Python による演習は、原則として「ML 演習番号.ipynb」ファイルが解答です。たとえば実践演習 1-1 の解答は「ML1-1.ipynb」です。

以下では、Weka を用いた演習の解答例を示します。

実践演習 3-1

minNumObj=1, unpruned = True にすることで精度 100% が実現できます。

実践演習 3-2

minNumObj=3, unpruned = False にすることで精度 96% が実現できます。これに加えて useMDLCorrection（数値特徴を分割するときの基準）を False にすると、精度 96.67% が実現できます。

実践演習 3-3

minNumObj=21 にすることで精度 73% が実現できます。このときの木は、葉に近い purpose の特徴で広がっているので複雑そうに見えるだけで、全体としては比較的単純です。minNumObj=25 にすると精度 72.1% に落ちますが、木はさらに単純になります。

実践演習 4-1

学習後、Classifier output ペインの Classifier model 以後が条件付き確率表です。(rainy,hot,high,TRUE) に対する yes の確率は、以下のようにして求められます。事前確率を掛けるのを忘れないように。また、すべてラプラス推定なので、事前に各値にたいして 1 事例あるものとして計算します。

```
-->y=(4/12)*(3/12)*(4/11)*(4/11)*(10/16)
 y  =
   0.0068871

-->n=(3/8)*(3/8)*(5/7)*(4/7)*(6/16)
 n  =
   0.0215242

--> y/(y+n)
 ans  =
   0.2424055
```

なお、Weka の BayesNet を用いて学習させ、結果表示の画面から XML 形式でベイジアンネットワークを保存し、ベイジアンネットワークエディタで読み込んで値を設定することでも計算結果を確認できます（ただし、BayesNet 学習時に 1 点注意が必要）。

実践演習 4-2

カテゴリカルデータに対しては、実践演習 4-1 と同じ結果が得られています。数値データに対しては、1 次元正規分布の平均と標準偏差が得られています。

実践演習 4-3

省略

実践演習 4-4

学習結果は、すべての特徴が play を親とする形のネットワーク（教科書 p.86 図 4.7(a)）になります。これは各特徴が独立であることを表現しているので、ナイーブベイズと等価です。

実践演習 4-5

教科書 p.86 図 4.7(b) のようになります。

実践演習 5-1

10-fold CV において、ナイーブベイズで 96% という結果が出ます。ナイーブベイズには調整するパラメータはありません[*1]。また、ロジスティック識別では 94% という結果が出ます。

これより、iris のような識別しやすいデータでは生成モデルと識別モデルはあまり違いがないことがわかります。

実践演習 5-2

10-fold CV において、ナイーブベイズで 48.6%、ロジスティック識別で 64.0% という結果が出ます。

これより、glass のような識別しにくいデータでは、識別モデルが有効な場合があることがわかります。

実践演習 6-1

Weka の MultilayerPerceptron で、GUI パラメータを True にして実行すると、入力層のノードがどのような値に対応しているのかを見ることができます。2 値特徴は windy=FALSE のように、どちらかの値を表すノード 1 つに変換されています。3 値（以上）の特徴は、それぞれの値についてノード 1 つが対応しています。

実践演習 6-2

trainingTime を 3000 回にすると、正解率が 100% になります。そのときの重みの値は、trainingTime が 500 回のときと比べて大きな値を取るものが多くなっています。

実践演習 6-3

ハイパーパラメータを調整した結果 (10-fold CV) は以下のようになります。

[*1] useKernelEstimator を True にすると、正規分布の最尤推定を行う代わりに、カーネル密度推定を行います。ただし、結果はあまり変わりません。

hiddenLayers	acc.(%)
5	63.1
6	65.0
7	68.2
8(auto)	67.8
9	69.2
10	66.4
11	69.6
12	69.6

実践演習 7-1

手順は p.125 例題 7.3 の通りで、データが ReutersCorn から ReutersGrain に変わるだけです。結果は、Poly Kernel 1 次で、F 値 0.771 となります。Kernel の次数を 2 次にすると、識別率が落ちてしまいます（おそらくデータ数不足)。RBF カーネルに変更してもあまり改善しないので、この程度のデータ量であれば Poly Kernel 1 次が適切であることがわかります。

また、stopwordsHandler の値を Rainbow にして単語ベクトルを構成すると、F 値 0.824 となります。一方、TFIDF を適用すると性能は下がる傾向にあります。

実践演習 8-1

データのページ下部にある Variables からデータを取得します。TeraPad などのエディタを用いる場合は、矩形削除や置換機能（空白 4 つを, に置換）をうまく使って arff を作成します。Excel を用いる場合は、シートにコピーした後、データツール→区切り位置を用いてデータ 1 つが 1 つのセルに収まるようにして、csv ファイル形式で保存してから、エディタで arff ファイルとして編集します。なお、欠損値が 2 つあるので、これらを?としておきます。

Use training set で以下のような回帰式が得られ、元ページの Parameter Estimates の値と一致していることが確認できます。

```
LPRICE2 =
      0.0012 * WRAIN +
      0.6164 * DEGREES +
     -0.0039 * HRAIN +
      0.0238 * TIME_SV +
    -12.1453
```

実践演習 8-2

相関係数 0.90 で正則化係数を変えても結果はほとんど変わりません。cpu データでは、CACH と CHMAX の係数が他の特徴と比べて極端に大きく、正則化がほとんど学習結果に影響しません。

実践演習 8-3

カテゴリカル特徴と数値特徴が混在した入力に対する回帰問題です。カテゴリカル特徴の値は、回帰式に対する切片の調整として用いられています。

実践演習 8-4

maxDepth の値を変えて、木の大きさを調整します。

maxDepth	cor.
3	0.752
4	0.788
5	0.793
6	0.795
7	0.796
8	0.796

木の大きさは、枝刈りの有無で調整することもできます。maxDepth の値を-1（無制限）として noPruning を False の場合で相関係数 0.796、True にして枝刈りを止めると相関係数 0.902 となります。

実践演習 8-5

デフォルトの設定（minNumInstances=4）で以下の木が得られ、相関係数 0.931 となります。minNumInstances を 2 として、さらに枝刈りを止めると相関係数 0.949 が得られますが、木はとても複雑なものになります。

```
CHMIN <= 7.5 : LM1 (165/12.903%)
CHMIN >  7.5 :
|   MMAX <= 28000 :
|   |   MMAX <= 13240 :
|   |   |   CACH <= 81.5 : LM2 (6/18.551%)
|   |   |   CACH >  81.5 : LM3 (4/30.824%)
|   |   MMAX >  13240 : LM4 (11/24.185%)
|   MMAX >  28000 : LM5 (23/48.302%)
```

実践演習 9-1

Weka の Bagging では、学習する木の数はパラメータ numIterations で調整します。この値を 3 にして J48（minNumObj=20）で決定木を学習すると、以下のようなよく似通った決定木が得られます。

```
Tree 1
plas <= 111
|   preg <= 7: tested_negative (313.0/25.0)
...
plas > 111
|   mass <= 28.1: tested_negative (97.0/19.0)
...
```

```
Tree 2
plas <= 127
|   mass <= 26.4: tested_negative (128.0/3.0)
...
plas > 127
|   mass <= 29.9
...

Tree 3
plas <= 127
|   age <= 25: tested_negative (188.0/10.0)
...
plas > 127
|   mass <= 29.9: tested_negative (65.0/16.0)
...
```

実践演習 9-2

Weka の randomForest でも実践演習 9-1 と学習結果を比較しやすいように、numIterations=3、maxDepth=3 としておきます。この設定では、以下のように比較的異なった決定木が得られます。

```
Tree 1

plas < 111.5
|   preg < 7.5
...
plas >= 111.5
|   age < 24.5
...

Tree 2
plas < 127.5
|   mass < 26.45
...
plas >= 127.5
|   plas < 161.5

Tree 3
mass < 29.95
|   age < 28.5
...
mass >= 29.95
|   age < 24.5
...
```

実践演習 9-3

Weka の AdaboostM1 でも実践演習 9-1 と同様の設定とします。この設定では、以下のように比較的異なった決定木が得られます。

```
Tree 1
plas <= 127
|   mass <= 26.4: tested_negative (132.0/3.0)
...
```

```
plas > 127
|   mass <= 29.9: tested_negative (76.0/24.0)
...

Tree 2
age <= 24
|   plas <= 92: tested_negative (31.43)
...
age > 24
|   plas <= 154

Tree 3
plas <= 101
|   mass <= 27.5: tested_negative (39.88)
...
plas > 101
|   preg <= 7
...
```

実践演習 10-1

- 例題 10.1(階層的): linkType の値を SINGLE, COMPLETE, WARD などに変えて、結果を比較してみましょう。
- 例題 10.2(kMeans): 初期値を決める方法 (initializationMethod) を k-means++ に変えて、結果に変化がでるか確認しましょう。
- 例題 10.3(XMeans): minNumClusters の値を 3 にすると、kMeans とほぼ同じ結果が出ることを確認しましょう。
- 例題 10.5(EM): Weka の EM アルゴリズムを用いたクラスタリングの実装では、クラスタ数を自動で決めることができます。その手順をマニュアル (パラメータ調整画面から More ボタンを押す) で調べてください。

実践演習 10-2

例題 10.4 の通り。どのぐらいの異常値で LOF 値に顕著な違いが出るかを、1 つの次元のみが平均から標準偏差の 2 倍離れているデータ、すべての次元がその次元の最大値をとるデータなどで試してください。

実践演習 11-1

outputItemSets を True にすると頻出項目がリストアップされますが、apriori のデフォルトパラメータでは出力される項目が多すぎます。lowerBoundMinSupport を 0.3 程度にして、出力された項目を見ることで、supermarket データの概要をつかみましょう。

実践演習 11-2

lowerBoundMinSupport が 0.3 では、confidence が 0.9 以上になる規則が 1 つもありません。規則を出力するための confidence を示す minMetric を小さく（たとえば 0.5 に）してみます。そうするといくつか規則

が出力されます。

また、fruit を規則の結論部にするためには classIndex を fruit の次元である 83 にし、car を True にします。そうすると、fruit が含まれるという条件の下で、頻出項目が求められ、そこから規則が導かれます。

```
Apriori
=======

Minimum support: 0.35 (1619 instances)
Minimum metric <confidence>: 0.5
Number of cycles performed: 13

Generated sets of large itemsets:

Size of set of large itemsets L(1): 8

Large Itemsets L(1):
bread and cake=t 3330
0  2325
baking needs=t 2795
0  1900
juice-sat-cord-ms=t 2463
0  1672
biscuits=t 2605
0  1837
frozen foods=t 2717
0  1861
milk-cream=t 2939
0  2038
vegetables=t 2961
0  2207
total=low 2948
0  1719

Size of set of large itemsets L(2): 2

Large Itemsets L(2):
bread and cake=t milk-cream=t 2337
0  1684
bread and cake=t vegetables=t 2298
0  1791

Best rules found:

 1. bread and cake=t vegetables=t 2298 ==> fruit=t 1791    conf:(0.78)
 2. vegetables=t 2961 ==> fruit=t 2207    conf:(0.75)
 3. bread and cake=t milk-cream=t 2337 ==> fruit=t 1684    conf:(0.72)
 4. biscuits=t 2605 ==> fruit=t 1837    conf:(0.71)
 5. bread and cake=t 3330 ==> fruit=t 2325    conf:(0.7)
 6. milk-cream=t 2939 ==> fruit=t 2038    conf:(0.69)
 7. frozen foods=t 2717 ==> fruit=t 1861    conf:(0.68)
 8. baking needs=t 2795 ==> fruit=t 1900    conf:(0.68)
 9. juice-sat-cord-ms=t 2463 ==> fruit=t 1672    conf:(0.68)
10. total=low 2948 ==> fruit=t 1719    conf:(0.58)
```

実践演習 11-3

ここでは、Apriori と比較した FPGrowth の学習速度の向上を体験してください。

実践演習 11-4

たとえば、diabetes データに対して、Discritize フィルタをかけて、値を離散化して Apriori で規則を抽出してみてください。