Mining Package Documentation

対応 NYSOL パッケージ: Ver. 1.2,2.0

revise history:
October 6, 2014 : mbonsai,mgpmetis.rb コマンドの追加
January 17, 2014 : first release

2014 年 10 月 6 日

目次

第 1 章

はじめに

1.1 概要

データマイニングは、膨大なデータから意味のあるパターンやルールを発見するために、統計学、パターン認識、人工知能等のデータ解析の技法を用いた技術であり、知識発見プロセスの 1 つに位置づけられる。データマイニングの技術は、相関ルールの抽出に代表される「頻出パターンの列挙」、与えられたデータのカテゴリを予測する「クラス分類」、実数値を予測する「回帰分析」、そして、類似するデータを分類する「クラスタリング」などが代表的な解析手法である。本データマイニングコマンドは、そのような解析手法を実現するためのコマンドであり、Ruby 上で大規模な CSV データを扱うための Ruby 拡張ライブラリを用いており、MCMD と同様に CSV 形式のデータを入力として与える。

1.2 インストール

M コマンドの動作確認は、以下の OS で確認している。

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

いずれも、すぐにインストール可能なパッケージが用意されている。上記の OS におけるバージョンに違いがあっても、パッケージをインストールが可能であろう。また他の OS であっても、ソースからのコンパイルを行うことでインストールが可能である。

1.2.1 Mac OS X

http://sourceforge.jp/projects/nysol/releases/より最新の gem パッケージをダウンロードし、以下の手順にしたがってインストールする。ただし、ファイル名は「`mining-1.1-x86_64-darwin.gem`」の形式に従っており、`"1.1"`は最新のバージョン番号に読み替える。

```
$ sudo gem install mining-1.1-x86_64-darwin.gem
```

1.2.2 Ubuntu Linux

http://sourceforge.jp/projects/nysol/releases/より最新の gem パッケージをダウンロードし、以下の手順にしたがってインストールする。ただし、ファイル名は以下に例示される形式に従っている (`"1.1"`は最新のバージョン番号に読み替える)。

- 32bit 環境: `mining-1.1-x86-linux.gem`
- 64bit 環境: `mining-1.1-x86_64-linux.gem`

```
$ sudo gem install mining-1.1_darwin.gem
```

1.2.3 ソースからのインストール

プログラムソースからコンパイルする際は、以下の手順に従う。

以下の手順に従い、mining の git サーバより最新の mining のソースをダウンロード＆インストールする。

```
$ git clone http://scm.sourceforge.jp/gitroot/nysol/mining.git
$ cd mining
$ make
$ sudo make install
```

第 2 章

マイニングコマンド

2.1　burst.rb burst 検知コマンド

与えられた系列データから burst 状態を検知する．検知アルゴリズムには HMM(Hidden Markov Model) を用いている．定常状態とバースト状態の 2 状態における確率分布を仮定し，与えられたデータから尤度が最大となるような状態遷移経路を出力する．確率分布としては，指数分布，ポアソン分布，正規分布，二項分布を指定できる．詳細は次節に示している．

入力データとしては，表 2.1 に示されるような数値系列データ (val 項目) である．その他の項目 (id 項目のような項目) は burst 検知には一切利用されない．ただし，出力結果は入力データに burst 項目を追加して出力される (表 2.2) ので必要な項目があれば入力データに含めておく．

表 2.1　入力

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

表 2.2　ポアソン分布 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

表 2.3　指数分布 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

表 2.4　正規分布 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

この系列データを単位時間あたりのイベント発生数 (例えば，1 時間あたりのメールの到着数) と考えると，ポアソン分布を用いた burst 検知を実施するればよい (表 2.2)．また，イベントの発生間隔 (例えば，メールの到着間隔秒数) と考えると，指数分布を用いた burst 検知を実施すればよい (表 2.3)．さらに，このデータを何らかの誤差系列 (例えば，株価推移) と考えれば正規分布を用いた burst 検知を実施すればよい (表 2.4)．このように，同じ入力データに対して，どのような分布を仮定するかによって，異なる burst 検知を実施することが可能で，分布の選択は応用課題によって決まってくる．

2.1.1　書式

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

注 1

定常状態における分布のパラメータ (母数) の与え方は以下の 3 通りある．

1. `param=`で指定した値とする．

```
i=       : 入力ファイル名【必須】
o=       : 出力ファイル名【オプション:default は標準出力】
d=       : デバッグ情報を出力するファイル【オプション】
dist=    : 仮定する分布名称 (exp:指数関数,poisson:ポアソン分布,gauss:正規分布,binom:二項分布)【必須】
f=       : burst 検知対象となる数値項目名 (i=上の項目名)【必須】
param=   : 定常状態における分布のパラメータ．注 1 参照【オプション】
pf=      : 定常状態における分布のパラメータ項目名 (i=上の項目名) 注 1 参照【オプション】
s=       : burst スケール (この値を大きくすると極端な burst のみを検知できる．詳細は注 1 参照)【オプション:default=2.0】
p=       : 同一状態遷移確率 (この値を高くすると異なる状態に遷移しにくくなる．詳細は注 2 参照)【オプション:default=0.6】
n=       : dist=binom の場合の試行回数【n= or nf=いずれかを指定】
nf=      : dist=binom の場合の試行回数の項目名
v=       : dist=gauss の場合の分散値 (指定がなければ f=項目のデータから推定)
nv=      : dist=gauss の場合の分散の項目名
--help   : ヘルプの表示
```

2. `pf=`で指定した項目の値を用いる．時刻に依存してパラメータが異なることが仮定できる場合のため．
3. `para=`,`pf=`の指定がなければ，`f=`で指定した値から自動的に計算される．

上記 3. のデータから計算する方法は，分布ごとに異なり以下の通りである．ただし，S は `s=`で指定した値，n はデータ件数，x_i は `f=`で指定した項目の i 行目の値とする．

分布	確率 (密度) 関数	パラメータ	定常状態パラメータ	burst 状態パラメータ
exp	$f(x) = \lambda e^{-\lambda x}$	λ:平均イベント発生回数	$\lambda_0 = N/\sum_i x_i$	$\lambda_1 = S\lambda_0$
poisson	$f(x) = \frac{\lambda^x e^{-\lambda}}{x!}$	λ:平均イベント発生回数	$\lambda_0 = \frac{1}{N}\sum_i x_i$	$\lambda_1 = S\lambda_0$
gauss	$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(x-\mu)^2/2\sigma^2}$	μ:平均	$\mu_0 = \frac{1}{N}\sum_i x_i$	$\mu_\pm = \mu_0 \pm S\sqrt{\sigma^2}$ *
binom	$f(x) = {}_nC_x p^x (1-p)^{n-x}$	p:成功確率	$p_0 = \frac{1}{Nn}\sum_i x_i$ **	$p_1 = S/(\frac{1-p_0}{p_0} + S)$

**$\sigma^2 = \frac{\sum(x_i - m)^2}{N-1}$

***n は `n=`によって指定する．

注 2

`p=`で指定した確率を p とすると，状態遷移確率は以下の通り設定される．

表 2.5 exp, poisson, binom での状態遷移確率

	定常	burst
定常	p	$1-p$
burst	$1-p$	p

表 2.6 gauss での状態遷移確率

	burst-	定常	burst+
burst-	p	$\frac{2}{3}(1-p)$	$\frac{1}{3}(1-p)$
定常	$\frac{1}{2}(1-p)$	p	$\frac{1}{2}(1-p)$
burst+	$\frac{1}{3}(1-p)$	$\frac{2}{3}(1-p)$	p

2.1.2 解説

定式化

HMM(Hidden Malkov Model は，直接は観測できない隠れ状態がマルコフ過程に従って推移していることを仮定して構築される確率モデルである．HMM は確率的状態遷移モデルとデータ生成モデルから構成され，観測される系列データは，隠れ状態におけるデータ生成モデルに従うと考える．時刻 t において観測されたデータ x_t は，隠れ状態 $z_t \in \{1, 2, \cdots, K\}$ に定義された確率分布 $p(x_t|z_t; \boldsymbol{\phi})$ に従って生成されるようにモデル化される．ここで，$\boldsymbol{\phi}$ は生成モデルのパラメータベクトルで，t に依存せず一定であると仮定する．また，隠れ状態 z_t は直前の状態 z_{t-1} にのみ依存して遷移し，その確率分布は $p(z_t|z_{t-1}; \mathbf{A})$ で表される．ここで $\mathbf{A} = \{a_{i,j}|i, j = 1, 2, \cdots, K\}$ は，状態 i から状態 j への遷移確率表で，t に依存せず一定であると仮定する．ただし，$\sum_j a_{i,j} = 1.0$ で，また初期状態 z_1 は確率ベクトル $\boldsymbol{\pi}$ に従うものとする．

以上より，観測データ系列 $X = x_1, x_2, \cdots, x_N$，および状態系列 $Z = z_1, z_2, \cdots, z_N$ の同時確率は式 (2.1) で与えられる [1].

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

burst 検知においては，$K = 2$，すなわち定常状態と burst 状態の 2 状態を仮定し，それぞれの状態において，異なるパラメータを持つ同一の確率分布から観測可能なデータ系列が得られると考える．そして，burst 検知問題とは，パラメータ $\boldsymbol{\pi}, \mathbf{A}, \boldsymbol{\phi}$ が与えられたなかで，データ系列 $\mathbf{X}$ を観測した時に，式 (2.1) で示された同時確率を最大化するような $\mathbf{Z}^*$ を見つける問題である (式 (2.2)).

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

メールの burst 検知例

以上の定式化に従い，以下では，メールの到着数 (表 2.1) に関する burst 検知を例にとり解説する．ここでの目的は，表 2.1 に示された数値列 $\mathbf{X} = 1, 1, 4, \cdots$ から，最も尤もらしい隠れ状態列 $\mathbf{Z} = z_1, z_2, \cdots, z_N (z_i \in \{0, 1 | 0$ は定常状態, 1 はburst状態 $\})$ を求めることである．

メールの時間ごとの到着数を確率変数と考えた場合，ポアソン分布を仮定するのが妥当であろう．ポアソン分布は平均到着数 λ をパラメータにとり，その確率関数は式 (2.3) で示される．

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

ここで，定常状態におけるパラメータ λ_0 は，コマンドパラメータ `param=`や `pf=`によっても与えるが，それらの指定がなければ `i=`で指定したデータから平均到着数により与えられる．表 2.1 の `val` 項目の平均値を計算すると $\lambda_0 = 4.29$ となる．そして，burst 状態におけるパラメータ λ_1 は，特に指定がなければ平常状態の 2 倍 (8.58) に設定される．この値は `s=`を設定することで変更することができる．以上より，状態の確率分布のパラメータベクトルは $\boldsymbol{\phi} = (4.29, 8.58)$ となる．表 2.7 に，データ系列 $\mathbf{X}$ の数値が各状態から出力される確率が示されている．数値”1”や”4”は，定常状態から出力される確率の方が高く，数値”10”は burst 状態から出力される確率の方が高いことがわかる．

次に状態遷移確率 $p(z_i | z_{i-1}; \mathbf{A})$ について考える．2 状態において遷移の組み合せは 4 通りあるが，本コマンドでは同一状態間の遷移確率に対して同一の確率値のみ与えることができ，特に指定しなければ 0.6 となる．すなわち $a_{0,0}, a_{1,1} = 0.6$ である．そして，異なる状態遷移確率は，$a_{0,1}, a_{1,0} = 0.4$ と計算される．状態遷移行列 $\mathbf{A}$ を式 (2.4) に示す.

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

そして最後に初期状態の確率ベクトルは $\pi = (1.0, 0.0)$ とし，初期状態は定常状態であることを前提とする．

以上で，式 (2.1) のパラメータ $\mathbf{A}, \boldsymbol{\phi}, \boldsymbol{\pi}$ が揃ったことになる．そこで次に式 (2.1) を最大化するような状態系列 $\mathbf{Z}^*$ を求める．データ系列 $\mathbf{X}$ のサイズは 21 で，考えうる状態系列の組み合せは，2^{21} = 約 200 万 通りあり，より長い系列になると，総当りで最適解を求める方法ではとても救解できなくなる．この問題は，Viterbi アルゴリズムと呼ばれる動的計画法により効率的に解くことができることが知られている．詳細は専門書に譲るとして，この方法で得られた状態系列が表 2.2 に示される通り得られる．

2.1.3　利用例

例 1 上記「解説」の例

```
-----------------------------------------------
# inp1.csv
id,val
```

表 2.7 ポアソン分布における各状態の確率

id	val	定常 ($\lambda = 4.29$)	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

```
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 Bayes 分類器

ベイズの定理による確率モデルを用いた教師あり学習の分類器を構築する．アイテムの出現を条件としたときの各クラスに属する確率が計算され，テストデータに対しては予測されたクラスが返される．本コマンドの特徴は以下の通りである。

- アイテムの頻度情報を扱えるように Multinomial Naïive Bayes を用いている。
- 未知データで初めてアイテムが出現した場合、そのアイテムを含む確率がゼロになる問題をラプラススムージングによって回避している。
- Complement Naïve Bayes が利用可能である。
- 交差検証を用いてモデルの判別精度が評価できる。
- モデルの構築モードと未知データを与えての予測モードがある。

2.2.1 解説

Naïve Bayes モデルは，独立性の仮定とベイズの定理を利用した確率モデルである．特徴値の出現を $w_i = 0, 1$ とする特徴ベクトル $\mathbf{w} = (w_1, w_2, \cdots, w_n)^\top$ を考える．$\mathbf{w}$ の出現を条件とした各クラス c の確率 $p(c|\mathbf{w})$ は，ベイズの定理により式 (2.5) で表される．

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

分母の $p(\mathbf{w})$ は，c によらず一定である．そこで，分子について見ると，$p(c)$ はクラス c の事前確率で，この確率がクラスの中で特徴ベクトル $\mathbf{w}$ の出現確率である尤度 $p(\mathbf{w}|c)$ によって，事後確率 $p(c|\mathbf{w})$ へと更新される．これがベイズの定理が意味することである．

$\mathbf{w}$ の次元が高くなると，単語の同時確率 $p(\mathbf{w}|c)$ の推定が困難となるため，Naïve Bayes 法は，式 (2.6) に示されるように，全ての単語の出現は独立であるというナイーブな仮定をおくことで，$p(\mathbf{w}|c)$ を計算する．

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

以上のことを踏まえると $p(c|\mathbf{w})$ は式 (2.7) で表され，そして，分類器を構築するために，最も尤もらしい仮説を採用する最大事後確率（MAP）決定規則を用いて，推定クラス $\hat{c}$ は式 (2.8) によって求めることができる．

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

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

ナイーブベイズ分類器が実際に利用される場合には，特徴ベクトルは，例えば，単語やアイテムなどであり，それらには，単語の出現回数や，アイテムの購買回数など頻度情報を伴う場合が多い．そこで特徴ベクトル $\mathbf{f}$ の要素 f_i を特徴値 i の出現頻度とすると，各推定クラス $\hat{c}$ は，式 (2.9) に示されるように，尤度に頻度 f_i を乗じて計算される．これが Multinomial Naïve Bayes モデルである．

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

ゼロ頻度問題

あるクラスとある特徴値の組合せが訓練例に出現しない場合に確率推定はゼロとなり，乗算に用いると積がゼロになってしまう．このゼロ頻度問題を回避するために，スムージングを行うことで確率値の推定をわずかに修正して，どの組合せの確率値もゼロにならないように調整する．ここでは特徴値の出現回数に 1 を加えるラプラススムージングを採用している．

Complement Naïve Bayes

ナイーブベイズ分類器で，あるクラスに属さない補集合を用いて学習させる拡張を Complement Naïve Bayes という．クラスに属さない特徴ベクトルを用いて学習するため，クラスを予測する際には，属さない確率が最も低いクラスを割り当てることになる．2 値の分類問題では同じ結果になるため意味がなく，多値の分類問題で各クラスのバラつきが多い場合には効果がある．-c オプションを指定すると Complement Naïve Bayes として実行される．

2.2.2　書式 1:モデル構築モード

```
mnb.rb [tid=] [item=] [w=] [class=] i= O= [seed=] [-complement] [ts=|cv=] [T=] [-mcmdenv] [--help]
```

```
tid=          : 1 つのサンプルを表す項目名【デフォルト:"tid"】
item=         : 1 つの変数を表す項目名【デフォルト:"item"】
w=            : 変数の重み項目名 【オプション】
              : 指定しなければ、全行 1 とする。
class=        : 目的変数の項目名 (i=上の項目名)【デフォルト:“class“】
i=            : 入力データのファイル名【必須】
O=            : 出力ディレクト名 【必須】
seed=         : 乱数の種 (0 以上の整数, 交差検証に影響)【オプション:default=-1(時間依存)】
-complement   : complement Naïve Bayes で実行【オプション】
ts=           : テストサンプル法によるテストデータの割合をパーセントで指定する。
              : 値を省略して " ts= " と指定すると 33.3 が用いられる。
cv=           : 交差検証法によるデータの分割数を指定する。値を省略して " cv= " と指定すると 10 が用いられる。
              : ts=,cv=のいずれも指定されていない場合は、i=で指定されたファイルを用いてモデルを作成する。
T=            : 作業ディレクトリ【デフォルト:“/tmp”】
-mcmdenv      : 内部の MCMD のコマンドメッセージを表示
--help        : ヘルプの表示
```

2.2.3　書式 2:予測モード

```
mnb.rb -predict i= I= o= [-complement] [T=] [-mcmdenv] [--help]
```

```
-predict      : 予測モード
i=            : 入力データのファイル名【必須】
I=            : モデル構築モードでの出力先ディレクトリパス 【必須】
o=            : 予測結果出力ファイル名 [必須]
-complement   : complement Naïve Bayes で実行【オプション】
T=            : 作業ディレクトリ【デフォルト:“/tmp”】
-mcmdenv      : 内部の MCMD のコマンドメッセージを表示
--help        : ヘルプの表示
              tid=,item=,w= については、モデル構築モードと同じ項目名を持つ入力ファイルが必要である。
```

2.2.4　本コマンドがモデル構築時に出力するデータ一覧

本コマンドがモデル構築時に出力するデータの一覧を表 2.8 に示す。

ファイル名	内容
category.csv	モデル構築時の word 種類数 (予測モードで利用)
clsProb.csv	モデル構築時の事前確率 (予測モードで利用)
clsWord.csv	モデル構築時の word 出現頻度 (予測モードで利用)
acclist.csv	検証毎の予測精度
acc.csv	予測精度
rsl_model.csv	入力データに対する予測結果
rsl_acc_train.csv	入力データに対する正解率
param.csv	実行パラメータ一覧

表 2.8 本コマンドがモデル構築時に出力するデータ一覧

■category.csv モデル構築時に i=で指定されたデータに含まれている item=で指定した項目の種類数を示している。以下の例では訓練データに 3 種類の単語が含まれていることを示している。

```
wCategory
3
```

■clsProb.csv モデル構築時に i=で指定されたデータに含まれるクラス別の件数とその出現確率を示している。以下の例では、class 項目は F と M で、memberNum は F と M それぞれ 10 件のデータがあることを示しており、prob 項目は、出現確率 (10/20) の自然対数を示している。

```
class,memberNum,totalId,prob
F,10,20,-0.6931471806
M,10,20,-0.6931471806
```

■clsWord.csv モデル構築時に i=で指定されたデータに含まれる単語の出現頻度をカウントしたもので、クラス毎に item=で指定した項目の w=で指定した出現頻度の合計を示している。以下の例では、wCnt 項目は、3 種類の単語 w1,w2,w3 の出現頻度をクラスごとに示している。

```
word,class,wCnt
w1,F,13
w1,M,33
w2,F,13
w2,M,28
w3,F,5
w3,M,15
```

■acclist.csv モデル構築時の精度を示したもので、cv=を指定した場合は、交差検証の回数分の予測精度が一覧として出力される。ts=を指定した場合は 1 回の予測精度が出力される。以下の例は、5 回の交差検証を実施した場合の結果であり、test 項目には実施回数、cnt 項目は正解数、totalCnt は検証したデータ数、そして accRate はその予測精度を示している。

```
test,ans,cnt,totalCnt,accRate
1,Match,4,4,1
2,Match,2,4,0.5
3,Match,3,4,0.75
4,Match,4,4,1
5,Match,4,4,1
```

■acc.csv　acclist.csv に含まれる予測精度の平均値を出力したものである。以下の例は、acclist.csv の例の値を平均したものである。

```
accRate
0.85
```

■rsl_model.csv　モデル構築時に i=で指定されたデータの tid=で指定したサンプル毎に、各クラスに属する確率と predictCls によって最終的に予測されたクラス項目が出力される。class は cls=で指定されたオリジナルのクラスを示している。-complement が指定された場合は、属さない確率が最も低いクラスを予測クラスとして出力している。

```
tid,F,M,class,predictCls
1,0.5149523047,0.4850476955,M,F
10,0.4929065867,0.5070934133,F,M
11,0.5019607343,0.4980392657,F,F
12,0.5089038694,0.4910961304,M,F
13,0.4918393826,0.5081606174,M,M
14,0.4966021486,0.5033978514,M,M
15,0.4929065867,0.5070934133,F,M
16,0.5019607343,0.4980392657,F,F
17,0.5019607343,0.4980392657,M,F
18,0.5149523047,0.4850476955,F,F
19,0.4932580133,0.5067419868,F,M
2,0.4922318437,0.5077681563,M,M
20,0.5008042081,0.4991957921,F,F
3,0.5070546328,0.4929453672,F,F
4,0.4983186364,0.5016813634,M,M
5,0.4929065867,0.5070934133,F,M
6,0.5070546328,0.4929453672,F,F
7,0.5115578733,0.4884421266,M,F
8,0.4918393826,0.5081606174,M,M
9,0.4983186364,0.5016813634,M,M
```

■rsl_acc_train.csv　モデル構築時に i=で指定されたデータを対象にした予測精度を出力している。以下の例は、rsl_model.csv の例で出力した class と predictCls 項目を利用して計算したものである。ans 項目はオリジナルのクラスと予測クラスの一致、不一致を示しており accRate 項目は正解率と不正解率を示している。

```
ans,cnt,totalCnt,accRate
Match,12,20,0.6
Unmatch,8,20,0.4
```

2.2.5　投稿者の性別判定に関する利用例

mnb.rb の利用例として、以下の train.csv データを用いて投稿内容に含まれる単語から性別の判定を行うケースを示す。ここでの目的は，train.csv に含まれるそれぞれの word を特徴ベクトルとして，class で指定された性別を判定するためのナイーブベイズモデルを構築し、test.csv の検証データ上の tid が，M と F のどちらになるかそのクラスを予測する。

train.csv を利用してまずは mnb.rb をモデル構築モードで実行すると、rsl_model.csv が出力される。それ以外の出力ファイルについては表 2.8 で示した通りである。次に未知データに対する予測を実施するために、test.csv に対して，mnb.rb を-predict を付けて実行する。最終的に rsl_predict_mode.csv が予測された結果である。

```
$ more train.csv
tid,word,freq,class
```

```
1,w1,2,M
1,w2,4,M
10,w1,1,F
11,w2,1,F
11,w1,2,F
12,w1,4,M
12,w2,4,M
13,w3,3,M
13,w2,2,M
13,w1,4,M
14,w1,5,M
14,w2,3,M
14,w3,2,M
15,w1,1,F
16,w1,2,F
16,w2,1,F
18,w2,4,F
18,w1,2,F
19,w2,2,F
19,w1,1,F
19,w3,3,F
2,w2,2,M
2,w1,3,M
2,w3,3,M
20,w1,1,F
20,w2,3,F
20,w3,2,F
4,w3,2,M
4,w2,3,M
4,w1,3,M
5,w1,1,F
6,w2,1,F
6,w1,1,F
7,w1,3,M
7,w2,4,M
8,w2,2,M
8,w3,3,M
8,w1,4,M
9,w1,3,M
9,w3,2,M
9,w2,3,M
17,w2,1,M
17,w1,2,M
3,w1,1,F
3,w2,1,F

$ more test.csv
tid,word,freq
21,w1,1
21,w2,2
22,w1,3
22,w2,1

$ mnb.rb tid=tid item=word w=freq class=class i=train.csv O=rsl1 seed=1 cv=

#MSG# separating data 1; 2014/08/19 00:42:10
#MSG# separating data 2; 2014/08/19 00:42:10
#MSG# separating data 3; 2014/08/19 00:42:10
#MSG# separating data 4; 2014/08/19 00:42:10
#MSG# separating data 5; 2014/08/19 00:42:10
#MSG# separating data 6; 2014/08/19 00:42:10
#MSG# separating data 7; 2014/08/19 00:42:10
#MSG# separating data 8; 2014/08/19 00:42:10
#MSG# separating data 9; 2014/08/19 00:42:10
#MSG# separating data 10; 2014/08/19 00:42:10
#MSG# Naive Bayes start using training data 1; 2014/08/19 00:42:10
#MSG# Naive Bayes start using test data 1; 2014/08/19 00:42:10
#END# mnb.rb tid=tid item=word w=freq class=class i=train.csv O=rsl1 seed=1 cv=; 2014/08/19 00:42:11
#MSG# Naive Bayes start using original data; 2014/08/19 00:42:11
#END# mnb.rb tid=tid item=word w=freq class=class i=train.csv O=rsl1 seed=1 cv=; 2014/08/19 00:42:11


$ mnb.rb i=test.csv I=rsl1 o=rsl_predict_model -predict
#MSG# Naive Bayes start using test data; 2014/08/19 00:43:31
#END# mnb.rb i=test.csv I=rsl1 o=rsl_predict_model -predict; 2014/08/19 00:43:31


$ more rsl1/rsl_model.csv
tid,F,M,class,predictCls
```

```
1,0.5149523047,0.4850476955,M,F
10,0.4929065867,0.5070934133,F,M
11,0.5019607343,0.4980392657,F,F
12,0.5089038694,0.4910961304,M,F
13,0.4918393826,0.5081606174,M,M
14,0.4966021486,0.5033978514,M,M
15,0.4929065867,0.5070934133,F,M
16,0.5019607343,0.4980392657,F,F
17,0.5019607343,0.4980392657,M,F
18,0.5149523047,0.4850476955,F,F
19,0.4932580133,0.5067419868,F,M
2,0.4922318437,0.5077681563,M,M
20,0.5008042081,0.4991957921,F,F
3,0.5070546328,0.4929453672,F,F
4,0.4983186364,0.5016813634,M,M
5,0.4929065867,0.5070934133,F,M
6,0.5070546328,0.4929453672,F,F
7,0.5115578733,0.4884421266,M,F
8,0.4918393826,0.5081606174,M,M
9,0.4983186364,0.5016813634,M,M


$ more rsl1/rsl_predict_model
tid,F,M,predictCls
21,0.5134492948,0.4865507052,F
22,0.4989489229,0.5010510771,M
```

2.3 mbonsai 系列データによる決定木生成

本コマンドは、系列パターンを説明変数として利用可能な決定木モデル構築コマンドである。系列データの解析は様々な応用分野において適用できる。顧客が購入したブランドの順序，スーパーや百貨店における売り場の巡回パターン，傷病の発症順序, これらは全て系列データとして解析できる。本コマンドのオリジナルのアイデアは、九州大学の研究チームによって開発された BONSAI[4] にあり、分子生物学におけるアミノ酸配列を解析することを目的に構築された手法である。この手法をビジネスデータに応用する中で、いくつかの改良を加えたものが本コマンドで、以下のような特徴をもつ。

- カテゴリ系列データから有用なパターンを抽出し、その有無を決定木における節点の条件とすることができる。
- モデル精度を高めるように、系列を構成するアイテムをグルーピングすることができる。
- 複数の系列データを扱うことができる。
- 系列データ以外にも数値変数、カテゴリ変数も含めてモデル化ができる。
- 決定木モデルの構築モードと未知データを与えての予測モードがある。
- 3 つ以上のクラス分類問題にも対応している。

まず直感的理解を得るために，以下に例を示す。ある小売店における顧客別のブランド `a,b,c` の購入系列と、その店からの離反の有無についてのデータが表 2.9 に示されている。ブランドの購入順序が顧客の離反に関係しているとの仮説のもと、離反の有無を目的変数に、そしてブランドの購買順序に関する部分的なパターン (ここでのパターンとは部分文字列、例えば、”ab”,”cbb”を考えればよい。パターンの定義の詳細は次節を参照のこと。) を説明変数にして決定木モデルを構築する。このような説明変数としてのパターンは候補パターンと呼ばれ、モデルの精度に寄与するであろうパターンが事前に生成され (その方法も次節を参照のこと)、それらのパターンを 0-1 変数としてデータセットが作成される (表 2.10)。このデータセットから一般的な方法で決定木が構築される。実際に本コマンドを用いて作成された決定木が図 2.11 に示されている。

表 2.9 系列データの例。一行が一人の顧客に対応し、目的変数として、それぞれの顧客が離反したかどうかが示されている。「ブランド系列」項目に示されたアルファベット a,b,c は、顧客が購入したブランドを表しており、それぞれに示された順序で購入したことを意味する。

ブランド系列	離反
bcaba	yes
bcabcaa	yes
aaabac	yes
caa	yes
cca	no
cacbc	no
bcc	no
acca	no

表 2.10 表 2.9 のブランド系列項目から抜き出された部分的な購入パターンを説明変数とし (候補パターンと呼ぶ)、全サンプルに対してパターンを含むかどうかの 0-1 データに変換されたデータセットの例。一行目の顧客は部分パターン”a”,”b”,”c”,”ab”を含んでいるが、”aa”,”cc”は含んでいない。

a	b	c	aa	ab	cc	···	離反
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

さらに BONSAI の特筆すべき特徴として、パターンを構成する要素 (アルファベットと呼ぶ) を自動的にグルーピングする機能がある。ここで、各アルファベットに対するグループをインデックスと呼ぶ。例えば、`a,b,c` の 3 つのブランドを 2 つのインデックスにグルーピングする仕方は 3 通りあるが ((a と bc、b と ac、c と ab)、それぞれのグルーピングに応じてオリジナルの系列のアルファベットを対応するインデックスで置換し、上述の手順で決定木を 3 つ構築し、分類精度の最もよい決定木を選択する。このようにして構築された決定木が図 2.12 に示されている。得られたグルーピングは、分類モデルの精度を高めるようなグルーピングになっているため、そこから有用な知見が得られることが期待できる。

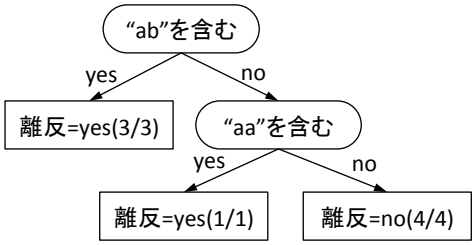

表 2.11　BONSAI による離反顧客モデルの構築例

アルファベット	a	b	c
インデックス	1	2	2

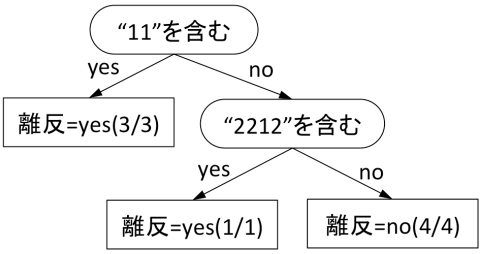

表 2.12　BONSAI による離反顧客モデルの構築例

2.3.1　詳細

以下では、決定木の構築に関する詳細について解説する。

正規パターン

アルファベット Σ 上の n 個の文字列定数を $\pi_1, \pi_2, ..., \pi_n$ とし，任意の $n+1$ 個の文字列を $x_0, x_1, ..., x_n$ とした時, 正規パターン (regular pattern, 単に「系列パターン」とも呼ぶ) は，$x_0\pi_1x_1\pi_2x_2\cdots\pi_nx_n$ の形式で与えられる．データ検索の分野におけるワイルドカードを考えると理解しやすいであろう (ワイルドカードでは x_i の代わりに “*” が使われる)．本コマンドでは、上記の正規パターンの定義以外にも、$x_0\pi x_1$、すなわち部分文字列 π に限定した正規パターンも扱うことができる (以下、「文字列パターン」と呼ぶ)。系列パターンと文字列パターンの切り替えは p=の第 2 パラメータで指定する。

先頭/末尾マッチ

正規パターンの各サンプルの系列データへのマッチングルールとして、先頭一致と末尾一致を指定することができる。それらの指定は p=の第 4(先頭一致), 第 5(末尾一致) パラメータで指定し、文字数によって与える。この指定により、正規パターンの先頭 (末尾) のアルファベットの位置が、系列データの先頭 (末尾) から指定の文字数以内であればマッチしたことになる。例えば、系列データ aabccd において、文字列パターン ab は、先頭一致文字数が 1 であればマッチしないが、2 であればマッチする。ab を系列パターンとすると、先頭一致文字列が 1 であってもマッチすることになる。このパラメータを指定しなければ、以上のようなマッチングの制限はないものとして動作する。

順序アルファベット

順序構造のあるアルファベットを扱うことができる。順序構造を仮定した場合、インデックスの生成規則に違いが出てくる。アルファベット集合 $\{a_1, a_2, \ldots, a_n\}$ について、$a_1 \prec a_2 \prec \cdots \prec a_n$ の関係があるとすると、連続する 3 つの任意のアルファベット a_i, a_{i+1}, a_{i+2} について、$\psi(a_i) = \psi(a_{i+2}) \Rightarrow \psi(a_i) = \psi(a_{i+1})$ の関係が成り立つようにインデックスが生成される。ここで、$\psi(a)$ は、アルファベット a に対応させるインデックスを表す。これは、あるグループに属するアルファベットは必ず連続するようにインデックス化されるということを意味する。例えば、3 つの順序アルファベット $a \prec b \prec c$ について、$\{a, b\}$ と $\{c\}$ にグルーピングすることはあるが、$\{a, c\}$ と $\{b\}$ にグルーピングすることはない。順序アルファベットの指定は p=の第 3 パラメータで指定する。

最適なアルファベットインデックスの探索空間とローカルサーチ

n 個のアルファベットを m 個以下のインデックスにグルーピングする場合の数は自明ではないが、$m = 2$ に限れば、その数は m^{n-1} 通り存在する。m, n が共に小さい場合は、全ての場合について決定木を構築すればよいが、その値が大きい場合は以下に示すローカルサーチの手法によって部分空間を探索している。それは、最初に各アルファベットに

対応するインデックスをランダムに設定し、その対応関係を少しずつ変更しながら決定木を構築し、分類精度に改善がなくなるまで変更を続けるというものである (詳細は文献 [4] を参照のこと)。よって、アルファベットとインデックスの対応関係の初期値によって異なる結果が得られることもある。そこで初期値を複数個用意して得られた結果から最適なモデルを選ぶ方法 (マルチスタート) も用意されている。本コマンドでは `iter=`によって、初期値の個数を指定することが可能である。デフォルトは `iter=1` である。

候補パターンの生成方法

表 2.10 に示されるような候補パターンは、ローカルサーチにおいてアルファベットインデックスが更新される度に新たに生成される。決定木の節点における分類規則は、この候補パターンに基づいてなされるために、その生成方法は重要である。本コマンドでは、以下に示すヒューリスティックな方法で候補パターンを列挙している。まず、インデックスにおける長さ 1 の正規パターンを構築し、優先キューに格納する。優先キューにおける優先順位は、正規パターンのエントロピー (後述) の昇順で決まる。そして、優先キューからエントロピーの最も低い正規パターンを選択し、その正規パターンにインデックスを一つ追加して、長さ 2 の正規パターンを構成し、再び優先キューに格納する。上記の手順を繰り返し、途中、長さ n の正規パターンを選択すれば、長さ $n+1$ の正規パターンが優先キューに格納される。ただし、n が 5 を超えた場合はインデックスの追加は行わない (正規パターンのサイズの上限の変更は `p=`の第 6 パラメータで指定可能)。そして、ユーザが指定した候補数 (`cand=`で指定) を超えれば終了する。正規パターンの評価に用いるエントロピーは、決定木の節点における分岐ルールの選択にも用いられ、正規パターンがクラスを極端な分布に分類すればするほど小さな値になる。正規パターン π のエントロピー $ent(\pi)$ は、式 2.10 で定義される。

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

ここで、c はクラス数を表し、$p_i^{\mathrm{m}(\pi)}(p_i^{\mathrm{u}(\pi)})$ は正規パターン π がマッチした (しなかった) サンプルにおけるクラス i の構成比を表している ($\sum_i^c p_i^{\mathrm{m}(\pi)} = 1$)。また $q^{\mathrm{m}(\pi)}(q^{\mathrm{u}(\pi)})$ は、正規パターン π がマッチした (しなかった) サンプルの全サンプルに対する構成比である ($q^{\mathrm{m}(\pi)} + q^{\mathrm{u}(\pi)} = 1$)。

その他の型の変数

本コマンドでは、系列データ以外にも、数値とカテゴリ変数を説明変数として指定することができる。そうすることで、正規パターンのルールと数値やカテゴリについてのルールが混在した決定木を構築することが可能となる。数値やカテゴリの分岐規則に生成ついては、C4.5 と同様の方法を利用している [5]。本コマンドでは、`p=,n=,d=`によって、系列項目、数値項目、カテゴリ項目の項目名をそれぞれ指定する。

分岐ルールの選択

本コマンドにおける決定木の構築は、2 分岐によるトップダウンの貪欲法を採用している。すなわち、木の節点における分割ルールは、その節点における情報のみに基づいた評価基準に従って決定される。評価基準としては、エントロピーゲインを最大にするような分岐ルールが選ばれる。ある節点に分類されたサンプルについて、クラス i である確率 (構成比) を p_i で表すと、その節点におけるエントロピーは、$ent = -\sum_{i=1}^{c} p_i \log p_i$ で計算される。ここで、p_i は節点に分類されたサンプル数 n とそのうちでクラス i に属するサンプル n_i との比 n_i/n で計算される。そして、式 2.10 で示された正規パターン π により分割した後のエントロピー $ent(\pi)$ との差がエントロピーゲインである。すなわち、これは正規パターン π によって分割することで、エントロピーをどの程度低減させるかを意味している。このような分割を繰り返し、ある節点に分類される全てのサンプルが一つのクラスに属するか、もしくはそれ以上分割できるなくなるまで木を成長させる。このような決定木は最大木と呼ばれる。

枝刈り

前項で示した方法に従って最大木 T_{max} を構築するが、一般的に木のサイズが大きくなると、モデルが訓練データに過適合 (overfitting) することにより、訓練データの分類精度は高まる (誤分類率が下がる) 一方で、未知データへの予

測精度が下がってしまう。この問題を回避するために、最大木よりサイズの小さな部分木 (ただし根節点を含む) を選択する「枝刈り (pruning)」を行う。

決定木 T を節点 t の集合として考え、最大木を $T_{max} = \{t_1, t_2, \ldots, t_k\}$ で、そして節点 t を根節点とした部分木を T_t で表すと、節点 $t \in P$ を根節点とする部分木を全て枝刈りした (葉節点に置き換えた) 決定木は $T_{max} - \bigcup_{t \in P} T_t + P$ で表される。決定木 T の誤分類率を $R(T)$ で表すと、枝刈りとは、未知データに対する誤分類率が最も低くなるような部分木 T^* を選択する問題と考える事ができる。T の未知データに対する真の誤分類率を直接求める事はできないので、訓練データから推定することになる。いま、その推定量を $C(T)$ で表すと、枝刈りは式 2.11 の通り定式化できる。

$$\underset{P \subseteq T_{max}}{\operatorname{argmin}}\, C(T_{max} - \bigcup_{t \in P} T_t + P) \tag{2.11}$$

この問題に対して、本コマンドでは、コスト複雑性枝刈り (cost-complexity pruning) と呼ばれる方法 [6] を採用している。この方法は、大きく二つのフェーズから構成さる。まず訓練データを用いて構築された最大木からネストした一連の部分木 $T_1 \supset T_2 \supset \cdots \supset T_k$ を選択する (ここで T_1 は最大木、T_k は根節点のみからなる木と考えればよい)。そして次に、それらの木の精度をテストサンプル法もしくは交差検証法 (cross validation) によって推定し、最も推定精度の高い木を選択する。

一連の部分木を選択するフェーズにおいては、まず決定木 T の評価関数としてコスト複雑性 $R_\alpha(T) = R(T) + \alpha|\tilde{T}|$ を定義し、この値が小さい程良い木と考える。この式は、決定木の誤分類率 $R(T)$ と決定木の複雑性 $|\tilde{T}|$(T の葉節点の数) のトレードオフを複雑性パラメータ $\alpha(\geq 0)$ によってバランスさせたものである。訓練データにおいては、木のサイズが大きくなるに従い誤分類率 $R(T)$ は単調に減少するが、逆に複雑性 $|\tilde{T}|$ は単調に増加する。そして α を調整する事で、誤分類率と複雑性のどちらを優先するかが調整される。ここで、α を一つの値に固定すると、$R_\alpha(T)$ を最小化しかつ最小サイズの部分木 $T(\alpha)$ がただ一つ存在することが知られている。さらに、α を小さい区間を単位として変化させていったとき、それぞれの α に対応する $T(\alpha)$ を列挙でき、結果として、ネストした部分木列 $T_1 \supset T_2 \supset \cdots \supset T_k$ が構築される (より詳細は [6] を参照のこと)。ここで、部分木 T_i は $\alpha \in (\alpha_i, \alpha_{i+1}]$ においてコスト複雑性最小かつ最小サイズの決定木である。以上により、ネストした一連の部分木と、それぞれの木に対応するコスト複雑性パラメータ $\alpha_1, \alpha_2, \ldots, \alpha_k$ が得られる。

そして次に、以上の方法により得られた一連の部分木の中から、未知データに対する推定誤分類率を最も低めるような最適な部分木 $\hat{T}^*$ を選択する。本コマンドでは、誤分類率の推定にテストサンプル法 (`ts=`を指定) もしくは交差検証法 (`cv=`を指定) を選ぶことができる。テストサンプル法では、訓練データ D を 1 : 2 の割合で D_1, D_2 に分割する。そして、D_2 を訓練データとして最大木を構築し、前フェーズで得た複雑性パラメータ $\alpha_1, \alpha_2, \ldots, \alpha_k$ それぞれに対応する部分木を選択し、その部分木によって D_1 を未知データとして予測した結果を未知データの誤分類率の推定値とする。交差検証法においては、訓練データ D を均等に n 分割し、$D_1, D_2, \ldots, D_n$ を得る。最初に D_1 を未知データとして、そしてその他を訓練データとしてテストサンプル法と同様の推定を行う。同様に残りの D_2〜D_n をそれぞれテストデータとして、同様のプロセスを n 回実施することで、全ての訓練データ D を一回ずつ未知データとして検証する事になる。そのようにして得られた平均誤分類率を未知データの推定値とする。上記の結果得られた一連の複雑性パラメータ $\alpha_1, \alpha_2, \ldots, \alpha_k$ に対応する決定木 $T_1, T_2, \ldots, T_k$ のうち、誤分類率が最も低い決定木を最適な決定木として選択する。また、最小誤分類率に標準誤差を加えた誤分類率より小さい部分木の中から、サイズの最も小さいサイズの木を選ぶ事も可能である (これは「1SE ルール」と呼ばれる)。

なお、テストサンプル法もしくは交差検証法を用いず、α の値を直接指定する事で枝刈りを実現する事もできる。この方法は、推定のための計算が不要なために決定木を高速に構築できるが、一方で α の値の指定が恣意的になるという欠点がある。

枝刈りの実際

本コマンドの枝刈りは、モデル構築モードにおいて、`ts=`、`cv=`、`alpha=`のいずれかのパラメータを指定する事で実施される。`ts=`もしくは `cv=`が指定された時は、テストデータを使った推定により、誤分類率最小のモデルが選択される。また、`alpha=`を指定した時は、指定の α に応じた枝刈りのモデルが選択される。`model.txt` と `model_info.txt` に出力される決定木やモデル評価は、選択されたモデルに基づいている。ただし、いずれの指定においても、内部では

最大木と全ての α に対する枝刈りを計算しており、予測モード (`-predict`) において、再度 `alpha=`を指定することで、異なるモデルを用いた予測も可能となる。

コスト考慮学習

分類モデルの応用においては、分類精度 (正答率) を高めるのではなく、誤分類によって生じるコスト (誤分類コスト) を低減させる方が有用なケースが多くある。離反顧客を離反しない顧客と予測した場合のコストと、離反しない顧客を離反すると予測した場合のコストは、それぞれで実施する施策を考慮するとそのコストは異なるかもしれない。このようなコストを考慮に入れてモデルを構築することを、一般的にコスト考慮学習 (cost sensitive learning) と呼ぶ。その方法は多数提案されているが、ここでは、Breiman らの提案する方法を用いている [6]。この方法は、分岐ルールの計算において、サンプルがクラス i である確率 p_i の計算をコストによる加重により修正するというものである。いま、クラス j をクラス i と予測したときのコストを $c(i|j)$ で表すと、クラス i のコスト合計 $\sum_j c(i|j)$ をクラス i の加重として確率 p_i を修正する。コスト計が大きいクラス i は、常にサンプル数が水増しされることになり (オーバーサンプリング)、クラス i の情報に敏感なモデルが構築されることになる。クラスファイルは表 2.13 に示されるように、実クラス (`real`) に対する予測クラス (`predict`) のコスト (`cost`) を一行とする CSV ファイルで指定する。実クラスと予測クラスの特定の組み合わせを省略した場合、もしくはコストファイルそのものを省略した場合、実クラスと予測クラスが同じ場合のコストは 0、異なる場合は 1 と設定される。

表 2.13 離反モデルにおけるコストの指定例。1 行目は、離反クラスが yes のサンプルを no と予測した場合のコストが 2 で、2 行目は、離反クラスが no のサンプルを yes と予測した場合のコストが 5 と設定されている。項目名は任意であるが、順序は、実クラス、予測クラス、コストの順番でなければならない。

real	predict	cost
yes	no	2
no	yes	5

2.3.2 出力データ

本コマンドで出力される各種データファイルが表 2.14 にまとめられている。

表 2.14 本コマンドがモデル構築モードで出力するデータ一覧

ファイル名	内容	備考
`model.pmml`	PMML[a] による決定木モデル	枝刈り情報を伴った最大木が出力されている。`-predict` 指定時の予測モードで利用される。
`alpha_list.csv`	複雑度パラメータ α 別のモデル情報	一連の α に応じたモデルの精度やサイズなど。
`model_min.txt`	推定誤分類率最小の枝刈りモデルの要約	`cv=` もしくは `ts=`を指定したときのみ出力される。
`model_1se.txt`	同じく 1SE ルールの枝刈りモデルの要約	`cv=` もしくは `ts=`を指定したときのみ出力される。
`model.txt`	指定の α による枝刈りモデルの要約	
`model_info_min.csv`	推定誤分類率最小の枝刈りモデルの各種情報	`cv=` もしくは `ts=`を指定したときのみ出力される。
`model_info_1se.csv`	同じく 1SE ルールの枝刈りモデルの各種情報	`cv=` もしくは `ts=`を指定したときのみ出力される。
`model_info.csv`	指定の α による枝刈りモデルの要約	
`predict_min.csv`	推定誤分類率最小の枝刈りモデルによる予測	`cv=` もしくは `ts=`を指定したときのみ出力される。
`predict_1se.csv`	同じく 1SE ルールの枝刈りモデルによる予測	`cv=` もしくは `ts=`を指定したときのみ出力される。
`predict.csv`	指定の α による枝刈りモデルによる予測	
`param.csv`	実行パラメーター一覧	パラメータのキーワードと値のペアを出力

[a] Predictive Model Mark-up Language の略で、数理モデルを XML により表現する業界標準である。PMML に準拠はしているが、本コマンド特有の拡張タグも利用していることに留意されたい。

■model.pmml　ここで出力される決定木は最大木で、各節点には以下に示されるように complexity penalty 属性が示されており、α がその値より大きい場合に、その枝は刈られる事になる。PMML にモデルとしての最大木と枝刈り情報を記録しているので、予測モードでの実行時に α を指定することで、その値に応じた決定木モデルにより予測することが可能となる。

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

■model_min.txt,model_1se.txt,model.txt　PMML データと異なり、枝刈り後のモデルの要約がテキスト形式で出力される。[alphabet-index] の欄には、アルファベットとインデックスの対応関係が示されている。以下の例では、アルファベット c がインデックス 1 に、b,a がインデックス 2 に対応している。決定木に表示されるパターンの分岐規則はこのインデックス記号によって示される。

```
[alphabet-index]
Field Name: ブランド系列
Index[1]={c}
Index[2]={b,a}
```

[decision tree] の欄には、決定木がテキスト形式で示され、その下にはモデルサイズ (葉節点の数) と最も深い葉の階層数が示される。

```
[decision tree]
if($ブランド系列 has 22)
  then $離反=yes (hit/sup)=(4/4)
  else $離反=no (hit/sup)=(4/4)

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

[Confusion Matrix by Training] の欄には、訓練データを決定木モデルで予測した場合の分類結果が示されている。また、コストによる分類表、クラス別の予測精度、全体の予測精度も示されている。[Confusion Matrix by Estimation] の欄は、ts=もしくは cv=を指定した時のみ出力される。内容は [Confusion Matrix by Training] の欄と同様であるが、テストデータによる結果であることが異なる点である。

```
[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
```

最後に [Selected Alpha] の欄には、枝刈りにおいて採用された複雑度パラメータの値が出力されている。

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

■predict_min.txt,predict_1se.txt,predict.txt　モデル構築に用いた訓練データに決定木モデルによる予測結果を追加した CSV データである。予測結果は、以下に示すように、predict 項目に予測確率の最も高かったクラス名が出力され、さらに各クラス毎 (以下では yes と no) に予測確率が出力される。ts=が指定された場合は、テストデータの予測結果が出力され、cv=が指定された場合は、交差検証における全テストデータの予測結果が出力される。また、alpha=が指定された場合は、訓練データそのものの予測結果が出力される。

```
ブランド系列, 離反,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　これらのファイルは、モデル情報を CSV 形式で出力したものである。nobs は訓練データの件数、alpha=は枝刈りの複雑度パラメータの値、accuracy,totalCost はテストデータにおけるモデルの正答率と総コストである。

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

■alpha_list.csv　枝刈りの複雑度パラメータ α の値別に、それぞれに対応する枝刈りされた決定木のエラー率、標準誤差、エラー率±標準誤差の値が示されている。

```
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　モデル構築時に用いられた各種パラメータの値が CSV 形式により格納されている。

2.3.3 書式 1:モデル構築モード

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

i=	: 訓練データファイル名
p=	: パターン項目名 (複数指定可)
	: 項目名の後にコロンで区切って 5 つのパラメータを指定できる。
	: p=**項目名**:is:seq:ordered:head:tail:rs
	: is: インデックスのサイズ
	: 指定を省略すれば、インデックスを生成せず、オリジナルのアルファベットのパターンを用いる。
	: seq: パターンの種類
	: true:部分系列パターン
	: false:部分文字列パターン (default)
	: ordered: インデックスの生成において、アルファベットに順序があると仮定するかどうか。
	: (is の指定を省略していれば無視される)
	: true: 順序あり, アルファベットをある閾値以上/未満でグルーピングする。
	: false: 順序なし (default)
	: head: 先頭一致の先頭からの文字数 (デフォルトは先頭一致を考慮しない)
	: tail: 末尾一致の先頭からの文字数 (デフォルトは末尾一致を考慮しない)
	: rs: 正規パターンのサイズ上限 (デフォルトは 5)
n=	: 数値項目名 (複数指定可)
d=	: カテゴリ項目名 (複数指定可)
c=	: クラス項目名
O=	: 出力ディレクトリ名 (text,PMML によるモデル, モデル統計など)
delim=	: パターンの区切り文字 (デフォルト:空文字, すなわち 1 バイト文字を 1 つのアルファベットと見なす)
cost=	: コストファイル名
seed=	: 乱数の種 (default=-1:時間依存)
cand=	: 説明変数としてのパターン数 (default=30, 範囲:1〜256)
iter=	: ローカルサーチ回数 (default=1)
leafSize=	: 一つのリーフに含まれるサンプル数の下限 (default:制限なし)
alpha=	: 枝刈り度を指定する。省略時は 0.01 が設定される。ただし、ts=もしくは cv=を指定した場合、
	: このパラメータは無効化される。
ts=	: テストサンプル法によるテストデータの割合を指定する。値を省略して"ts="と指定すると 0.333 が用いられる。
cv=	: 交差検証法によるデータの分割数を指定する。値を省略して"cv="と指定すると 10 が用いられる。
	: ts=,cv=のいずれも指定されていない場合は、alpha=0.01 と指定されたと見なされる。
	: alpha=,ts=,cv=を指定したとしても、PMML に最大木と枝刈り度が記録されるので、
	: 予測モードにおいて、alpha の値を変更して予測することが可能である。
--help	: ヘルプの表示

2.3.4 書式 2:予測モード

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

`-predict` : 予測モードで動作させる。予測モードでは必須
`i=` : 入力データ【必須】
: 項目名は、モデル構築モードで利用した項目と同じでなければならない。
`I=` : モデル構築モードでの出力先ディレクトリパス【必須】
: 利用するファイルは以下のとおり。
: bonsai.pmml: pmml による決定木モデル
`o=` : 予測結果ファイル名
: 入力データに”predict”という項目を追加して出力する。
: 項目名は、モデル構築モードで利用した項目と同じでなければならない。
`alpha=` : 枝刈りの複雑度パラメータ。
: 0 以上の実数値を与える以外に、以下の 2 つは特殊な意味を持つシンボルとして指定できる。
: min: 推定誤分類率最小の枝刈りモデルに対応する α の値。
: 1se: 同じく 1SE ルールの枝刈りモデルに対応する α の値。
: これら 2 つのシンボルによる指定は、モデル構築時に `ts=`もしくは `cv=`を指定した時のみ有効である。
: 省略時の動作:
: モデル構築時に `ts=`もしくは `cv=`を指定した場合は、`min` が用いられる。
: モデル構築時に `alpha=`を指定した場合は、その値が用いられる。
`delim=` : パターンの区切り文字 (デフォルト:空文字, すなわち 1 バイト文字を 1 つのアルファベットと見なす)
`--help` : ヘルプの表示

2.3.5 利用例

例 1 モデル構築の例

ここでは、`ts=,cv=,alpha=`のいずれも指定されていないので、`alpha=0.01` で枝刈りされた結果が `model.txt` に出力される。

```
$ more input.csv
ブランド系列,離反
bcaba,yes
bcabcaa,yes
aaabac,yes
caa,yes
cca,no
cacbc,no
bcc,no
acca,no

$ mbonsai p=ブランド系列:2 c=離反 i=input.csv O=result1

$ more result1/model.txt

[alphabet-index]
Field Name: ブランド系列
Index[1]={a}
Index[2]={b,c}

[decision tree]
if($ブランド系列 has 11)
  then $離反=yes (hit/sup)=(3/3)
  else if($ブランド系列 has 2212)
    then $離反=yes (hit/sup)=(1/1)
    else $離反=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="ブランド系列" optype="categorical" dataType="string">
      <Value value="b"/>
      <Value value="c"/>
      <Value value="a"/>
    </DataField>
    <DataField name="離反" optype="categorical" dataType="string">
      <Value value="yes"/>
      <Value value="no"/>
    </DataField>
  </DataDictionary>
  <TreeModel functionName="classification" splitCharacteristic="binarySplit">
    <MiningSchema>
      <MiningField name="ブランド系列">
        <Extension extender="KGMOD" name="alphabetIndex">
          <alphabetIndex alphabet="b" index="2"/>
          <alphabetIndex alphabet="c" index="1"/>
          <alphabetIndex alphabet="a" index="2"/>
        </Extension>
      </MiningField>
      <MiningField name="離反" 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="ブランド系列" 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="ブランド系列" 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>
```

例 2 例 1 のモデルから未知データを予測する

構築した決定木モデルで未知データ (以下では訓練データを未知データとして使っている) を予測する例。予測結果としては、各クラスの予測確率 (yes,no 項目)、およびその確率が最も高いクラス名 (predict 項目) が出力される。

```
$ more unknown.csv
ブランド系列
bcaba
bcabcaa
aaabac
caa
cca
cacbc
bcc
acca

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

$ more predict.csv
ブランド系列,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 グラフ分割コマンド

本コマンドは、ミネソタ大学で開発されたグラフ分割ソフトウェア METIS (http://glaros.dtc.umn.edu/gkhome/metis/metis/overview) を利用しやすいように作成されたコマンドである。METIS によるグラフ分割では、与えられた無向グラフについて、各パーティション (以下クラスタとも呼ぶ) に含まれる節点数をできるだけ同じにして、枝のカット数を最小化するように複数のパーティションに分割する。

METIS が直接扱うデータ構造は、節点を整数で表したフォーマットであるが、本コマンドでは任意の文字列を扱うことができる。またグラフデータは、特殊なフォーマットを仮定せず、枝データと節点データを別々に CSV データとして与える。内部では、gpmetis コマンドをコールしており、前処理として、ユーザが与えたグラフデータを gpmetis が扱えるデータに変換している。データ変換だけを実行することも可能である。

入力データは，表 2.15 に示されるように、一行が一つの枝を表す節点ペアとして与えられた CSV ファイルである。孤立した節点がなく、節点の重み (後述) を指定しない場合は枝データのみ与えればよい。対応するグラフが図 2.1 に示されている。このグラフを 2 分割したければ、以下のようにコマンドを実行すればよい。

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

結果として、表 2.16 に示されるように、節点名とその節点が属するクラスタ番号が出力される。図 2.2 には、最小本数の枝をカットして 2 分割されている様子が示されている。

表 2.15　入力データ (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

表 2.16　分割結果データ (output.csv)

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

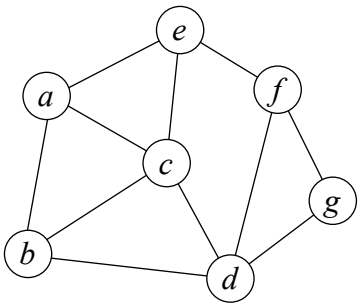

図 2.1　分割対象のグラフ

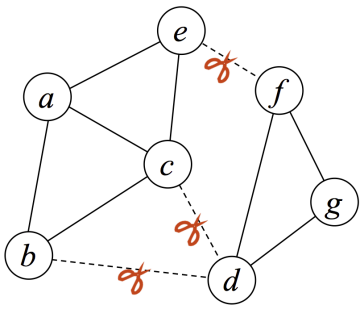

図 2.2　最小カットによる 2 分割の結果。7 つの節点があるので、最も均等に分割したとして 4:3 の分割になるが、図に示された 3 つの枝をカットするのが最もカット数が少ないカット方法である。

2.4.1 書式

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

```
ei=        : 枝ファイル名 (節点ペア)【必須】
ef=        : 枝ファイル上の節点ペア項目名 (2 項目のみ)【デフォルト:"node1,node2"】
ew=        : 枝ファイル上の重み項目名 (1 項目のみ)【オプション:省略時は全ての枝の重みを 1 と見なす】
           : 重みは整数で指定しなければならない。
ni=        : 節点ファイル名【オプション】
nf=        : 節点ファイル上の節点項目名 (1 項目のみ)【デフォルト:"node"】
nw=        : 節点ファイル上の重み項目名 (複数項目指定可)【オプション:省略時は全ての重みを 1 と見なす】
           : 重みは整数で指定しなければならない。
o=         : 出力ファイル名【オプション:default は標準出力】
kway=      : 分割数【必須】
ptype=     : 分割アルゴリズム【デフォルト:kway】
balance=   : パーティション一様化パラメータ【デフォルト: ptype=rb の時は 1.001、ptype=kway の時は 1.03】
           : 式 2.13 における β の値を指定する。
ncuts=     : 分割フェーズで、初期値を変えて試行する回数【オプション:default=1】
dat=       : 指定されたファイルに gpmetis コマンド用のデータを出力する。
map=       : 指定されたファイルに gpmetis コマンド用の節点番号と i=上の節点名のマッピングデータを出力する。
-noexe     : 内部で gpmetis を実行しない。dat=,map=の出力だけが必要な場合に指定する。
--help     : ヘルプの表示
```

2.4.2 アルゴリズム

gpmetis によるグラフ分割アルゴリズムは、1) 粗化 (coarsening)、2) 分割 (partioning)、3) 逆粗化 (uncoarsening) の 3 つのプロセスに分けることができる。粗化においては、枝で接続された複数の節点を統合する過程で、数百の節点から構成される小さなグラフに縮約するプロセスである。そして粗化されたグラフについて、できるだけパーティションサイズを均等にして (統合された節点数を考慮に入れて)、枝のカット数が最小となるように分割する。そして最後に、統合された節点を元に戻す逆粗化によって全節点を分割することになる 2.3。

gpmetis のアルゴリズムの特徴は以下のとおりである。グラフ分割問題は NP 完全であることが知られており、グラフサイズが大きくなると求解が困難となる。そこで、gpmetis は、前処理でグラフサイズを小さくする粗化によってこの問題を回避している。しかし、粗化することで分割の自由度が小さくなるため、一般的には分割精度 (後述の目的関数により定義される枝カット最小化基準) が悪くなる。そこで逆粗化の過程で局所の分割構造を改善する (refinement) ことで粗化に伴う精度の悪化を補う。gpmetis で採用されているアルゴリズムは近似アルゴリズムであり、最適解が得られる保証はないことに注意する。

問題設定

無向グラフ $G = (V, E)$ について、節点集合 V を k 個のパーティション $V_1, V_2, \ldots, V_k$ に分割したい。この時、パーティションをまたいだ枝数を最小化 (枝カット最小化) し、かつ各パーティションに属する節点の数をできるだけ一様にした分割を考える。より一般的に、頂点 u, v に張られた枝 $(u, v) \in E$ の重みを $w(u, v) \geq 0$、節点 v に与えられた重みを w_v で表すと、枝カット最小化の目的関数は、式 2.12 の通り定式化できる。また、パーティションの一様性は、一様化パラメータ $\beta \geq 1.0$ を導入した制約条件として定式化される (式 2.13)。一様化は、パーティションあたりの平均節点数 (重み) に対する、最大節点数 (重み) の比として定義され、β を大きくすれば、よりアンバランスな分割が許容される。なお、β はコマンドパラメータ `balance=`で指定できる。

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

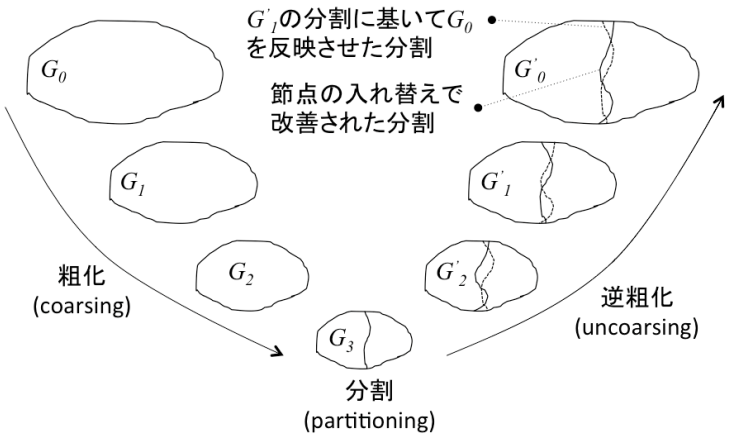

図 2.3　マルチレベルグラフ分割の概念図 (文献 [2] の Figure1 を編集)。粗化の過程では、節点数が数百になるまで、グラフマッチングに基づいて節点を併合することでオリジナルグラフを縮約していく。縮約されたグラフに対して、節点数をできるだけ一様にして、枝のカット数を最小化するような分割を見つける。そして、逆粗化の過程で、併合された節点を元に戻していく。その際、パーティション間で節点の入れ替えを行うことで、最小カットを改善する。図中、G_2 と G'_2 では、構成される節点集合は同じであるが、分割のためにカットされた枝集合が異なる。

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

再帰二分割法と k-Way 分割法

gpmetis が採用しているアルゴリズムの特徴は、NP 完全であるグラフ分割問題に対して、最適解を求めるのではなく、マルチレベル分割法 (multi-level patitioning) を採用することで、グラフサイズが大きくなっても現実的な時間で計算できるようにしていることにある。マルチレベル分割法とは、次のような複数のフェーズにより構成されるアルゴリズムの一般的呼称である。すなわち、オリジナルのグラフを縮約 (以下「粗化 (Coarsing)」と呼ぶ) することで、十分に小さいサイズにしてからグラフ分割を実施し、その後、分割精度 (枝カット最小化) を改善させながら粗化グラフを元のサイズのグラフに戻していく (以下「逆粗化 (Uncoarsing)」と呼ぶ)。

gpmetis コマンドには、異なる 2 つの分割アルゴリズム、マルチレベル再帰二分割法 (multi-level recursive bisectioning) とマルチレベル k-way 分割法 (multi-level k-way partitioning) が実装されている。節点集合 V を k 分割する時、再帰二分割方では、オリジナルのグラフを「粗化、分割、逆粗化」によって 2 分割し、それぞれのパーティションを更に再帰的に分割していく[*1]。再帰二分割法を Algorithm1 に示す。一方で k-way 分割法では、粗化した後にそのグラフを直接 k 分割し、そして逆粗化して終了する。k-way 分割法を Algorithm2 に示す。いずれか一方が常に優れた分割となるということはないが、実行時間については、再帰的な分割を行わないため k-way 分割法の方が優れている[*2]。

以下では、Algorithm1,2 に示された各サブ関数、Coarsen()、2WayPartition()、KwayPartition()、Uncoarsen()、UncoarsenKway() について、以下でその概要のみを示しておく。詳細についていは [2] を参照されたい。

粗化 (Coarsen 関数)

粗化の目的は、分割を効率的に行うためにグラフのサイズを小さくすることである。粗化のアルゴリズムは、再帰二分割法、k-way 分割法で共通である。粗化においては、オリジナルグラフ $G_0 = (V_0, E_0)$ から、極大マッチング M を求めそのマッチングの各要素 (枝) を併合して新たな節点とすることで、新たなグラフ $G_1 = (V_1, E_1)$ を生成する。こ

[*1] 再帰分割法では k が 2 のべき乗でなくても、パーティションの重みを計画的にアンバランスにすることで、全体としてバランスのとれた分割が可能となる。例えば、$|V| = 9, k = 3$ であれば、最初の二分割で 3:6 に分割し、そして 6 のグラフを再帰的に 3:3 に分割すればよい。ただし、Algorithm1 では、その詳細は示していない。

[*2] $k = 256$ で概ね 3〜4 倍高速との報告がある [3]。

Algorithm 1 k 分割グラフ分割アルゴリズム: **再帰二分割法**

```
function BISECT(G, P)
    G : 分割対象グラフ, P : パーティション集合
    if G のサイズが 1/k then
        P = P ∪ {G}
    else
        C=Coarsen(G)                          ▷ 粗化
        C_1, C_2 = 2wayPartition(C)           ▷ 粗化グラフを 2 分割する
        G_1, G_2 = Uncoarsen(C_1, C_2)        ▷ 逆粗化
        P=BISECT(G_1, P)                      ▷ G_1 で再帰呼び出し
        P=BISECT(G_2, P)                      ▷ G_2 で再帰呼び出し
    end if
    return P
end function
```

Algorithm 2 k 分割グラフ分割アルゴリズム: k**-way 分割法**

```
function KWAY(G)
    G : 分割対象グラフ
    C=Coarsen(G)                              ▷ 粗化
    C_1, C_2, ..., C_k = KwayPartition(C)     ▷ 粗化グラフを k 分割する
    G_1, G_2, ..., G_k = UncoarsenKway({C_1, C_2, ..., C_k})   ▷ k 分割そのまま逆粗化
    return {G_1, G_2, ..., G_k}
end function
```

こで、グラフ $G = (V, E)$ におけるマッチング M とは、枝集合 E の部分集合で ($M \subseteq E$)、任意の 2 つの枝 $e_1, e_2 \in M$ が頂点をお互いに共有しないような枝集合のことである。そして、マッチング M に、それ以上枝を追加できないようなマッチングを、特に極大マッチングと呼ぶ。以上の操作を節点サイズが数百になるまで再帰的に適用し、一連の祖化グラフ $G_1, G_2, \ldots, G_m$ が作成される。gpmetis では極大マッチングを求めるアルゴリズムがいくつか用意されているが詳細は省く。ただし、いずれも乱数に基づくヒューリスティックな方法であることを記しておく。

分割 (2WayPartition 関数、KwayPartition 関数)

ここでは、粗化されたグラフ $G = (V, E)$ を 2 つのパーティションに分割する方法 (2WayPartiotion 関数) について説明する。k 個のパーティションへの分割 (KWayPartiotion 関数) は、2 分割の方法を再帰的に適用することで実現される。gpmetis で採用されている 2 分割のアルゴリズムは効率性を重視した非常に単純な方法である。まず節点集合 V から初期節点をランダムにひとつ選び、それに接続される節点を次々と加えていき、半分の節点 (重み合計) を加えた時点で終了するというものである。ここで、節点を加えていく節点集合を P で表すと、P に加える節点 $v \in V \setminus P$ の選び方に、GGP(Graph Grawing Algorithm) と GGGP(Greedy Graph Growing Algorithm) の 2 つの方法がある。GGP では、ランダムに節点 v を選択する。一方、GGGP では、P への接続と $V \setminus P$ への接続との差が最も大きくなるような節点 v、すなわち式 2.14 で計算されるゲイン g_v が最も大きくなるような節点 v を選ぶ。式中 $w(v, u)$ は節点 v と u に張られた枝の重みを表す。GGGP は、カット数を減らすような節点 v を greedy に選択していることになる。gpmetis では、デフォルトで GGGP が用いられるが、節点の重み制約を複数指定した場合には GGP が用いられる。

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

逆粗化 (Uncoarsen 関数、UncoarsenKway 関数)

粗化のプロセスで得られた一連の粗化グラフ $G_1, G_2, \ldots, G_m$ を逆方向 ($G_m, G_{m-1}, \ldots$) にたどり、併合された節点と枝を元に戻していく。分割プロセスで G_m は既に k 個のパーティションに分割されているが、たとえその分割が最適なものであっても、G_m を G_{m-1} に戻したとき、最適な分割でなくなるかもしれない。そこでヒューリスティックな手法により、いくつかの節点をパーティション間で移動させることで分割精度 (枝カット最小化) を改善させる。この

ような過程を G_0 が得られるまで繰り返し、その時の分割を最終解とする。

その他のパラメータ

粗化、分割、逆粗化のいずれも乱数を用いたアルゴリズムのため、乱数系列によって異なる結果が得られる。そこで ncuts=パラメータを指定することで、粗化から逆粗化を複数回実行し、最も優れた分割を選ぶことができる。再帰二分割法では、Algorithm1 の 6,7,8 行目、k-way 分割方では、Algorithm2 の 3,4,5 行目を指定された回数繰り返す。

gpmetis では、以上に紹介した以外にもアルゴリズムの動作を制御するパラメータがいくつも用意されている。本コマンドでは、そのうちの代表的なもののみを扱うことが可能である。もし本コマンドで扱えないパラメータを設定したければ、dat=で gpmetis 用のデータ (節点番号を整数で表したデータ) を出力することができるのでそのデータを直接 gpmetis で処理させればよい。

2.4.3　利用例

例 1 上記「解説」の例

再帰二分割法 (ptype=rb) によって 2 つのパーティションに分割する。コマンドで ef=にて項目名を指定していないのは、枝データの項目名がデフォルトの node1,node2 になっているからである。

```
$ 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
```

例 2 節点と枝の重みを指定する例

edge.csv ファイルの v 項目を枝の重みとして、node.csv ファイルの v 項目を節点の重みとして利用している。

```
$ 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
```

参考文献

[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）, パターン認識と機械学習（下）：ベイズ理論による統計的予測, 13 章, 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.