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。