Mining Package Documentation

NYSOL Package version: 1.2,2.0

Revision history:
October 6, 2014 : Addition of mbonsai,mgpmetis.rb commands
January 17, 2014 : first release

July 27, 2015

Contents

Chapter 1

Introduction

1.1 Abstract

Data mining is the initial process in knowledge discovery to discover meaningful patterns from large data sets by applying data analysis technique such as statistics, pattern recognition, and artificial intelligence. Examples of data mining techniques includes extraction of association rules in "enumeration of frequent patterns", predict data category with the use of "classification", prediction of real value using "regression analysis", and classification of similar data with "clustering". This set data mining commands allow users to apply the mentioned approach in data analysis. In addition, a Ruby extension library is provided for processing of large-scale CSV data in Ruby, which reads input data in CSV format as it is done in MCMD.

1.2 Installation

M-Command supports the following operating system architectures.

- Mac OS X 10.7.5(Lion)
- Ubuntu Linux 12.04(32bit, 64bit)

Installation packages are available. Users would be able to install MCMD on systems with slight variation of the OS versions listed above. The software can be compiled and installed from the source code for installation in other OS.

1.2.1 Mac OS X

The mining gem package is included within the NYSOL package which can be downloaded from http://www.nysol.jp/en/home.

1.2.2 Ubuntu Linux

Download the latest NYSOL rpm package from http://www.nysol.jp/en/home.

1.2.3 Install from Source Code

The mining package is included within the NYSOL package. Follow the steps in the installation page to compile the program from source. http://www.nysol.jp/en/home/install.

Chapter 2

Mining Manuals

2.1 burst.rb - Burst Detection Command

Detection of burst state is used for analyzing a given series of data, Hidden Markov Model (HMM) is used as the algorithm. The phenomenon assumes probability distributions are used to describe the state transitions in two modes, namely steady-state and burst states, and returns the likelihood that maximizies all possible state sequences for the given data. Different types of probability distributions can be specified, such as Exponential distribution, Poisson distribution, Normal distribution, and Binomial distribution. Details is shown in the next section.

The input data shown in Table 2.8 is a numerical sequenced data (val item). Other fields (such as id) are not used for burst detection. Since the burst column from the input data is included in the output data (Table 2.2), thus, only include necessary items in the input data.

Table 2.1: Input

id	val
a01	1
a02	1
a03	4
a04	1
a05	1
a06	10
a07	7
a08	4
a09	5
a10	8
a11	12
a12	1
a13	1
a14	1
a15	6
a16	8
a17	2
a18	8
a19	2
a20	3
a21	4

Table 2.2: Poisson distribution burst

id	val	burst
a01	1	0
a02	1	0
a03	4	0
a04	1	0
a05	1	0
a06	10	1
a07	7	1
a08	4	0
a09	5	0
a10	8	1
a11	12	1
a12	1	0
a13	1	0
a14	1	0
a15	6	0
a16	8	1
a17	2	0
a18	8	1
a19	2	0
a20	3	0
a21	4	0

Table 2.3: Exponential distribution burst

id	val	burst
a01	1	1
a02	1	1
a03	4	1
a04	1	1
a05	1	1
a06	10	0
a07	7	0
a08	4	0
a09	5	0
a10	8	0
a11	12	0
a12	1	1
a13	1	1
a14	1	1
a15	6	0
a16	8	0
a17	2	0
a18	8	0
a19	2	0
a20	3	0
a21	4	0

Table 2.4: Normal distribution burst

id	value	burst
b01	1	0
b02	-4	0
b03	-2	0
b04	1	0
b05	1	0
b06	10	0
b07	7	0
b08	2	0
b09	5	0
b10	8	1
b11	10	1
b12	1	0
b13	1	0
b14	1	0
b15	7	0
b16	-8	-1
b17	-3	-1
b18	5	0
b19	1	0
b20	1	0
b21	1	0

Assume that the number of events in this data series is measured based on time factor (for example, the number email arrived per hour), burst detection can be generated from Poisson distribution (Table 2.2). In addition, given intervals between events (For example, the interval of seconds of arrivals of mail), burst detection is typically generated from a exponential distribution (Table 2.3). Furthermore, given an error series (such as stock price trends), burst detection could be generated from normal distribution (Table 2.4). Thus, it is possible to use the same input data to carry out different burst detections depending on the distribution methods. The choice of distribution method is determined by the application to the problem.

Format

```
burst.rb i= f= dist= [o=] [d=] [s=] [p=] [param=] [pf=] [n=] [nf=] [v=] [nv=] [--help]
```

Note 1

There are three ways to define parameters of distribution at steady-state as follows.

1. Specify the value at `param=`.

`i=` : Input file name [required paramenter]
`o=` : Output file name [optional: to standard output by default]
`d=` : Debug information file [optional]
`dist=` : Type of distribution (exp:exponential distribution,poisson:Poisson distribution,gauss:normal distribution, binom:binomial distribution) [required parameter]
`f=` : Numeric field name of the target for burst detection (field names in i=) [required parameter]
`param=` : Parameters of distribution at steady-state. See note 1 [optional]
`pf=` : Field name(s) of the parameters of distribution at steady-state (field names in i=) See note 1 [optional]
`s=` : Burst scale (extreme burst can be detected if this value is increased. See note 1 for details) [optional : default value=2.0]
`p=` : Probability of same state transition (it is difficult to transition to a different state by increasing this value. See note 2 for details) [optional : default value =0.6]
`n=` : Number of trials when dist=binom [Specify n= or nf=]
`nf=` : Field name of the number of trials when dist=binom
`v=` : Variance value when dist=gauss (if the value is not specified, the default value is estimated from the data where the fields is defined at f=)
`nv=` : Field name of the variance when dist=gauss
`--help` : Display help information

2. Use the value from the field specified at `pf=`. This assumes that the time dependent on the parameter is different.
3. If `para=` and `pf=` is not specified, the calculation is automatically derived from the value specified in `f=`.

The method of calculation from the data as mentioned in the third method above differs according to different distribution. Nevertheless, S is the value specified by `s=`, n= is the number of data, x_i is the i-th row of value of the item specified by `f=`.

Distribution	Probability (density) function	Parameter (par)	Steady-state par	Burst state par
exp	$f(x) = \lambda e^{-\lambda x}$	λ:Average number of events	$\lambda_0 = N/\sum_i x_i$	$\lambda_1 = S\lambda_0$
poisson	$f(x) = \frac{\lambda^x e^{-\Gamma}}{x!}$	λ:Average number of events	$\lambda_0 = \frac{1}{N}\sum_i x_i$	$\lambda_1 = S\lambda_0$
gauss	$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(x-\mu)^2/2\sigma^2}$	μAverage	$\mu_0 = \frac{1}{N}\sum_i x_i$	$\mu_\pm = \mu_0 \pm S\sqrt{\sigma^2}$ *
binom	$f(x) = {}_nC_x p^x (1-p)^{n-x}$	p:Success probability	$p_0 = \frac{1}{Nn}\sum_i x_i$ **	$p_1 = S/(\frac{1-p_0}{p_0} + S)$

$^*\sigma^2 = \frac{\sum (x_i - m)^2}{N-1}$
$^{**}n$ is specified by `n=`.

Note 2

When p is assumed as the probability specified by `p=`, the probability of state transition is set as follows.

Table 2.5: State transition probability of exp, poisson, binom

	steady-state	burst
steady-state	p	$1-p$
burst	$1-p$	p

Table 2.6: State transition probability of ingauss

	burst-	Steady-state	burst+
burst-	p	$\frac{2}{3}(1-p)$	$\frac{1}{3}(1-p)$
steady-state	$\frac{1}{2}(1-p)$	p	$\frac{1}{2}(1-p)$
burst+	$\frac{1}{3}(1-p)$	$\frac{2}{3}(1-p)$	p

Explanation

Formulation

HMM (Hidden Malkov Model) is a probabilistic model that is built on the assumption of Markov process with hidden state that can not be observed directly. HMM is comprised of stochastic state transition model and data generation model. The observed data series follow the data generation model in hidden state.

Time t observed in data x_t will be modelled according to the probability distribution$p(x_t|z_t;\boldsymbol{\phi})$ that is defined hidden state $z_t \in \{1, 2, \cdots, K\}$. $\boldsymbol{\phi}$ is the vector parameter of generation model, with the assumption that it is constant and does not depend on t. Furthermore, hidden state z_t transitions depend on the previous state z_{t-1}, the probability distribution is shown in $p(z_t|z_{t-1};\mathbf{A})$.

$\mathbf{A} = \{a_{i,j}|i, j = 1, 2, \cdots, K\}$ is the transition probability table from state i to j with the assumption that it is constant and does not depend on t. However, in $\sum_j a_{i,j} = 1.0$, the initial state z_1 is assumed to follow the probability vector $\boldsymbol{\pi}$.

From the above, the joint probability of the observed data series $X = x_1, x_2, \cdots, x_N$ and state series $Z = z_1, z_2, \cdots, z_N$ are given by the formula (2.1).[1]

$$p(\mathbf{X}, \mathbf{Z}; \boldsymbol{\pi}, \mathbf{A}, \boldsymbol{\phi}) = p(z_1; \boldsymbol{\pi}) \left[\prod_{i=2}^{N} p(z_i|z_{i-1}; \mathbf{A}) \right] \prod_{j=1}^{N} p(x_j|z_j; \boldsymbol{\phi}) \tag{2.1}$$

In burst detection, $K = 2$, i.e. assuming two states: steady state and burst, the observable data series can be obtained from the same probability distribution with different parameters in each state.

In the burst detection problem, given the parameter $\boldsymbol{\pi}, \mathbf{A}, \boldsymbol{\phi}$, find $\mathbf{Z}^*$ to maximize the joint probability shown in the formula(2.1) at the time of observing the data series $\mathbf{X}$ (formula(2.2)).

$$\mathbf{Z}^* = \underset{\mathbf{Z}}{\operatorname{argmax}}\ p(\mathbf{X}, \mathbf{Z}; \boldsymbol{\pi}, \mathbf{A}, \boldsymbol{\phi}) \tag{2.2}$$

Burst detection example of email

Corresponding to the above formula, the following example explains burst detection in relation to the number of emails arrived (Table 2.8). The objective here is to seek the hidden state sequence $\mathbf{Z} = z_1, z_2, \cdots, z_N$ $(z_i \in \{0, 1|0 is steady-state, 1 is burst state\})$ from the numeric sequence $\mathbf{X} = 1, 1, 4, \cdots$ shown in Table 2.8.

Consider the number of email arrivals as random variable, it will be appropriate to assume as Poisson distribution. Poisson distribution takes the average number of arrivals λ as parameter, and the probability function represent by the formula (2.3).

$$f(x) = \frac{\lambda^x e^{-\Gamma}}{x!} \tag{2.3}$$

The parameter λ_0 in steady-state is defined by the command parameters at `param=` or `pf=`. When the parameters are not specified, the parameter is specified by the average arrivals from the data specified at `i=`. The calculation result of the average value of the field `val` in Table 2.8 is $\lambda_0 = 4.29$. Further, the parameter λ_1 in burst state is set as 2 times the normal state (8.58) unless otherwise specified. The value can be changed by defining the value at `s=`.

From the above, the probability distribution of the state of the parameter vector is $\boldsymbol{\phi} = (4.29, 8.58)$. Table 2.7 shows the probability of the numeric value of data series $\mathbf{X}$ from each state. The probability of steady-state for numeric value "1" and "4" is high, and the probability of burst state for "10" is high.

Next, let's consider state transition probability $p(z_i|z_{i-1};\mathbf{A})$. There are four combinations of transition in the two state. In this command, it is possible to provide the same probability values for the transition probabilities between the same states, the default value is 0.6 unless otherwise stated. In other words, $a_{0,0}, a_{1,1} = 0.6$. Further, the transition probabilities of different states are calculated by $a_{0,1}, a_{1,0} = 0.4$. The state transition probability $\mathbf{A}$ is shown in the formula below (2.4).

$$\mathbf{A} = \begin{pmatrix} a_{0,0} = 0.6 & a_{0,1} = 0.4 \\ a_{1,0} = 0.4 & a_{1,1} = 0.6 \end{pmatrix} \tag{2.4}$$

Table 2.7: Probability of each state in Poisson distribution

id	val	steady-state($\lambda = 4.29$)	burst($\lambda = 8.58$)
a01	1	0.0590	0.0016
a02	1	0.0590	0.0016
a03	4	0.1935	0.0426
a04	1	0.0590	0.0016
a05	1	0.0590	0.0016
a06	10	0.0079	0.1117
a07	7	0.0725	0.1278
a08	4	0.1935	0.0426
a09	5	0.1658	0.0730
a10	8	0.0389	0.1369
a11	12	0.0011	0.0622
a12	1	0.0590	0.0016
a13	1	0.0590	0.0016
a14	1	0.0590	0.0016
a15	6	0.1185	0.1043
a16	8	0.0389	0.1369
a17	2	0.1264	0.0070
a18	8	0.0389	0.1369
a19	2	0.1264	0.0070
a20	3	0.1806	0.0199
a21	4	0.1935	0.0426

The probability vector of the initial state at the end is $\pi = (1.0, 0.0)$, assuming that the initial state is a steady-state. From the above, the parameters $\mathbf{A}, \boldsymbol{\phi}, \boldsymbol{\pi}$ of the formula(2.1) are met. Next, find out the series of state $\mathbf{Z}^*$ to maximize the formula(2.1).

Given the size of the data series $\mathbf{X}$ is 21, there are about $2^{21} = 2million$ combinations of state sequences for consideration. The optimal solution of long sequences can not be solved by brute force. Viterbi algorithm is a dynamic programming method for finding the most likely sequence of hidden states. Refer to specialized textbooks for detailed theoretical explanation, however this method generates state sequence as shown in Table 2.2.

Examples

Example 1 Example from the "Explanation" section above

```
------------------------------------------------
# inp1.csv
id,val
a01,1
a02,1
a03,4
a04,1
a05,1
a06,10
a07,7
a08,4
a09,5
a10,8
a11,12
a12,1
```

```
a13,1
a14,1
a15,6
a16,8
a17,2
a18,8
a19,2
a20,3
a21,4

$ burst.rb i=inp1.csv f=val dist=poisson o=out1.csv

# out1.csv
id,val,burst
a01,1,0
a02,1,0
a03,4,0
a04,1,0
a05,1,0
a06,10,1
a07,7,1
a08,4,0
a09,5,0
a10,8,1
a11,12,1
a12,1,0
a13,1,0
a14,1,0
a15,6,0
a16,8,1
a17,2,0
a18,8,1
a19,2,0
a20,3,0
a21,4,0
-----------------------------------------------
```

2.2 mnb.rb Naïve Bayesian Classifier

Build a probability classifier model by applying Bayes' theorem using supervised learning. The probability of the class variable is calculated based on the presence of a particular feature of the item, and returns the predicted class from the test data in the output. The model used Multinomial Naive Bayes to handle the frequency distribution of the item. However, the probability becomes zero if the new item only appeared during validation stage. The problem can be avoided by applying Laplace smoothing. Table 2.8 shows an excerpt of training data which is used as input data. "Transaction ID" is used to identify each transaction, and each transaction must contain "item", "frequency" and "class information". In Table 2.8, the transaction fields correspond to "id", "word", "freq", "class". Table 2.9 shows an example of validation (testing) data. In the validation data, the class field is not included, the class is predicted from the Naive Bayes model constructed with the training data.

Table 2.8: Training data(train.csv)

id	word	freq	class
1	w1	2	M
1	w2	4	M
2	w1	1	M
2	w2	2	M
2	w3	3	M
4	w1	3	M
4	w2	3	M
4	w3	2	M
5	w1	1	F
6	w1	2	F
6	w2	1	F

Table 2.9: Validation data(test.csv)

id	word	freq
3	w2	8
3	w3	2
7	w1	1
7	w2	3

Format

```
mnb.rb k= f= w= c= i= o= [I=] [O=] [-c] [-debug] [--help]
```

```
i=       : File name of training data [required parameter].
o=       : Output file name [required parameter].
k=       : Field name of transaction ID (i= & I=on field name) [required parameter].
f=       : Field name (i= & I=on field name) [required parameter].
w=       : Field name of weight (i= & I=on field name) [required parameter].
c=       : Field name of class (class information) [required parameter].
I=       : File name of testing data [optional].
O=       : Output file name of testing data [required parameter].
-c       : Execute with ComplementBayes [optional].
-debug   : Debug mode (Returns the detailed message in output, current output of work file is not removed).
```

Noten1

The field names in the training data and the test data must be specified at k=, f=, w=.

Comment

Naïve Bayes model is a probability model based on Bayes' theorem with independence assumptions. Consider the vector with particular features $\mathbf{w} = (w_1, w_2, \cdots, w_n)^\top$ where the appearance of the feature is represented

by the value $w_i = 0, 1$. The probability $p(c|\mathbf{w})$ of each class c, with the presence of feature expressed in $\mathbf{w}$ is represented in the Bayes' theorem formula shown in (2.5) .

$$p(c|\mathbf{w}) = \frac{p(\mathbf{w}|c)p(c)}{p(\mathbf{w})} \tag{2.5}$$

The denominator $p(\mathbf{w})$ is constant regardless of the value of variable c. Therefore, looking at the numerator, $p(c)$ is the posterior probability of class c. The posterior probability $p(c|\mathbf{w})$ is updated when this probability of occurrence of the feature vector for the specific class has the likelihood of $p(\mathbf{w}|c)$. This refers to the Bayes' Theorem.

If there are more dimensions of $\mathbf{w}$, it becomes difficult to estimate the joint probability $p(\mathbf{w}|c)$ for a single character. As shown in the formula (2.6), naive Bayes method calculates $p(\mathbf{w}|c)$, with the naive assumptions that the occurrence of all words is independent of each other.

$$p(\mathbf{w}|c) = \prod p(w_i|c) \tag{2.6}$$

From the above, $p(c|\mathbf{w})$ is represented by the formula (2.7). In order to build a classifier using Maximum-a-Posterior (MAP) probability decision rule which adopts the most plausible hypothesis, the estimated class $\hat{c}$ can be determined by the formula (2.8).

$$p(c|\mathbf{w}) \propto \sum_i \ln p(w_i|c) \tag{2.7}$$

$$\hat{c} = \underset{c}{\operatorname{argmax}}\ p(c) \sum_i \ln p(w_i|c) \tag{2.8}$$

When the naive Bayes classifier is applied in practice, words or items are used as feature vectors. These include cases with frequency information such as occurrences of the number of words and the number of items purchased.

Therefore, assuming that the element f_i of the vector with particular feature vector $\mathbf{f}$ has a frequency of occurrence with a feature value of i, each estimated class $\hat{c}$ is shown in the equation (2.9), calculated by multiplying the frequency of likelihood f_i. The above is known as Multinomial Naïve Bayes model.

$$\hat{c} = \underset{c}{\operatorname{argmax}}\ p(c) \sum_i f_i \ln p(w_i|c) \tag{2.9}$$

Problem of Zero Frequency

If the combination of the class and feature value is not present in the training examples, the probability estimation becomes zero, consequently, the product used for multiplication becomes zero. In order to avoid the problem of zero frequency, modify the estimated probability value with smoothing, and adjust the probability values of all combinations so that it will not become zero. In this case, Laplacian smoothing is applied by adding 1 to the number of occurrences to the feature value.

Complement Naïve Bayes

In Naive Bayes classifier, Complement Naive Bayes refers to an extension to learn a the model by using a complementary set that does not belong to any particular class. When predicting a class using the vector with features which do not belong to any class, the probably of the unclassified vector will be assigned to the lowest class. When classifying two values, it is not meaningful since the same result will be returned, however, the effect is more prominent when there are a lot of class dispersion in a multi-class classification problem. If -c option is specified, it is executed as Complement Naïve Bayes.

An Example to Determine the Gender Author

This example uses the above data to determine the single character gender information contained in the author's proposal submission data.

Table 2.8 shows the vectors containing the feature "word" in the training data, and build a naive Bayes model to determine the gender as specified in the "class" field, and the objective is to predict whether the class is M or F for each unique id in the validation data in Table 2.9.

When running the command (mnb.rb) below with train.csv and test.csv, 2 files namely out.csv and test_out.csv will be returned as output. The prediction probabilities of F and M are contained in field names F,M corresponding to each id in out.csv. The class with higher probability value is returned as the predicted class (predictCls) in the output.

In this example, class and PredictCls are in complete agreement, thus, the accuracy rate of the training data is 100%. The resulting naive Bayes model is built as shown in nb_test_out.csv, Table 2.9 shows the predicted class (predictCls) obtained from the feature "word" from each vector. Both vectors with id 3 and 7 has a predicted class as M (Male).

```
-----------------------------------------------
$ more train.csv
id,word,freq,class
1,w1,2,M
1,w2,4,M
2,w1,1,M
2,w2,2,M
2,w3,3,M
4,w1,3,M
4,w2,3,M
4,w3,2,M
5,w1,1,F
6,w1,2,F
6,w2,1,F

$ more test.csv
id,word,freq,class
3,w2,8,M
3,w3,2,M
7,w1,1,F
7,w2,3,F

$ mnb.rb i=train.csv I=test.csv O=test_out.csv o=out.csv k=id f=word w=freq c=class
#MSG# naiveBayes start; 2013/12/31 23:59:59
#END# nb.rb i=train.csv I=test.csv O=test_out.csv o=out.csv k=id f=word w=freq c=class; 2013/12/31 23:59:5

$ more out.csv
id,F,M,class,predictCls
1,0.4689127516,0.5310872483,M,M
2,0.4296687087,0.5703312913,M,M
4,0.4748995053,0.5251004949,M,M
5,0.5353401796,0.4646598204,F,F
6,0.5309945758,0.4690054242,F,F

$ more test_out.csv
id,F,M,predictCls
3,0.3993866359,0.6006133641,M
7,0.4451382443,0.5548617557,M
```

2.3 mbonsai Decision Tree Generated from Sequence Data

This command builds a decision tree model based on sequence patterns. Analysis of the sequence data can be applied in a variety of applications such as order of brands purchased by customers, sales floor cyclic patterns in department stores and supermarkets, the onset order of injuries. The original idea of this command, BONSAI[4], was developed by a research team from Kyushu University and Kyushu Institute of Technology. This technique is applied for the analysis of amino acid sequences in molecular biology. Similarly, this command implemented several improvements for the application of sequence analysis in business data with the following features.

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

First, the example below illustrates how to obtain an intuitive understanding of this command. A data set contains purchase sequences of brand `a,b,c` by customers from a retail store, and the corresponding churn status of the customer is shown in Table 2.10. Under the hypothesis of which purchase order of the brand is related to the estrangement of the customer, we will build a decision tree model with partial pattern of brand purchasing order as explanatory variable, and the churn status as target variable (Refer to the next section on details of the definition of patterns that is considered partial string, such as "ab", "cbb"). Pattern used as explanatory variables is referred to as candidate pattern, the pattern generated (More details on this method is available in the next section) will contribute to the accuracy of the model, afterwards, the patterns are converted to 0-1 variable (Table 2.11). Decision tree is constructed in a conventional manner from this data set consisting of candidate patterns. The actual decision tree created with this command is shown in Figure 2.12.

Table 2.10: Example of sequence data. Each line corresponds to one customer, with the target variable indicating the status of continual purchase of the customer. The alphabetic characters a, b, c, shown in "BrandSequence" column represents the brand purchased by the customer in the respective order as shown.

BrandSequence	Churn
bcaba	yes
bcabcaa	yes
aaabac	yes
caa	yes
cca	no
cacbc	no
bcc	no
acca	no

Table 2.11: The BrandSequence column is extracted from Table 2.10 and matched against partial purchase patterns as explanatory variables (referred to as candidate patterns). All samples are matched,, the presence of patterns is converted to 0-1 formatted data. In this example, the first record contains the partial pattern "a", "b", "c", "ab" in the sequence, however, " aa " and " cc " is not included.

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

A notable feature of the BONSAI is the function to group elements (known as alphabet) that make up the pattern automatically. The grouping of each alphabet is referred to as index.

For example, there are three ways to group the 3 brands `a,b,c` into two index (a and bc, b and ac, c and ab), each grouping is replaced with the matching index. Three decision trees are built with the above described procedure, the best decision tree is selected based on classification accuracy.

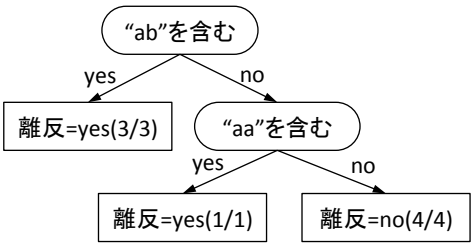

Table 2.12: Example of Building Customer Churn Model with BONSAI

アルファベット	a	b	c
インデックス	1	2	2

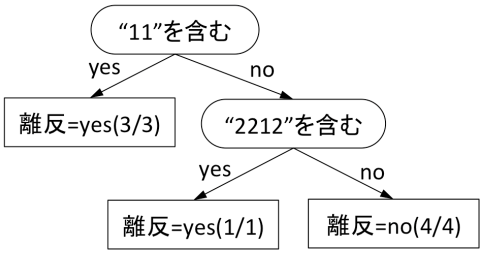

Table 2.13: Example of Building Customer Churn Model with BONSAI

Decision tree built in this manner is shown in Figure 2.13. The result of grouping with index increases the accuracy of the classification model, further more, useful knowledge could also be obtained from the result.

2.3.1 Details

The details concerning the construction of the decision tree are explained as follows.

Regular pattern

Given, n character string constants $\pi_1, \pi_2, ..., \pi_n$ on alphabet Σ, with any of the $n+1$ character string $x_0, x_1, ..., x_n$, regular pattern (also referred to as sequence pattern) is represented in the format $x_0\pi_1x_1\pi_2x_2\cdots\pi_nx_n$. This technique is easier to understand when the concept of wildcard is applied to the field of data retrieval (“*” is used instead of x_i as wildcard). In this command, besides the definition of regular pattern described above, that is $x_0\pi x_1$, the regular pattern can be handled by restricting to the substring π (hereinafter referred to as ”string pattern”). The use of string pattern and sequence pattern is specified in the second parameter at `p=`.

Begin / End Match

It is possible to specify the matching of beginning and ending string as matching rule for regular pattern of each sample of sequence data. This can be specified in the fourth (match beginning), and fifth (match ending) parameters of `p=`. Based on this specification, the position of the alphabet of the beginning (or ending) of regular pattern, the beginning (or ending) of sequence data is matched within the specified number of characters. For example, given the sequence data `aabccd`, when matched with character substring pattern `ab` from the left, if the matching sequence only contains 1 character, it is not considered as match, if the beginning of the sequence contains first two characters of the substring pattern, it is considered as match. If the sequence pattern is made up of `ab`, it is possible to match the first charter in the beginning. If this parameter is not specified, it operates without restriction of matching as described above.

Alphabetical Order

This command can deal with order structure that contains alphabets. The key difference of order structure is the rules for creating the index. Given the alphabet set $\{a_1, a_2, \ldots, a_n\}$, with the relationship of $a_1 \prec a_2 \prec \ldots \prec a_n$ for any three consecutive alphabet a_i, a_{i+1}, a_{i+2}, an index is generated to satisfy the relationship of $\psi(a_i) = \psi(a_{i+2}) \Rightarrow \psi(a_i) = \psi(a_{i+1})$. Here, $\psi(a)$ represents an index to be associated with the alphabet a. This means that the alphabets belonging to a group for indexing must always follow consecutive order. For

example, given three consecutive alphabet $a \prec b \prec c$, $\{a, b\} and \{c\}$ is grouped together, but $\{a, c\} and \{b\}$ is not. Consecutive alphabet can be specified in the third parameter of `p=`.

Local Search and Search Space of Optimal Alphabet Index

Given n number of alphabet(s), the number of cases to be grouped into m or less indexes is not certain, but when $m = 2$ is set as limit, the number can be presented in m^{n-1} ways. If both the value of m, n are small, it is possible to construct a decision tree for all cases, and if the value is large, the local search method of searching in subspace is applied. Initially, the random index is set to corresponding alphabet, the corresponding relationship changes bit by bit when building the decision tree, and continue to change until there is no improvements in classification accuracy (Refer to literature [4] for details). Thus, sometimes the results may differ depending on the initial value of the corresponding relationship between the index and the alphabet. In order to obtain the same results from the initial value and select the best model (multi-start), the initial value can be specified at `iter=` in this command. The default value is `iter=1`.

Method of Generating Candidate Patterns

Candidate patterns as shown in Table 2.11 are generated from the updated alphabet index by local search. The generation of the classification rules for the candidate patterns is based on the nodes of the decision tree. In this command, candidate patterns are enumerated by the heuristics method described below. First, the index for the regular pattern with length 1 is constructed, and is stored in the priority queue. Precedence in the priority queue is determined by ascending order (see below) of the entropy of a regular pattern Afterwards, select the regular pattern with the lowest entropy in the priority queue, an index is added to the selected regular pattern, regular pattern with length of 2 is created, and stored in priority queue again. Repeat the above steps such that, regular pattern with length n is selected, and regular pattern with length $n + 1$ is stored in the priority queue. However, the index cannot be added when n is greater than 5 (the upper size limit of the regular pattern can be specified in the sixth parameter of `p=`). In addition, the process will terminate if the size is greater than the number of candidates (specified by `cand=`) specified by the user. Entropy is used to evaluate regular patterns and is also used in the selection of the splitting rules at the nodes of the decision tree, the value of entropy becomes smaller as the regular patterns are classified into extreme distribution classes. Entropy $ent(\pi)$ of regular pattern π is defined in Equation 2.10.

$$ent(\pi) = -q^{\mathrm{m}(\pi)} \sum_{i=1}^{c} p_i^{\mathrm{m}(\pi)} \log p_i^{\mathrm{m}(\pi)} - q^{\mathrm{u}(\pi)} \sum_{i=1}^{c} p_i^{\mathrm{u}(\pi)} \log p_i^{\mathrm{u}(\pi)} \tag{2.10}$$

Here, c represents the number of classes, $p_i^{\mathrm{m}(\pi)}(p_i^{\mathrm{u}(\pi)}$ represents the composition ratio of class i that matches (or unmatch) with the sample in the regular pattern π expressed as $(\sum_i^c p_i^{\mathrm{m}(\pi)} = 1)$. In addition, $q^{\mathrm{m}(\pi)}(q^{\mathrm{u}(\pi)})$ represents the composition ratio of matching (or unmatch) with regular patterns π out of all samples $(q^{\mathrm{m}(\pi)} + q^{\mathrm{u}(\pi)} = 1)$.

Other Types of Variables

In this command, besides sequence data, numerical and categorical variables can be specified as an explanatory variable. By doing so, it is possible to construct decision tree rule that includes regular patterns with categorical and numerical rules. The creation of branch rules based on category and numbers is similar to that in C4.5 [5]. In this command, columns with sequence, numeric, and categorical data can be defined at `p=,n=,d=` parameters.

Selection of Splitting Rule

This command adopts a top down greedy method for the splitting of branches. In other words, splitting rules at the node of the tree is determined according to the evaluation criteria from the information in the node. The evaluation criteria is based on the selection of branching rules which maximizes the entropy gain. For samples that have been classified into certain nodes, the probability (composition ratio) of class i is represented by p_i, the entropy at node is calculated by $ent = -\sum_{i=1}^{c} p_i \log p_i$. At node p_i, number of samples n are classified into class i, the ratio of classified sample n_i can be calculated by n_i/n. Equation 2.10 shows the entropy gain from the difference of entropy $ent(\pi)$ after splitting the regular pattern π. It means by splitting the regular pattern π, how much entropy is reduced. The procedure is repeated until all samples are classified into respective classes at the node, and the tree can no longer be grown bigger. This type of decision tree is referred to as maximal tree.

Pruning

Construct a maximum tree T_{max} according to the method shown in the previous section, but since the tree size is larger than normal tree, the training data tends to overfit the model, where the classification accuracy of training data is high (misclassification rate decreases), but prediction accuracy of test data decreases. In order to avoid this problem, a small section (including the root node) of the maximal tree is removed during pruning that may be based on noisy or erroneous data.

In consideration to the set of nodes t from decision tree T, where the maximal tree is denoted by $T_{max} = \{t_1, t_2, \ldots, t_k\}$, node t and root node of the subtree is represented by T_t, the subtree of node $t \in P$ to root node is pruned (replaced with leaf node) from the decision tree expressed as $T_{max} - \bigcup_{t \in P} T_t + P$. The misclassification rate of decision tree T is denoted by $R(T)$, the pruning is based on the selection of subtree T^* with the lowest misclassification rate on unknown data. It is not possible to determine the actual misclassification rate for unknown data of T, instead, it is estimated based on the training data. Given the prediction amount is $C(T)$, the formula of pruning is expressed in the formula 2.11.

$$_{P \subseteq T_{max}} C(T_{max} - \bigcup_{t \in P} T_t + P) \tag{2.11}$$

To solve this problem, this command applies the cost-complexity pruning method [6]. This method is comprised of two key phases. First, a series of subtrees $T_1 \supset T_2 \supset \ldots \supset T_k$ is selected which is nested from the maximal tree build from the training data (here T_1 is the maximal tree, T_k is the root node). Next, the accuracy of these trees is estimated by test sample using cross-validation, the tree with highest prediction accuracy is selected.

When selecting the series of subtree, the evaluation function of decision tree T measuring the cost complexity is defined as $R_\alpha(T) = R(T) + \alpha|\tilde{T}|$, the decision tree model is considered better if the value is smaller. This equation is obtained by balancing the misclassification rate of decision tree $R(T)$, and the tradeoffs of decision tree complexity $|\tilde{T}|$ (number of leaf nodes in T) where the complexity parameter is $\alpha(\geq 0)$. In the training data, when the tree size grows larger, the misclassification rate $R(T)$ decreases monotonically, and yet, the complexity $|\tilde{T}|$ increases monotonically. When α is adjusted, the priority of misclassification rate and complexity is also adjusted accordingly. Here, α is fixed as a single value, the pruning process find the subtree $T(\alpha)$ that minimizes the function $R_\alpha(T)$. Further, $T(\alpha)$ continues to minimize the tree when α increases, it is possible to enumerate α corresponding to $T(\alpha)$, as a result, the nested subtree $T_1 \supset T_2 \supset \ldots \supset T_k$ is created (Refer to [6] for more details). Here, the subtree T_i consist of the smallest size decision tree with the minimal cost complexity $\alpha \in (\alpha_i, \alpha_{i+1}]$. In the above series of nested subtree, the corresponding cost complexity parameter $\alpha_1, \alpha_2, \ldots, \alpha_k$ can be obtained.

Next, among the series of subtree obtained from the above method, select the optimal subtree $\hat{T}^*$ with the minimum prediction misclassification rate when applied to unknown data. In this command, users can select between test sample method (specified by `ts=`) or cross-validation (specified by `cv=`). In test sample method, training data D is partitioned into two sets D_1, D_2 at a ratio of$1:2$. Then, the maximal tree based on D_2 training data is built, based on the complexity parameter $\alpha_1, \alpha_2, \ldots, \alpha_k$ obtained from the previous phase, the

corresponding subtree is selected. From this subtree, D_1 is used as the set of unknown data to predict the value of misclassification rate. In cross-validation method, training data D is partitioned equally into n sets as $D_1, D_2, \ldots, D_n$. First, D_1 is treated as unknown data for prediction, similarly, this applies to other training data using the test sample method. Similar process is applied n times to the remaining test data $D_1, D_2, \ldots, D_n$, then all training data D is applied as one set for validation. Thus, the average misclassification rate is obtained by this method. From the results obtained above, complexity parameter $\alpha_1, \alpha_2, \ldots, \alpha_k$ for decision tree $T_1, T_2, \ldots, T_k$, the optimal decision tree with the lowest misclassification rate is selected. In addition, the smallest tree whose estimated mean error rate is within one standard error of the estimated mean error rate of the best tree selected (this is called " 1 SE rule ").

Nevertheless, pruning can be done by directly specifying the α value without using the cross-validation or test sample method. A decision tree can be built at high speed without having to process calculations for prediction, but the drawback is that the specified α value is arbitrary.

Actual Pruning

In the model building mode, pruning can be done by specifying any of the parameters `ts=`, `cv=`, `alpha=` in this command. When `cv=` or `ts=` is specified, test data is used for prediction, the model with the minimum misclassification rate is selected. In addition, when `alpha=` is specified, the pruned model corresponding to the specified α value is selected. The selection of the model based on the evaluation of the decision tree model is saved in `model.txt` and `model_info.txt`. However, regardless of any specification, all α corresponding to the maximal tree is calculated internally, at prediction mode (`-predict`), if `alpha=` is specified, it is possible that different models are used for prediction.

Learning Cost Considerations

When applying classification model, instead of increasing classification accuracy (percentage of correct answers), there are many instances when it is useful to minimize the cost of misclassification. Cost is used when predicting customer churn, there are various considerations when implementing cost sensitivity, such as consideration of the cost of customer who stayed who is predicted as customer who left. Construction of model that takes misclassification costs into consideration is known as cost sensitive learning. Various methods have been proposed, the method proposed by Breiman et al [6] is used in this case. This method is used for the calculation of the branch rule, probability p_i of class i in the sample is therefore modified by assigning cost calculation. Now, the cost of class j that is predicted as class i expressed as $c(i|j)$, given the sum of costs of class i is expressed as $\sum_j c(i|j)$, weight is added to class i and probability p_i is updated. If the total cost of class i is large, it means the number of samples are inflated (oversampling), sensitive model based on information of class i is built. The class file in CSV format is shown in Table 2.14, actual class (`real`), the corresponding predicted class (`predict`) and its cost (`cost`) is displayed in the same row. Given the combination of actual class and predicted class is not specified, and the cost file is not defined, when actual class and predicted class is the same, the cost is 0, otherwise, 1 is set.

Table 2.14: Example of specifying the cost in churn model. In first row, the churn class yes in the actual sample is predicted as no, the cost is set to 2. In the second row, when the churn class no is predicted as yes, the cost is set to 5. The column name can be pre-defined, but must be arranged in the order of actual class, predicted class, and cost.

real	predict	cost
yes	no	2
no	yes	5

2.3.2 Output Data

The output of various data files from this command are summarized in Table 2.15.

Table 2.15: List of output data from model building mode in mbonasi

File name	Content	Remarks
`model.pmml`	The decision tree model represented by PMML[a]	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 model such as the size and accuracy of the model.
`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

[a] Predictive Model Markup Language is an industry standard to describe data mining and mathematical models represented in XML-based file format. Note that this specific command uses an extended tag of the PMML standard.

model.pmml Based on the maximal tree of the decision tree created, `complexity penalty` attribute is shown for each node, the branch will be pruned when α is greater than the value of complexity penalty. Since the maximal tree and pruning information is recorded in PMML, α is specified for the execution of prediction model, it is possible that the corresponding value will be used to predict the decision tree model.

```
                :
<Node id="0" score="yes" recordCount="8" >
    <Extension extender="KGMOD" name="complexity penalty" value="0.500000"/>
                :
```

model_min.txt,model_1se.txt,model.txt Unlike PMML data, a summary of pruned model is saved in text format. The section `[alphabet-index]` shows the alphabet corresponding to the index. In the following example, alphabet `c` corresponds to index `1`, while `b,a` corresponds to index `2`. The branching rule of the pattern shown in the decision tree is indicated by index symbol.

```
[alphabet-index]
Field Name: BrandSequence
Index[1]={c}
Index[2]={b,a}
```

The section `[decision tree]` shows the decision tree in text format, information such as model size (the number of leaf nodes) and number of layers of the deepest leaf is shown below the tree.

```
[decision tree]
if($BrandSequence has 22)
```

```
  then $Churn=yes (hit/sup)=(4/4)
  else $Churn=no (hit/sup)=(4/4)

number of leaves: 2
deepest level: 1
```

The section `[Confusion Matrix by Training]` shows the prediction classification results of decision tree model using training data. In addition, classification table by cost, prediction accuracy by class, overall prediction accuracy is also shown.

The section `[Confusion Matrix by Estimation]` is shown when `ts=` or `cv=` is specified. The same format applies to `[Confusion Matrix by Training]` section, the result of test data will be shown differently.

```
[Confusion Matrix]
## TRAINING DATA ##
## By count
         Predicted As ...
        yes     no      Total
yes     4       0       4
no      0       4       4
Total   4       4       8

## By cost
         Predicted As ...
        yes     no      Total
yes     0       0       0
no      0       0       0
Total   0       0       0

## Detailed accuracy by class
class,recall,precision,FPrate,F-measure
yes,1,1,0,1
no,1,1,0,1

## Summary
accuracy=1
totalCost=0
```

Finally, the section `[Selected Alpha]` shows the value of the complexity parameters used for pruning.

```
[Selected Alpha]
alpha: 0
```

predict_min.txt,predict_1se.txt,predict.txt The prediction results is added to the training data used for building the decision tree model in CSV format. The prediction results, as described below, output the highest prediction probability in the `predict` column, and the prediction accuracy for each class (`yes` and `no` as shown below). When `ts=` is specified, it returns the prediction results of test data, when `cv=` is specified, it returns the prediction results by all test data using cross validation. In addition, when `alpha=` is specified, the prediction result of the training data is returned.

```
Brand Sequence,Churn,predict,yes,no
bcaba,yes,yes,1,0
bcabcaa,yes,yes,1,0
aaabac,yes,yes,1,0
caa,yes,yes,1,0
cca,no,no,0,1
cacbc,no,no,0,1
bcc,no,no,0,1
acca,no,no,0,1
```

model_info_min.csv,model_info_1se.csv,model_info.csv These files store the model information in CSV format. The column `nobs` refer to the number of records in training data, `alpha=` refers to the value of pruning complexity parameter, `accuracy,totalCost` refer to the percentage of correct answers in the test model and the total cost.

```
nobs,alpha,accuracy,totalCost
8,0,1,0
```

alpha_list.csv Display the error rate, standard error, error rate $\pm$ standard error corresponding to the α value of pruning complexity parameter for the pruned decision tree.

```
alpha    ,leafSize,errorRate,SE     ,up     ,lo
0        ,102     ,0.0082   ,0.0015 ,0.0098 ,0.0066
8.15e-05 , 98     ,0.0085   ,0.0016 ,0.0101 ,0.0069
9.41e-05 , 91     ,0.0091   ,0.0016 ,0.0108 ,0.0074
0.000124 , 82     ,0.0100   ,0.0017 ,0.0118 ,0.0083
    :       :         :         :       :       :
```

```
0.035669 ,  2     ,0.0878   ,0.0049 ,0.0927 ,0.0828
1.79e+308,  1     ,0.1399   ,0.0060 ,0.1460 ,0.1339
```

param.csv Various parameter values used when building the model is stored in CSV format.

2.3.3 Format 1: Model building mode

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

```
 i=          : Training data file name
 p=          : Column name of pattern (multiple fields can be specified)
             : Specify up to five parameters after the column name, each separated by a colon.
             : p=column_name:is:seq:ordered:head:tail:rs
             : is: Size of the index
             :   When the parameter is not specified, an index is not generated, instead, original alphabet of the pattern is used.
             : seq: Type of pattern
             :   true: Partial sequence pattern
             :   false: Partial string pattern (default)
             : ordered: Arrangement in alphabetical order 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, that is 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 explanatory 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 v
 cv=         : Specify partition of data by cross-validation method. When "cv=" is not specified, the default value is set as 10.
             : If either ts=,cv= is not specified, default value of alpha=0.01 will be applied.
             : Even when alpha=,ts=,cv=, is specified, pruning degree of maximum tree is recorded in PMML,
             : the value of alpha could change in prediction mode.
 --help      : Show help
```

2.3.4 Format 2: Prediction mode

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

`-predict`	: Operate in prediction mode. This parameter is required for prediction mode.
`i=`	: Input data [required]
	: The column names must be the same as the columns that was used for building the model.
`I=`	: Destination directory path for model building mode [required]
	: File required are as follows.
	: bonsai.pmml: pmml containing the decision model
`o=`	: Output file name containing prediction result
	: The "predict" column is added to the input data in output.
	: Columns must be the same as columns that was used in building the model.
`alpha=`	: Specify pruning complexity parameter.
	: Accepts real number greater than 0, as well as the following two symbols with special functions.
	: min: α value that corresponds to pruned model which minimizes the estimated misclassification rate.
	: 1se: Alpha value that corresponds to the pruned model with the same 1SE rule.
	: Designation of the two symbols is effective only when `ts=` or `cv=` is specified when building the model.
	: 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, that is 1 byte character is regarded as 1 alphabet)
`--help`	: Show help

2.3.5 Examples

Example 1 An example of building a model

In this example, since `ts=,cv=,alpha=` parameters are not specified, pruning is carried out when `alpha=0.01`, the result is saved to the file `model.txt`.

```
$ more input.csv
BrandSequence,Churn
bcaba,yes
bcabcaa,yes
aaabac,yes
caa,yes
cca,no
cacbc,no
bcc,no
acca,no

$ mbonsai p=BrandSequence:2 c=Churn i=input.csv O=result1

$ more result1/model.txt

[alphabet-index]
Field Name: BrandSequence
Index[1]={a}
Index[2]={b,c}

[decision tree]
if($BrandSequence has 11)
  then $Churn=yes (hit/sup)=(3/3)
  else if($BrandSequence has 2212)
    then $Churn=yes (hit/sup)=(1/1)
    else $Churn=no (hit/sup)=(4/4)

numberOfLeaves=3
deepestLevel&=& 2

[Confusion Matrix by Training]
## By count
         Predicted As \ldots
```

```
        yes        no        Total
yes       4         0         4
no        0         4         4
Total        4         4         8

## By cost
         Predicted As \ldots
        yes        no        Total
yes       0         0         0
no        0         0         0
Total        0         0         0

## Detailed accuracy by class
class,recall,precision,FPrate,F-measure
yes,1,1,0,1
no,1,1,0,1

## Summary
accuracy=1
totalCost=0

$ more result1/model.pmml
<?xml version="1.0" encoding="UTF-8"?>
<PMML version="4.0">
  <Header copyright="KGMOD">
    <Application name="mclassify" version="1.0"/>
    <Timestamp>2014/07/20 23:00:13</Timestamp>
  </Header>
  <DataDictionary numberOfFields="2">
    <DataField name="BrandSequence" optype="categorical" dataType="string">
      <Value value="b"/>
      <Value value="c"/>
      <Value value="a"/>
    </DataField>
    <DataField name="Churn" optype="categorical" dataType="string">
      <Value value="yes"/>
      <Value value="no"/>
    </DataField>
  </DataDictionary>
  <TreeModel functionName="classification" splitCharacteristic="binarySplit">
    <MiningSchema>
      <MiningField name="BrandSequence">
        <Extension extender="KGMOD" name="alphabetIndex">
          <alphabetIndex alphabet="b" index="2"/>
          <alphabetIndex alphabet="c" index="1"/>
          <alphabetIndex alphabet="a" index="2"/>
        </Extension>
      </MiningField>
      <MiningField name="Churn" usageType="predicted"/>
    </MiningSchema>
    <Node id="0" score="yes" recordCount="8">
      <Extension extender="KGMOD" name="pruning" value="0.000000"/>
      <True/>
      <ScoreDistribution value="yes" recordCount="4"/>
      <ScoreDistribution value="no" recordCount="4"/>
      <Node id="1" score="yes" recordCount="4">
        <Extension extender="KGMOD" name="pruning" value="0.000000"/>
        <Extension extender="KGMOD" name="patternPredicate" value="substring">
          <SimplePredicate field="BrandSequence" operator="contain">
            <index seqNo="1" value="2"/>
            <index seqNo="2" value="2"/>
          </SimplePredicate>
        </Extension>
        <ScoreDistribution value="yes" recordCount="4"/>
        <ScoreDistribution value="no" recordCount="0"/>
      </Node>
      <Node id="2" score="no" recordCount="4">
        <Extension extender="KGMOD" name="pruning" value="0.000000"/>
        <Extension extender="KGMOD" name="patternPredicate" value="substring">
          <SimplePredicate field="BrandSequence" operator="notcontain">
            <index seqNo="1" value="1"/>
            <index seqNo="2" value="1"/>
          </SimplePredicate>
```

```
        </Extension>
        <ScoreDistribution value="yes" recordCount="0"/>
        <ScoreDistribution value="no" recordCount="4"/>
      </Node>
    </Node>
  </TreeModel>
</PMML>
```

Example 2 Predict unknown data with the model from example 1

This example predict unknown data (using training data and unknown data in the following) based on the decision tree model constructed. The prediction result includes prediction accuracy of each class (`yes,no` column), class name with the highest accuracy (`predict` column) is returned.

```
$ more unknown.csv
BrandSequence
bcaba
bcabcaa
aaabac
caa
cca
cacbc
bcc
acca

$ mbonsai -predict i=unknown.csv I=result1 o=predict.csv

$ more predict.csv
BrandSequence,predict,yes,no
bcaba,yes,1,0
bcabcaa,yes,1,0
aaabac,yes,1,0
caa,yes,1,0
cca,no,0,1
cacbc,no,0,1
bcc,no,0,1
acca,no,0,1
```

2.4 mgpmetis.rb Graph Partitioning Command

This command allows users to easily execute the graph partitioning software METIS developed by University of Minnesota (http://glaros.dtc.umn.edu/gkhome/metis/metis/overview). Given an undirected graph, METIS partitions into finite elements (referred to as clusters below) containing the same number of nodes, while minimising the number of branch cut required. In the data structure for direct handling by METIS, node is represented as integer, however, it can be represented by any character in this command. In addition, graph data do not require a special format, edge and node data can be saved in CSV data. The CSV data will be converted internally by mgpmetis command into a format which can be handled by METIS. It is also possible to apply gpmetis command for data conversion into METIS format.

Input data is shown in Table 1, each row corresponds to an edge containing a node pair saved as CSV data. There are no isolated nodes in the input data, when the weight (please refer to details below) of node is not specified, only branch data is created. The corresponding graph is shown in Figure 1. For example, users can execute the following command in order to partition the graph into two parts.

```
$ mgpmetis.rb kway=2 ef=node1,node2 ptype=rb ei=input.csv o=output.csv
```

The result is shown in 2, where the output contains the node name and the corresponding cluster number. Figure 2 shows that the graph is partitioned into two parts by cutting the minimum number of edges.

Table 2.16: Input data (input.csv)

node1	node2
a	b
a	c
a	e
b	c
b	d
c	d
c	e
d	f
d	g
e	f
f	g

Table 2.17: Partition output data (output.csv)

node	cluster
a	1
b	1
c	1
d	0
e	1
f	0
g	0

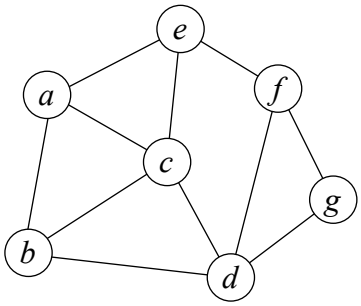

Figure 2.1: Graph of partition target

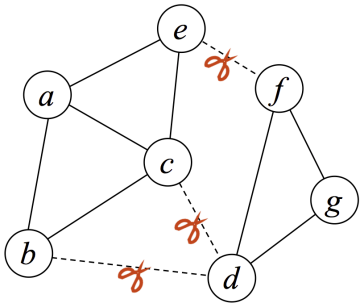

Figure 2.2: Results of two partition after minimum cut. Since there are 7 nodes, partition closest to equal division is 4:3 partition, thus, this figure shows minimally, 3 edges are removed to create the partitions.

2.4.1 Format

```
mgpmetis.rb kway= [ptype=rb|kway] ei= [ef=] [ew=] [ni=] [nf=] [nw=] [o=]
            [balance=] [ncuts=] [dat=] [map=] [-noexe] [--help]
```

`ei=`	: File name of branch (node pairs) [required]
`ef=`	: Field name of node pair (two columns) in edge file [default: "node1,node2"]
`ew=`	:Field name of weight in edge file [optional: weight is 1 when this is not specified]
	: Weight must be specified as an integer.
`ni=`	: File name of node [optional]
`nf=`	: Field name of node (1 column) in node file [default: "node"]
`nw=`	: Field name of weight in node file (multiple fields can be specified) [optional: weight is 1 when this is not specified]
	: Weight must be specified as an integer.
`o=`	: Output file name [optional: default uses standard output]
`kway=`	: Number of divisions [required]
`ptype=`	: Partition algorithm [default: kway]
`balance=`	: Balanced partition parameter [default: when ptype=rb 1.001, when ptype=kway is 1.03]
	: Specify the β value as shown in equation 2.
`ncuts=`	: The number of trials for initial value in partition phase [option: default=1]
`dat=`	: File name of the data used for gpmetis command.
`map=`	: File name of the mapping data of node number corresponding to node name specified in i= parameter used for gpmetis c
`-noexe`	: Do not execute gpmetis. This is used when only output data at dat=,map= parameters.
`--help`	: Show help

2.4.2 Algorithm

gpmetis is a graph partition algorithm that can be divided into three processes namely 1) coarsening, 2) partitioning 3) uncoarsening. During coarsening, the process of integrating multiple nodes connected by edges, and reduces to small graphs consisting of hundreds of nodes. In the coarsened graph, it is possible to obtain balanced partitions (with consideration of number of integrated nodes), while minimising the number of cut on edges. Finally, the uncoarsened process revert all partitioned nodes back to integrated nodes 3.

The features of gpmetis algorithm are as follows.

It becomes difficult to solve a NP complete graph partitioning problem when the graph size increases. Therefore, gpmetis avoid the bottleneck by applying coarsening during preprocessing to reduce the size of the graph. However, since coarsening reduces the flexibility of different ways to partition the graph, the general partition accuracy (objective function that describes the minimisation of edge cut is described later) is decreased. Note that the algorithm in gpmetis is an approximation algorithm, thus optimal solution is not guaranteed.

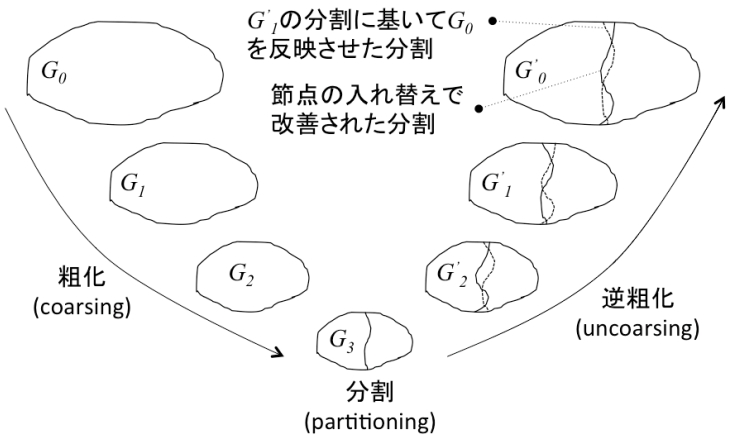

Figure 2.3: Conceptual diagram of multi-level graph partitioning (Refer to literature [2] of Figure 1). In coarsening process, when the number of nodes reached several hundreds, original graph is reduced by merging the nodes to build condensed, smaller graphs. The reduced graphs have uniform number of nodes, with minimal number of edge cut for partition. In the uncoarsening process, the merged nodes are revert back to original. At that time, the nodes between partitions are replaced to improve the minimum cut. In the figure, G_2 and G'_2 consists the same node set, yet the cut edge set is different for partition purposes.

Problem Setting

In undirected graph $G = (V, E)$, node set V has k number of partitions into $V_1, V_2, \ldots, V_k$. In this case, the edges attached to partition is minimized (minimize edge cut), the nodes belong to each partition are divided equally. Generally, vertex u, v extends edge to $(u, v) \in E$ with weight $w(u, v) \geq 0$, weight of node v is represented as w_v, the function to minimize edge cut is represented in equation 1. In addition, by introducing constraints to unify parameter $\beta \geq 1.0$ allows creation of uniform partitions (equation 2). Balancing partitions corresponds to the ratio of average number of nodes (weight) in partition to the biggest number of nodes (weight), thus a larger β value allows for more unbalanced partition. β can be specified by `balance=` parameter in the command.

$$\underset{V_1,V_2,\ldots,V_k}{\sum_{(u,v)\in E \wedge u\in V_i \wedge v\in V_j \wedge i\neq j}} w(u,v) \tag{2.12}$$

$$\text{subject to} \quad \frac{\max_i \sum_{v\in V_i} w_v}{\frac{1}{k}\sum_{i=1}^{k}\sum_{v\in V_i} w_v} \leq \beta \tag{2.13}$$

Recursive bisection and k-way split

gpmetis adopt an algorithm suitable for NP-complete graph partition problem, instead of finding the optimal solution, it is possible to apply multi-level partitioning, thereby making it possible to calculate increase in graph size at real time. Multi-level partitioning generally follows several phases of the constructed algorithm. By contracting the original graph (referred to as coarsening), graph is divided when it is at a sufficiently small size. Afterwards, when partition accuracy is improved (minimize edge cut), uncoarsening process will return the graph to original size.

gpmetis command comprised of two partition algorithm namely multi-level recursive bisection and multi-level k-way partition. When a node set V is divided into k partitions, recursive bisection partition the original graph into two subgraphs through "coarsening, partition, uncoarsening" phases, and recursively divides each of the partition [1].

Recursively bisection is shown in Algorithm1. On the other hand, in k-way split method, the graph is directly partitioned into k parts after coarsening, and the process ends with uncoarsening. k-way partition is shown in Algorithm2. Either method provides excellent way of partition, in terms of execution time, k-way partition method is superior since it does not perform recursive partition[2].

Algorithm 1 k split graph partition algorithm: **Recursive bisection**

```
1: function BISECT(G, P)
2:     G : Partitiontargetgraph, P : Partitionset
3:     if size of G is 1/k then
4:         P = P ∪ {G}
5:     else
6:         C=Coarsen(G)                          ▷ Coarsening
7:         C_1, C_2 = 2wayPartition(C)           ▷ Coarsen graph bisection
8:         G_1, G_2 = Uncoarsen(C_1, C_2)        ▷ Uncoarsen
9:         P=BISECT(G_1, P)                      ▷ G_1 in recursion
10:        P=BISECT(G_2, P)                      ▷ G_2 in recursion
11:    end if
12:    return P
13: end function
```

[1] Recursive partition do not multiply k by power of 2, weight of partitions are deliberately unbalanced so that it is possible to divide uniformly as a whole. For example, with $|V| = 9, k = 3$, it is initially partitioned into two parts at ratio of 3:6, afterwards, the 6 graphs are recursively partitioned into 3:3. The details are not shown in Algorithm1.

[2] Given $k = 256$, it is reported that it is 3 to 4 times faster [3].

Algorithm 2 k split graph partition algorithm: k**-way partitioning method**

```
1: function KWAY(G)
2:     G : Partitiontargetgraph
3:     C=Coarsen(G)                                   ▷ Coarsening
4:     C_1, C_2, ..., C_k = KwayPartition(C)          ▷ Partition coarsened graph into k splits
5:     G_1, G_2, ..., G_k = UncoarsenKway({C_1, C_2, ..., C_k})   ▷ k split during uncoarsening
6:     return {G_1, G_2, ..., G_k}
7: end function
```

In the following, Algorithm1,2 shows each sample function, Coarsen(), 2WayPartition(), KwayPartition(), Uncoarsen(), UncoarsenKway() with summary below. Please refer to [2] for details.

Coarsen function)

The purpose of coarsening is to reduce the size of the graph in order to perform partition efficiently. Coarsening algorithm is common to both recursive bisection and k-way partition. The coarsening phase finds out maximal matching M from the original graph $G_0 = (V_0, E_0)$, each element (edge) is merged with new node if matched, to create new graph $G_1 = (V_1, E_1)$. Where graph $G = (V, E)$ matches M, and subset of edge set E ($M \subseteq E$), the vertices of any 2 edges $e_1, e_2 \in M$ does not share the same edge set with each other. When matching M, if no more branches can be added, it is known as maximal matching. The above operation is applied recursively until the node size is at several hundred, a series of coarsened graphs $G_1, G_2, \ldots, G_m$ are created. The details on finding maximal matching algorithm is not described in gpmetis. However, note that this method is based on a heuristic method using random numbers.

2WayPartition function, KwayPartition function)

Below details 2 way partition for the coarsened graph $G = (V, E)$. Division of k number of partitions can be achieved by recursively applying bisection method. The bisection algorithm employed in gpmetis is very simple and emphases on efficiency. First, select the initial node randomly from the node set V, and connect with another added node, the process ends when only half a node (total weight) is added. When the node is added to the node set, it is represented as P, there are two methods for P to add node $v \in V \setminus P$, they are GGP(Graph Grawing Algorithm) andGGGP(Greedy Graph Growing Algorithm). GPP selects node v randomly. Yet in GGGP, P is connected to $V \setminus P$ at which node v has the greatest difference. Equation $w(v, u)$ shows the edge extension weight between node v and u. GGGP use greedy algorithm to reduce the number of cut when selecting node v. GGGP is applied in gpmetis by default, GGP is used when the weight constraints of node are specified more than once.

$$g_v = \sum_{(v,u) \in E \wedge u \in (P)} w(v, u) - \sum_{(v,u) \in E \wedge u \in (V \setminus P)} w(v, u) \tag{2.14}$$

Uncoarsen function, UncoarsenKway function)

A series of coarsened graphs $G_1, G_2, \ldots, G_m$ are obtained through the coarsen process. The reverse direction $(G_m, G_{m-1}, \ldots)$ returns the combined nodes and edges to the original state. Partition process G_m divides into k partitions, if the partition is optimal, when G_m is returned to G_{m-1}, it may become less than optimal partitioning. Some of the nodes may be moved between partitions to improve partition accuracy (minimize edge cut). The above process is repeated until G_0 is obtained, which is the final solution of partition.

Other parameters

The coarsening, partition, uncoarsening algorithm use random number, different result may be obtained by a series of random number. Therefore, when `ncuts=` parameter is specified, the process of coarsening to uncoarsening is executed multiple times, thus it is possible to choose the best split. Recursive bisection method repeats the number of times specified in line 6,7,8 of Algorithm1, and line 3, 4, 5 of Algorithm2.

In gpmetis, there are a number of parameters that can be used to control partitioning other than those from the mentioned algorithm. This command can handle some of the parameters. If you want to set parameters that cannot be handled by this command, users can process data directly with METIS and create its processing data with this command. Specify the output data used by METIS at `dat=` (node number is shown as integer in data).

2.4.3 Examples

Example 1. Example of the above illustration

Partition into two parts using recursive bisection method(`ptype=rb`). If the field name is not specified at `ef=` in this command, field names of edge data is designated as `node1,node2` by default.

```
$ more input.csv
node1,node2
a,b
a,c
a,e
b,c
b,d
c,d
c,e
d,f
d,g
e,f
f,g
$ mgpmetis.rb ei=input.csv o=output.csv kway=2 ptype=rb
$ more output.csv
node,cluster
a,1
b,1
c,1
d,0
e,1
f,0
g,0
```

Example 2 . Specify node and edge weight

Use of weight in column `v` in `edge.csv` file, and column `v` in `node.csv` file.

```
$ more edge.csv
n1,n2,v
a,b,1
a,c,1
a,e,1
b,c,1
b,e,1
b,g,2
c,d,3
c,g,1
d,e,1
e,f,1
$ more node.csv
n,v
a,1
b,1
c,3
```

```
d,1
e,1
f,1
g,3
$ mgpmetis.rb ni=node.csv ei=edge.csv o=rsl.csv ptype=rb kway=2 ef=n1,n2 nf=n nw=v ew=v
$ more rsl.csv
n,cluster
a,1
b,1
c,0
d,0
e,1
f,1
g,1
```

Bibliography

[1] C.M. ビショップ著, 元田浩, 栗田多喜夫, 樋口知之, 松本裕治, 村田昇（編）, Patten Recognition and Machine Learning（II）：Statistical Prediction by Bayes Theory, Chapter 13, pp.323–370, 2008.

[2] Karypis, G. and Kumar, V., "A fast and high quality multilevel scheme for partitioning irregular graphs", SIAM Journal on Scientific Computing 20 (1), pp.359–392, 1999.

[3] Karypis, G. and Kumar, V., "Multilevel k-way Partitioning Scheme for Irregular Graphs", Journal of Parallel and Distributed Compupting 48, pp.96–129, 1998.

[4] S. Shimozono, A. Shinohara, T. Shinohara, S. Miyano, S. Kuhara and S. Arikawa, Knowledge Acquisition from Amino Acid Sequences by Machine Learning System BONSAI, *Trans. Information Processing Society of Japan*, Vol. 35, pp. 2009-2018, 1994.

[5] J. R. Quinlan, *C4.5: Programs for Machine Learning*, San Meteo: Morgan Kaufmann, 1993.

[6] L. Breiman, J. Friedman, R. Olshen, C. Stone *Classification and regression trees*, Wadsworth: Belmont, CA, 1984.