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

iris.arff データでは、minNumObj=1, unpruned = True にすることで精度 100% が実現できます。一方、iris.2D.arff データでは 1 事例の誤りをどうやっても消すことができません。これはデータ中に同じ特徴ベクトルに対して異なるクラスが正解として付与されているものがあるからです。

実践演習 3-2

minNumObj=2〜4, 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、グラフ構造を表示させている画面から 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（計算結果を同じにするためには、BayesNet 学習時に estimator (SimpleEstimator) のパラメータ alpha を 1 にする必要があります)。

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.80 図 4.10(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

親ノードの最大値が 1 のときは、ナイーブベイズ識別器と同じものが得られ、正解率は 72.0% となります。親ノードの最大値を 2 にすると、複雑なベイジアンネットワークが得られ、学習データに対する正解率は上がりますが、10-fold CV では 70.3% と下がってしまいます。

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

wine.csv ファイルから、学習に用いない 1981 年以降のデータを削除しておきます。また欠損値が 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。

[*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。

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

アッシェンフェルターのワイン方程式については、http://tenmei.cocolog-nifty.com/matcha/2014/07/post-746c.html のブログに面白いエントリーが掲載されています。

実践演習 6-2

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 6-3

カテゴリ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 6-4

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 が 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 6-5

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 7-1

手順は p.127 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

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 8-2

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 8-3

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。

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

ある程度までは中間層のユニット数の増加に伴って性能が向上しますが、それ以上は性能が不安定になります。

実践演習 10-1

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 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
...
plas > 111
|   mass <= 28.1
...

Tree 2
plas <= 127
|   mass <= 26.4
...
plas > 127
|   mass <= 29.9
...

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

実践演習 10-2

Weka の randomForest でも実践演習 10-1 と学習結果を比較しやすいように、numIterations=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
|   mass < 28.1
...

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
|   pedi < 0.53
...
```

実践演習 10-3

Weka の AdaboostM1 でも実践演習 10-1 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
...
plas > 127
|   mass <= 29.9
...

Tree 2
mass <= 26.3: tested_negative (114.59/21.09)
mass > 26.3
|   age <= 56
...

Tree 3
plas <= 123
|   plas <= 101
...
plas > 123
|   plas <= 165
...
```

実践演習 10-4

それぞれの手法で性能はほとんが変わりませんが、Bagging, RandomForest では複雑な木の多数決で、AdaBoostM1 では単純な木の重み付き和で同等の性能が実現できていることを確認してください。

実践演習 11-1

- 例題 11.1(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 11.2(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 11.4(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 11.7(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 11-2

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。