=] [cost=] [seed=] [cand=] [iter=]
 [cv= | ts=] [leafSize=] [--help]
```

The functions of the parameters are summarized in Table 6. Details of the usage of the

[1]The predictive model markup language (PMML) is an industry standard used to describe data mining and mathematical models represented in XML-based file formats. Note that this specific command uses an extended tag of the PMML standard.

`i=`	Training data file name.
`p=`	Column name of pattern (multiple fields can be specified). Users can specify up to five parameters after the column name, each separated by a colon, e.g., `p=column_name:is:seq:ordered:head:tail:rs`. The details of each are as follows: `is`: Size of the index – if this parameter is not specified, an index is not generated, instead, the original alphabet of the pattern is used. `seq`: type of pattern – true: partial sequence pattern; false: partial string pattern (default). `ordered`: Alphabetical order arrangement when generating the index (this parameter is ignored when `is` is not specified) – true: ordered, group alphabet above / below the threshold value; false: unordered (default) `head`: Match string or numeric characters from the beginning (default: start of string is not considered for matching). `tail`: Match string or numeric characters from the end (default: end of string is not considered for matching). `rs`: Upper size limit of regular pattern (default: 5).
`n=`	Column name with numerical data (multiple fields can be specified).
`d=`	Column name with categorical data (multiple fields can be specified).
`c=`	Column name of class.
`O=`	Output directory name (text, PMML model, and model statistics).
`delim=`	Delimiter character of pattern (default: empty character; a 1 byte character is regarded as 1 alphabet).
`cost=`	Name of cost file.
`seed=`	Seed of random number (default=−1: time dependent).
`cand=`	Number of patterns as predictor variable (default=30, range: 1–256).
`iter=`	Iterations of local search (default=1).
`leafSize=`	Lower limit of the number of samples in one leaf (default: no limit).
`alpha=`	Specify the degree of pruning. However, when `cv=` or `ts=` is specified, this parameter is disabled. Default=0.01.
`ts=`	Specify the percentage of test data partitioned using the test sample method. When `ts=` is not specified, the default value is set as 0.333.
`cv=`	Specify partition of data by cross-validation method. When `cv=` is not specified, the default value is set as 10. If either `ts= or cv=` is not specified, the default value of `alpha=0.01` will be applied. Even when `alpha=, ts=, and cv=`, are specified, the pruning degree of the maximum tree is recorded in PMML; the value of α could change in prediction mode.

Table 6: List of parameters for model building mode in **mbonsai**.

parameters in the training model are illustrated in Section 6.

5.3. Parameters used in prediction mode

The command for the prediction of new cases follows the format:

```
mbonsai -predict i= I= o= [alpha=] [--help]
```

`-predict`	Prediction mode [required for prediction].
`i=`	Input data [required]. The column names must be the same as the columns that were used for building the model.
`I=`	Destination directory path for model building mode [required]. Required files include: `bonsai.pmml`: pmml containing the decision model.
`o=`	Output file name containing the prediction result. The `predict` column is added to the input data in output. Columns must be the same as columns that were used in building the model.
`alpha=`	Specify the pruning complexity parameter. This parameter accepts real numbers greater than 0, as well as the following two symbols. Designation of the two symbols is effective only when `ts=` or `cv=` is specified when building the model. `min`: α value that corresponds to the pruned model, which minimizes the estimated misclassification rate. `1se`: α value that corresponds to the pruned model with the same 1SE rule. Default behavior: If `ts=` or `cv=` is specified when building the model, `min` is used. If you specify `alpha=` when building the model, the specified value is applied.
`delim=`	Delimiter character of pattern (default: empty character; a 1 byte character is regarded as 1 alphabet).

Table 7: List of parameters for prediction mode in **mbonsai**.

The list of user-defined parameters for predicting new data using the model generated in training mode are listed in Table 7. The `-predict` option must be switched on for running **mbonsai** in prediction mode.

6. Two applications to illustrate mbonsai

We use a real data example to illustrate the learning algorithm of **mbonasi** for the construction of a classification tree and the prediction for new cases. The data set contains purchase history of member customers of a drugstore chain in Japan (Hamuro, Katoh, Matsuda, and Yada 1999). The drugstore chain, Pharma, has collected purchase history of all of its 3,000,000 member customers. Using two subsets of data extracted from the drugstore chain's database, we will illustrate how **mbonasi** builds decision tree models to (1) identify core customers from the drugstore's perspective and classify new customers, and (2) identify loyal customers from a brand's perspective and predict loyal new customers. The first data set contains sequence data for 16,092 members, and the second dataset contains the transaction history of 114,069 members.

6.1. Application I: Identification of core customers

Core customers are those that have consistently generated a high level of profit and ultimately form a stable basis of income streams for a company. The objective in this example is to identify core customers based on the first 13 weeks of purchase history. **mbonsai** is used to construct a tree from transaction data of 16,092 customers using the following identified variables: average profit per visit ("Profit"), pattern of level of profit ("ProfitPattern"), number of visits during that period ("Visit"), and weekly visit pattern ("VisitPattern"). "Profit" and "Visit" are both numerical variables, whereas "ProfitPattern" and "VisitPattern" are

ProfitPattern	VisitPattern	Profit	Visit	Target
55552342	1100011101011	7969	11	Core
525	1000001100000	5379	3	Core
5	1000000000000	1538	1	Core
3545	1000000001011	2760	8	NonCore
42	1000000010000	566	2	NonCore
...				

Table 8: Example of input data `core.csv`. Each line corresponds to one customer, and the target variable indicates the status of the customer. The predictor variables include "ProfitPattern", "VisitPattern", "Profit", and "Visit".

sequence data. Each weekly profit value is transformed into a 5-level ordinal variable to form the sequence "ProfitPattern" (5 = high, 1 = low). In contrast, "VisitPattern" is a sequence of 13 binary (0/1) variables where 0 and 1 correspond to non-visit and visit in a specific week over the 13 week period. The target 2-class variable indicates the status of the customer – i.e., whether the customer belongs to the core customer group. Table 8 shows a sample of the data in CSV format.

Core customer classification model

Based on the program and file locations explained in the previous section, the following command creates the first classification model, and the output is saved in the `result_core` directory:

```
~/parentdir$ mbonsai/cmd/mbonsai p=ProfitPattern:2::true,VisitPattern \
> n=Profit,Visit c=Target i=data/core.csv O=result_core seed=100
```

The field name of pattern variables is specified by the `p=` parameter. The number of alphabet indexes is set at 2 for "ProfitPattern", and since it is an ordered sequence, the parameter is defined as `ProfitPattern:2::true`. The third parameter is blank after `ProfitPattern:2` and defaults to ordered sequence. "VisitPattern" only contains 0 and 1 in the sequence, customized parameters are not required. The field names of numeric variables are specified by the `n=` parameter, and the target variable by the `c=` parameter. The input file can be specified by the `i=` parameter, whereas the output file is specified by the `O=` parameter. A random seed can be specified by the `seed=` parameter, the seed is set at 100 in this example. The use of the same random seed would ensure that the same set of results would be obtained. Using a different random seed could lead to slightly different results. The results shown below may differ depending on the random number generator in your system.

The summary results of the model is by default stored in `result_core/model.txt`. Results are also interactively displayed as sections. `[alphabet-index]` shows the alphabet corresponding to the index. "ProfitPattern" classes 5, 2, and 3 are indexed in index 1 and category classes 4 and 1 are indexed as 2. "VisitPattern" is by default indexed as 0 and 1.

```
[alphabet-index]
Field Name: ProfitPattern
Index[1]={5,2,3}
Index[2]={4,1}
```

```
Field Name: VisitPattern
Index[1]={1}
Index[2]={0}
```

`[decision tree]` shows the pruned decision tree in text format. Information such as model size (number of leaf nodes) and number of layers of the deepest leaf is reported as well. The results of the tree show 1 split where profit per visit is less than or equal to 1480.5, in which case customers are classified as non-core, and if profit per visit is more than 1480.5, customers are classified as core.

```
[decision tree]
if($Profit <= 1480.5 )
  then $Target=NonCore (hit/sup)=(5680/8037)
  else $Target=Core (hit/sup)=(5249/7175)

numberOfLeaves=2
deepestLevel==1
```

`[Confusion Matrix by Training]` shows performance of the model using training data. The confusion matrices by count and by cost are shown. Overall prediction accuracy is 0.718446 for this model. Out of 15,212 cases, the sum of misclassification cost in this model is 4283. When the parameter cross validation `cv=` or test sample `ts=` is specified, the confusion matrix by estimation will be shown instead.

```
[Confusion Matrix by Training]
## By count
         Predicted As ...
        Core    NonCore Total
Core    5249    2357    7606
NonCore 1926    5680    7606
Total   7175    8037    15212

## By cost
         Predicted As ...
        Core    NonCore Total
Core    0       2357    2357
NonCore 1926    0       1926
Total   1926    2357    4283

## Detailed accuracy by class
class,recall,precision,FPrate,F-measure
Core,0.690113,0.731568,0.253221,0.710236
NonCore,0.746779,0.706731,0.309887,0.726203

## Summary
accuracy=0.718446
totalCost=4283
```

Prediction using the classification model

mbonsai allows the use of test data to validate the classification model. The option `-predict` needs to be invoked. The predict mode reads from the `model.pmml` file, which has been generated in the previous step. The directory of the model output files can be specified by the `I=` parameter. Test data are specified by the `i=` parameter. In the test data, the original class label is known and is presented in the `Target` column. Execute the following command to build the prediction model using test data.

```
~/parentdir$ mbonsai/cmd/mbonsai -predict i=data/core_test.csv \
> o=result_core/predict.csv I=result_core
```

The predicted class for each individual is saved in the `predict` column in the output.

```
ProfitPattern,VisitPattern,Profit,Visit,Target,predict,Core,NonCore
4423,1010000100001,1203,4,Loyal,NonLoyal,0.2932686326,0.7067313674
5,1000000000000,816,1,Loyal,NonLoyal,0.2932686326,0.7067313674
525,1010000000010,4308,4,Loyal,Loyal,0.7315679443,0.2684320557
231511,1001110110000,859,6,Loyal,NonLoyal,0.2932686326,0.7067313674
4445,1101000001000,2724,4,Loyal,Loyal,0.7315679443,0.2684320557
555215,1001100011010,6103,7,Loyal,Loyal,0.7315679443,0.2684320557
5522521,1110010000111,4832,9,Loyal,Loyal,0.7315679443,0.2684320557
55,1000000000100,2760,2,Loyal,Loyal,0.7315679443,0.2684320557
...
```

Within the predict mode, a confusion matrix of classifier accuracy can be calculated with the `mcm` command in the **mbonsai** package. Specify the field name of the actual class as `ac=` parameter, the predicted class as `pc=` parameter, the predict model results from the previous step as `i=`, and the output directory as `O=` parameter, and run the command as follows.

```
~/parentdir$ mbonsai/cmd/mcm i=result_core/predict.csv ac=Target pc=predict \
> O=result_core/evalAcc
```

Three output files are generated in the `result_core/evalAcc` directory. The file `summary.csv` contains summary of model accuracy information, `class.csv` contains positive predictive accuracy information, and finally, `confMatrix.txt` contains the confusion matrix of positive and negative instances of prediction outcome.

Below shows the results from the three output files:

```
[summary.csv]
evaluation,value
accuracy,0.7047337278
error rate,0.2952662722
total records,1690
unpredictable records,0

[class.csv]
Target,TP,FN,FP,TN,upCnt,upRate,recall,precision,f1
```

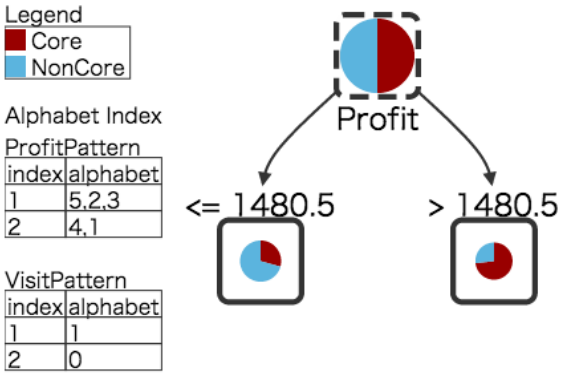

Figure 4: Visualization of the decision tree with samples of the two classes shown in a pie chart.

```
Core,559,286,213,632,0,0,0.6615384615,0.7240932642,0.6914038343
NonCore,632,213,286,559,0,0,0.7479289941,0.688453159,0.7169597277

[confMatrix.txt]
        a       b       u
a       559     286     0
b       213     632     0

a:Core
b:NonCore
u:unpredictable
```

Out of 1,690 cases in `core_test.csv`, 1,191 are correctly classified with an accuracy of 0.705.

Visualization of the decision tree

The decision tree can be visualized as a SVG (scalable vector graphics) graph that **mbonsai** embeds in an HTML file. The input to the SVG file is based on the PMML file, which has been created when the decision model was built. The following Ruby command generates the graph:

```
~/parentdir$ ruby mbonsai/view/mdtree.rb i=result_core/model.pmml \
> o=result_core/tree.html
```

The decision tree is visualized in Figure 4. The number of samples of each class at each node is shown in a pop-out area by placing the mouse cursor over the desired class. The two classes are compared in a pie chart by default. However, the chart can also be shown as a bar graph by adding the option `-bar` as follows. The diagram with bar chart option and pop-out area is shown in Figure 5.

```
~/parentdir$ ruby mbonsai/view/mdtree.rb i=result_core/model.pmml \
> o=result_core/tree_bar.html -bar
```

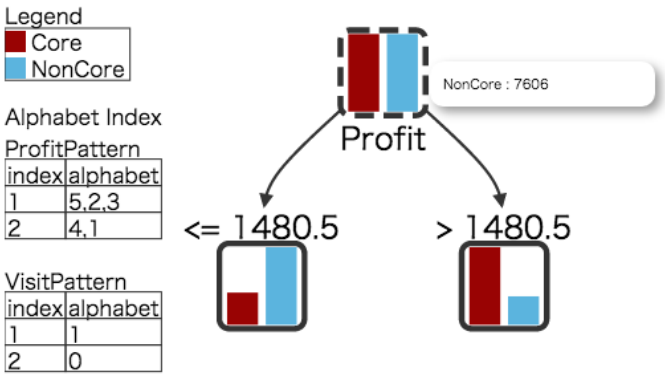

Figure 5: Visualization of the decision tree with samples of the two classes shown in bar graph.

Note: Users can modify the color of the graph by editing the `colorSet` in the `dictColor` function in `mbonsai/view/mdtree.rb`.

Refine the model

Altering the tuning parameter(s) α and/or leaf size allows users to adjust the tree. To illustrate the idea, we reran **mbonsai** using the α value of 0.00001 for pruning, together with a minimum number of 1,500 samples in each leaf.

```
~/parentdir$ mbonsai/cmd/mbonsai i=data/core.csv \
> p=ProfitPattern:2::true,VisitPattern n=Profit,Visit c=Target seed=100 \
> alpha=0.00001 leafSize=1500 O=result_core_mod
```

Results of alphabet index and decision tree from `result_core_mod/model.txt`:

```
[alphabet-index]
Field Name: ProfitPattern
Index[1]={5,2,3,4}
Index[2]={1}
Field Name: VisitPattern
Index[1]={1}
Index[2]={0}

[decision tree]$Target
if($Profit <= 1480.5 )
  then if($Profit <= 548.5 )
    then if($ProfitPattern has 1)
      then if($Profit <= 290.5 )
        then if($Profit <=Core -4373 )
          then $Target=Core (hit/sup)=(4/4)
          else $Target=NonCore (hit/sup)=(1915/2411)
        else $Target=NonCore (hit/sup)=(1035/1393)
      else $Target=NonCore (hit/sup)=(679/784)
```

```
    else if($Profit <= 971.5 )
      then if($ProfitPattern has 2212)
        then $Target=NonCore (hit/sup)=(8/8)
        else if($ProfitPattern has 11121)
          then $Target=Core (hit/sup)=(25/48)
          else if($ProfitPattern has 1121)
            then $Target=NonCore (hit/sup)=(44/58)
            else if($Visit <= 8.5 )
              then if($VisitPattern has 22111)
                then $Target=NonCore (hit/sup)=(20/24)
                else if($ProfitPattern has 12111)
                  then $Target=Core (hit/sup)=(9/12)
                  else $Target=NonCore (hit/sup)=(1017/1579)
              else $Target=Core (hit/sup)=(15/24)
      else $Target=NonCore (hit/sup)=(927/1692)
  else $Target=Core (hit/sup)=(5249/7175)

numberOfLeaves=13
deepestLevel==9

[Confusion Matrix by Training]
## By count
         Predicted As ...
        Core    NonCore Total
Core    5302    2304    7606
NonCore 1961    5645    7606
Total   7263    7949    15212

## By cost
         Predicted As ...
        Core    NonCore Total
Core    0       2304    2304
NonCore 1961    0       1961
Total   1961    2304    4265


## Detailed accuracy by class
class,recall,precision,FPrate,F-measure
Core,0.697081,0.730001,0.257823,0.713162
NonCore,0.742177,0.710152,0.302919,0.725812

## Summary
accuracy=0.719629
totalCost=4265
```

When an α value different from the default is used, an associated tree with 13 leaves and 9 levels is generated. Note that for the new value of α, the misclassification cost is only reduced

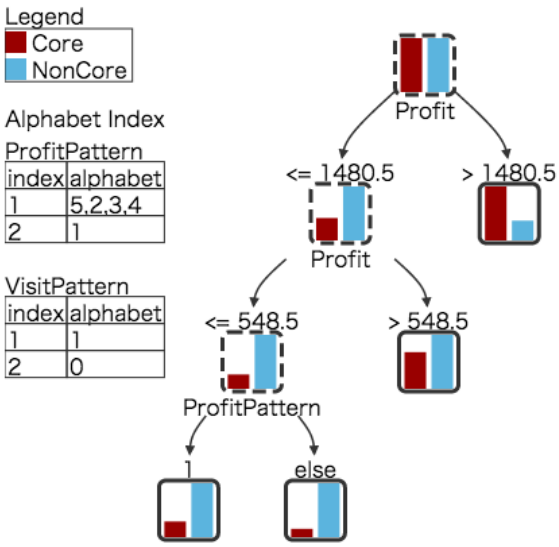

Figure 6: The decision tree as generated when α is set as 0.00001 for pruning with a minimum 1,500 samples in each leaf.

slightly from 4283 to 4265. The decision tree with the new tuning parameters is visualized in Figure 6.

```
~/parentdir$ ruby mbonsai/view/mdtree.rb i=result_core_mod/model.pmml \
> o=result_core_mod/tree.html -bar
```

6.2. Application II: Prediction of diaper purchase

In this application, we examine brand loyalty (brand A in this application) of customers by using a decision tree for analyzing brand purchase sequences. The objective is to predict who will stay loyal to brand A through continual purchasing of brand A diaper with a possible switch from size M to size L as the baby grows. There are seven major brands included in the data and they are denoted by A, B, C, D, E, F, and G. Among 1,838 customers who purchased baby diapers of size M from brand A at least four times, those who purchased baby diapers at least five times after switching from M to L are classified as loyal customers. Using this definition, 918 customers are labeled as loyal to brand A. Together with the labels, purchase patterns of M-sized diapers are used as training data to learn the decision tree model.

"BrandPattern" is represented by the string of diaper brands purchased in sequential order, which is used as a predictor variable to predict whether the customer will continue to purchase size L of brand A diaper after at least four purchases of size M diaper. A sample of the dataset is shown in Table 9.

Create a decision tree model

In this application, we encode the 7 brands in "BrandPattern" into a two-class alphabet index defined at `p=`. We then apply 5-fold cross-validation defined at `cv=`. The following **mbonsai** command is used to create the training model, followed by results of alphabet index and decision tree from `result_brand/model_1se.txt`:

BrandPattern	Target
CAAA	Loyal
AAAAAA	Loyal
AAAAAAAAAAF	Loyal
AACAAAAAAAAA	NotLoyal
ABBBBB	Loyal
...	

Table 9: Example of input data `brand.csv`. Each line corresponds to one customer, with the variable "Target" indicating the status of the continual purchase of diapers. "BrandPattern" is a predictor variable.

```
~/parentdir$ mbonsai/cmd/mbonsai i=data/brand.csv p=BrandPattern:2 \
> c=Target O=result_brand cv=5 seed=500
```

```
[alphabet-index]
Field Name: BrandPattern
Index[1]={A}
Index[2]={C,F,B,G,D,E}

[decision tree]
if($BrandPattern has 11)
  then $Target=Loyal (hit/sup)=(693/791)
  else if($BrandPattern has 1)
    then if($BrandPattern has 12)
      then $Target=NotLoyal (hit/sup)=(77/121)
      else $Target=Loyal (hit/sup)=(24/40)
    else $Target=NotLoyal (hit/sup)=(636/702)

numberOfLeaves=4
deepestLevel==3

[Confusion Matrix by Training]
## By count
        Loyal   NotLoyal        Total
Loyal   717     110     827
NotLoyal        114     713     827
Total   831     823     1654

## By cost
         Predicted As ...
        Loyal   NotLoyal        Total
Loyal   0       110     110
NotLoyal        114     0       114
Total   114     110     224

## Detailed accuracy by class
```

```
class,recall,precision,FPrate,F-measure
Loyal,0.866989,0.862816,0.137848,0.864897
NotLoyal,0.862152,0.866343,0.133011,0.864242

## Summary
accuracy=0.864571
totalCost=224

[Confusion Matrix by Estimation]
## By count
         Predicted As ...
        Loyal   NotLoyal        Total
Loyal   703     124     827
NotLoyal        111     716     827
Total   814     840     1654

## By cost
         Predicted As ...
        Loyal   NotLoyal        Total
Loyal   0       124     124
NotLoyal        111     0       111
Total   111     124     235

## Detailed accuracy by class
class,recall,precision,FPrate,F-measure
Loyal,0.85006,0.863636,0.13422,0.856795
NotLoyal,0.86578,0.852381,0.14994,0.859028

## Summary
accuracy=0.85792
totalCost=235

[Selected Alpha]
alpha: 0.00209438
```

Based on the results from the decision tree, brand A is indexed into 1, and brands C, F, B, G, D, and E are indexed into 2. The model rule states that if "BrandPattern" contains `11`, which corresponds to 2 consecutive purchases of brand A size M diapers, then the customer is likely to be a loyal customer of brand A – i.e., they would continue to use size L diapers from the same brand. The command `mdtree.rb` generates the decision tree which is visualized in Figure 7.

In this model, the accuracy is 0.8579. Note that in the `result_brand` directory, the files `model_info.csv`, `model_1se.csv`, `model_info_min.csv`, and `model_min.txt` are created in cross-validation mode. The accuracy of the training data can be compared across the four files to inspect the possible variation in estimates of accuracy. In addition, `predict.csv`, as well as `predict_1se.csv` and `predict_min.csv`, are also generated. The predictive accuracy on

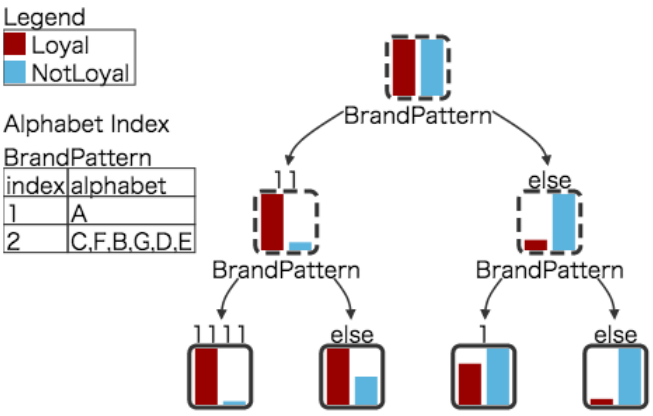

Figure 7: The decision tree for the brand pattern dataset.

the test set can be cross-checked using the three files. When the decision tree model is cross-validated against the test data, the classification appears to be both accurate and consistent across different partitions of data, suggesting that the model is stable and reliable.

The cross-tabulation results are saved in `predict_1se.csv` and `predict_min.csv` respectively in the `result_brand` directory. Statistical calculation of recall, precision, and F-measure are calculated from the predict data by the `mcm` command as shown below:

```
~/parentdir$ mbonsai/cmd/mcm i=result_brand/predict_1se.csv ac=Target \
> pc=predict O=result_brand/evalAcc_1se
```

```
[summary.csv]
evaluation,value
accuracy,0.8579201935
error rate,0.1420798065
total records,1654
unpredictable records,0

[class.csv]
Target,TP,FN,FP,TN,upCnt,upRate,recall,precision,f1
Loyal,703,124,111,716,0,0,0.8500604595,0.8636363636,0.8567946374
NotLoyal,716,111,124,703,0,0,0.8657799274,0.8523809524,0.8590281944

[confMatrix.txt]
        a       b       u
a       703     124     0
b       111     716     0

a:Loyal
b:NotLoyal
u:unpredictable
```

```
~/parentdir$ mbonsai/cmd/mcm i=result_brand/predict_min.csv ac=Target \
> pc=predict O=result_brand/evalAcc_min
```

```
[summary.csv]
evaluation,value
accuracy,0.8639661427
error rate,0.1360338573
total records,1654
unpredictable records,0

[class.csv]
Target,TP,FN,FP,TN,upCnt,upRate,recall,precision,f1
Loyal,708,119,106,721,0,0,0.8561064087,0.8697788698,0.8628884826
NotLoyal,721,106,119,708,0,0,0.8718258767,0.8583333333,0.8650269946

[confMatrix.txt]
        a       b       u
a       708     119     0
b       106     721     0

a:Loyal
b:NotLoyal
u:unpredictable
```

7. Explanation of output data

7.1. Output data

The data files generated by running the **mbonsai** command are summarized in Table 10. The main output files including `model.pmml`, `predict.csv`, `model_info.csv`, `alpha_list.csv`, and `param.csv` are described below.

`model.pmml` Based on the maximal tree of the decision tree created, the `complexity penalty` attribute is shown for each node. The branch would be pruned if α is greater than the value of `complexity penalty`. As the maximal tree and pruning information is recorded in PMML, different values of α can be used for prediction.

```
<Node id="0" score="Loyal" recordCount="15212" >
       <Extension extender="KGMOD" name="complexity penalty"
       value="0.218446"/>
             :
```

`predict.csv` The prediction result is added to the training data in CSV format. The prediction result, as described below, outputs the highest prediction probability in the column "predict", and the prediction accuracy for each class ("Loyal" and "NonLoyal" as shown below). When `ts=` is specified, it returns the prediction results of test data; when `cv=` is specified, it returns the prediction results of k-fold cross-validation, where k is user-specified.

File name	Content	Remarks
`model.pmml`	The decision tree model represented by PMML.	Records pruning information for maximum tree. Prediction mode is selected when `-predict` is specified.
`alpha_list.csv`	Other model information of the complexity parameter α.	Series of α corresponding to the series of models.
`model_min.txt`	Summary of pruned model with minimum classification prediction error.	Created when `cv=` or `ts=` is specified.
`model_1se.txt`	Summary of pruned model with the same 1SE rule.	Created when `cv=` or `ts=` is specified.
`model.txt`	Summary of pruned model for the specified α value.	
`model_info_min.csv`	Various information of pruned model with minimum classification prediction error.	Created when `cv=` or `ts=` is specified.
`model_info_1se.csv`	Various information of pruned model with the same 1SE rule.	Created when `cv=` or `ts=` is specified.
`model_info.csv`	Summary of pruned model for the specified α.	
`predict_min.csv`	The prediction information of pruned model with minimum classification prediction error.	Created when `cv=` or `ts=` is specified.
`predict_1se.csv`	The prediction information of pruned model with the same 1SE rule.	Created when `cv=` or `ts=` is specified.
`predict.csv`	The prediction pruned model for the specified α.	
`param.csv`	List of execution parameters.	Returns the pair of keyword-value for the specified parameters.

Table 10: List of output data from model building mode in **mbonsai**.

When `alpha=` is specified, the prediction result of the training data using the specified α value is returned.

```
ProfitPattern,VisitPattern,Profit,Visit,Target,predict,Core,NonCore
55552342,1100011101011,7969,11,Core,Core,0.7315679443,0.2684320557
525,1000001100000,5379,3,Core,Core,0.7315679443,0.2684320557
5,1000000000000,1538,1,Core,Core,0.7315679443,0.2684320557
31,1000000010000,91,2,Core,NonCore,0.2057237661,0.7942762339
52,1000001000000,1995,2,Core,Core,0.7315679443,0.2684320557
```

`model_info.csv` The file stores the model information in CSV format. The column "nobs" refers to the number of records in training data, column "alpha" refers to the value of pruning complexity parameter and "accuracy" and "totalCost" respectively refer to the percentage of correct answers in the test model and the total cost.

```
nobs,alpha,accuracy,totalCost
15212,0.01,0.7184459637,4283
```

`alpha_list.csv` The file stores the error rate, standard error, and error rate plus/minus one standard error corresponding to the α value of the pruning complexity parameter of the resulting decision tree. The following shows a snapshot from the file.

```
alpha,leafSize,errorRate,SE,up,lo
0,4164,0.02511175388,0.001268594376,0.02638034825,0.0238431595
1.200198434e-05,4152,0.02524322903,0.0012718252,0.02651505423,0.02397140383
1.469936877e-05,4142,0.02537470418,0.001275046945,0.02664975113,0.0240996572
1.756910609e-05,4122,0.02570339206,0.001283062043,0.0269864541,0.02442033002
...
```

`param.csv` The file contains the various parameter values used when building the model in CSV format.

8. Summary and remarks

In this paper, we have presented the implementation of the **mbonsai** software with extended analytical capability of decision trees. The **mbonsai** software is built upon existing tree algorithms published by of Quinlan (1993) and Olshen *et al.* (1984). A substantial body of literature in advancing the tree-based classification methodology can be found in Wu *et al.* (2008), Strobl, Malley, and Tutz (2009), Kuhn and Johnson (2013), and Loh (2014). Additionally, Hothorn (2018) provides a recent overview of open source R-based decision tree software programs.

mbonsai extends the decision tools C4.5 with advancements shown through several examples. For tree pruning and growing strategies, **mbonsai** closely follows CART, with the exception of using entropy instead of the Gini index that CART uses. The most innovative feature of **mbonsai** is its ability to handle multiple variables that contain sequence data. The graphical output from **mbonsai** is saved as a SVG file providing a tree map with classification distribution at each node. This also greatly enhances the presentation of results.

Although here we only illustrate **mbonsai** using transaction data, the program can be used in other areas where sequence data are available. For example, like its predecessor **BONSAI**, **mbonsai** can be applied to genetic data. Finally, as far as we know, this version of **mbonsai** is a unique attempt to directly analyze sequence data using tree-based methods. We envision future versions to include improvements such as bias correction and ensemble of trees.

Acknowledgments

The authors are grateful to the original **BONSAI** developing team, with special thanks to Naoki Katoh for improving the algorithm for **mbonsai**. The development of **mbonsai** was part of the ERATO Minato Discrete Structure Manipulation System Project, which was funded by the Japan Science and Technology Agency, Japan (PI: Shin-ichi Minato). The project was also

partially funded by the National Institute of Health Grants R01HL101066-01 (PI: Edward Ip) and UL1TR001420 (awarded to the Wake Forest Clinical and Translational Science Institute).

References

Breiman L (1996). "Bagging Predictors." *Machine Learning*, **24**(2), 123–140. doi:10.1007/bf00058655.

Console L, Picardi C, Theseider D (2003). "Temporal Decision Trees: Model-Based Diagnosis of Dynamic Systems On-Board." *Journal of Artificial Intelligence Research*, **19**, 469–512. doi:10.1613/jair.1194.

Freund Y, Schapire RE (1997). "A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting." *Journal of Computer and System Sciences*, **55**(1), 119–139. doi:10.1006/jcss.1997.1504.

Grubinger T, Zeileis A, Pfeiffer KP (2011). "**evtree**: Evolutionary Learning of Globally Optimal Classification and Regression Trees in R." *Journal of Statistical Software*, **61**(1), 1–29. doi:10.18637/jss.v061.i01.

Hamuro Y, Katoh N, Matsuda Y, Yada K (1999). "Mining Pharmacy Data Helps to Make Profits." *Data Mining and Knowledge Discovery*, **2**(4), 391–398. doi:10.1023/a:1009748731133.

Hothorn T (2018). *CRAN Task View: Machine Learning & Statistical Learning*. Version 2018-07-21, URL https://CRAN.R-project.org/view=MachineLearning.

Katoh N, Yada K, Hamuro Y (2003). "Business Application for Sales Transaction Data by Using Genome Analysis Technology." In *Discovery Science*, pp. 208–219. Springer-Verlag, Berlin.

Kawata H, Hamuro Y, Kato N, Yada K (2001). "Discovery of Business Opportunities from Purchase Transaction Data by Using Genome Analysis Technology." In *Annual Conference of the Japan Society of Artificial Intelligence Proceedings*.

Kim H, Loh WY (2001). "Classification Trees with Unbiased Multiway Splits." *Journal of the American Statistical Association*, **96**(454), 589–604. doi:10.1198/016214501753168271.

Kuhn M, Johnson K (2013). *Applied Predictive Modeling*. Springer-Verlag, New York.

Liaw A, Wiener M (2002). "Classification and Regression by **randomForest**." *R News*, **2**(3), 18–22. URL https://www.R-project.org/doc/Rnews/.

Loh WY (2014). "Fifty Years of Classification and Regression Trees." *International Statistical Review*, **82**(3), 329–348. doi:10.1111/insr.12016.

NYSOL Corporation (2014). ***NYSOL*** *Package Current Release: Version 2.0*. **NYSOL** Corporation. URL http://www.nysol.jp/.

Olshen R, Breiman L, Friedman J, Stone C (1984). *Classification and Regression Trees*. Wadsworth International Group, Belmont.

Quinlan JR (1993). ***C4.5**: Programs for Machine Learning*. Morgan Kaufmann, San Francisco.

Ridgeway G (2017). ***gbm**: Generalized Boosted Regression Models*. R package version 2.1.3, URL https://CRAN.R-project.org/package=gbm.

Rokach L, Romano R, Maimon O (2008). "Mining Manufacturing Databases to Discover the Effect of Operation Sequence on the Product Quality." *Journal of Intelligent Manufacturing*, **19**(3), 313–325. doi:10.1007/s10845-008-0084-6.

Rulequest (2013). ***C5.0** and **See5***. New South Wales, Australia. URL http://rulequest.com/see5-info.html.

Salford (2015). ***CART** Classification and Regression Trees*. San Digeo. URL http://www.salford-systems.com/products/cart.

Segal MR (1992). "Tree Structured Methods for Longitudinal Data." *Journal of the American Statistical Association*, **87**(418), 407–418. doi:10.2307/2290271.

Sela RJ, Simonoff JS (2012). "RE-EM Trees: A Data Mining Approach for Longitudinal and Clustered Data." *Machine Learning*, **86**(2), 169–207. doi:10.1007/s10994-011-5258-3.

Shimozono S, Shinohara A, Shinohara T, Miyano S, Kuhara S, Arikawa SY (1994). "Knowledge Acquisition from Amino Acid Sequences by Machine Learning System **BONSAI**." *Transactions of Information Processing Society of Japan*, **35**(10), 2009–2018.

Strobl C, Malley J, Tutz GH (2009). "An Introduction to Recursive Partitioning: Rationale, Application, and Characteristics of Classification and Regression Trees, Bagging, and Random Forests." *Psychological Methods*, **14**(4), 323–348. doi:10.1037/a0016973.

Thomas D, Fowler C, Hunt A (2013). *Programming Ruby 1.9 & 2.0: The Pragmatic Programmer's Guide*. The Facets of Ruby, 4th edition. The Pragmatic Bookshelf, Raleigh.

Wang Z (2018). ***bst**: Gradient Boosting*. R package version 0.3-15, URL https://CRAN.R-project.org/package=bst.

Wu X, Kumar V, Quinlan JR, Ghosh J, Yang Q, Motoda H, McLachlan GJ, Ng A, Liu B, Yu P, Zhou Z, Steinbach M, Hand DJ, Steinberg D (2008). "Top 10 Algorithms in Data Mining. Knowledge and Information Systems." *Knowledge and Information Systems*, **14**(1), 1–37. doi:10.1007/s10115-007-0114-2.

Yada K, Hamuro Y, Katoh N (2007a). "Data Mining Technique for Gene Analysis Makes Profits in the Supermarket." In AL Dixon, KA Machleit (eds.), *Marketing Theory and Applications: 2007 AMA Winter Educators' Conference*, volume 18, pp. 122–129. American Marketing Association.

Yada K, Ip EH, Katoh N (2007b). "Is This Brand Ephemeral? A Multivariate Tree-Based Decision Analysis of New Product Sustainability." *Decision Support Systems*, **44**(1), 223–234. doi:10.1016/j.dss.2007.03.014.

Yu Y, Lambert D (1999). "Fitting Trees to Functional Data, with an Application to Time-of-Day Patterns." *Journal of Computational and Graphical Statistics*, **8**(4), 749–762. doi:10.2307/1390825.

Zhang H, Singer B (2010). *Recursive Partitioning and Applications*. Springer-Verlag, New York. doi:10.1007/978-1-4419-6824-1.

Affiliation:

Yukinobu Hamuro
Institute of Business and Accounting
Kwansei Gakuin University
Nishinomiya, Hyogo, Japan
E-mail: hamuro@kwansei.ac.jp

Masakazu Nakamoto, Stephane Cheung
JST CREST Project
Kwansei Gakuin University
Osaka, Japan
previously:
JST ERATO Minato Discrete Structure Manipulation System Project
Hokkaido University
E-mail: nain@gmail.com, stephane.cheung@gmail.com

Edward Hak-Sing Ip
Department of Biostatistical Sciences
Department of Social Sciences and Health Policy
Translational Science Institute
Wake Forest School of Medicine
Winston-Salem, North Carolina
E-mail: eip@wakehealth.edu
URL: http://www.wakehealth.edu/Faculty/Ip-Edward-Hak-Sing.htm

Journal of Statistical Software http://www.jstatsoft.org/
published by the Foundation for Open Access Statistics http://www.foastat.org/
August 2018, Volume 86, Issue 6 *Submitted:* 2016-03-19
doi:10.18637/jss.v086.i06 *Accepted:* 2017-09-27