

実践編4：音声データを使った機械学習に挑戦しよう！

この演習では、機械学習の2つの手法を使って、動物の鳴き声の音声データを使った機械学習に挑戦しましょう。1つ目は、教師あり機械学習で動物の鳴き声を分類するプログラムを作ります。さらに、判別器の性能を確認する混同行列についても説明します。2つ目は、教師なし機械学習の手法を使って動物の鳴き声データをグループ分けします。機械は人間と同じように動物ごとにグループに分けられるでしょうか？

教師あり機械学習で鳴き声を聞き当てよう

教師あり機械学習を使って、猫、鳥、牛の3種類の動物の鳴き声を判別するプログラムを作りましょう。(*1)

教師あり機械学習の訓練データにするには、各データにそれぞれラベルをつける必要があります。この演習では、ファイルを読み込む際に、それぞれの動物の名前をラベルとしてつけるプログラムを使うことによって、簡単にラベル付きの訓練データを作っていきます。

1. 準備 訓練データを準備する

3種類の動物の鳴き声データファイルが、AudioDataFilesフォルダの中のcats、birds、cowsというフォルダにそれぞれ10個ずつ入っています。これらのファイルをWolfram言語で扱えるようにするため、Import関数を使って読み込みます。事前にこれらのファイルをAudioDataFilesフォルダごとWolfram Cloudにアップロードしておきましょう。(*2)

(*1) この演習は、音声データファイルをWolfram言語にインポートして行います。外部ファイルのインポートは、Wolfram Cloudの有料アカウントが必要です。詳しくは*ページを参照して下さい。

(*2) 演習のためのサンプルデータの入手方法は*ページを参照してください。この演習では、演習を行うWolframノートブックがあるフォルダの下にAudioDataFilesフォルダがあることを想定しています。ファイルのインポートについては本書サポートページを参照してください。

まず、以下を実行して、AudioDataFilesフォルダにアクセスできるようにします。

```
In[ ]:= SetDirectory[NotebookDirectory[] <> "AudioDataFiles"]
```

ディレクトリ… ノートブックのディレクトリ

以下は、ファイルを読み込む際に、それぞれの動物の名前をラベルとしてつけるプログラムです。

catsフォルダの中のファイルに「ねこ」、birdsフォルダのファイルには「とり」、cowsフォルダのファイルには「うし」というラベルをつけてImport関数でファイルを読み込み、それぞれの動物の訓練データをneko, tori, ushi に代入しておきます。Map関数を使うと、フォルダ内のファイル1つ1つがImport[#]の#に入って、一度に処理できます。

ここでは、最後に;(セミコロン)をつけて、読み込んだ音声情報を画面に表示しないようにしています。

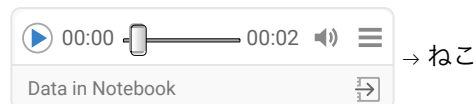
```
neko = Map[Import[#] → "ねこ" &, FileNames["cats/*"]];
tori = Map[Import[#] → "とり" &, FileNames["birds/*"]];
ushi = Map[Import[#] → "うし" &, FileNames["cows/*"]];
```

適用 インポート ファイル名

変数neko, tori, ushi に代入されているデータがどうなっているかを見えます。
猫の1つ目のデータを取り出してみましょう。

```
neko[[1]]
```

Out[]:=



上記のように、「ねこ」のラベルが付いていることが確認できます。プレイボタン▶を押すと、鳴き声を聴くことができます。tori, ushi も同様の方法で確認してみましょう。

各動物10個ずつのデータがあるので、そのうちの7個を訓練データ、残り3個をテストデータにします。ここでは、RandomSample関数を使って、それぞれランダムに訓練データとテストデータに分けています。

```
{trainneko, testneko} = TakeDrop[RandomSample[neko], 7];
{traintori, testtori} = TakeDrop[RandomSample[tori], 7];
{trainushi, testushi} = TakeDrop[RandomSample[ushi], 7];
```

リスト… 乱数のサンプル

猫、鳥、牛のそれぞれの訓練データ、テストデータをまとめて、traindata, testdata とします。

```
traindata = Join[trainneko, traintori, trainushi];
testdata = Join[testneko, testtori, testushi];
```

データの個数を確認してみましょう。Map関数を使って、Length関数をtraindataとtestdataに対して一度に適用し、要素の数（データの件数）の結果を出力しています。訓練データが21個、テストデータが9個あることが確認できます。

```
Map[Length[#] &, {traindata, testdata}]
```

Out[]=

```
{21, 9}
```

訓練データからランダムに1つを取り出して見てみましょう。「音声 -> ラベル」という形式になっていますね。

```
RandomSample[traindata, 1]
```

Out[]=

```
{ { 00:00 00:02 } → とり }
```

2. 学習 学習モデルを訓練する

1.で準備した訓練データ traindataをClassify関数に与えて実行すると訓練が始まります。（訓練には少し時間がかかるので、結果が表示されるまで待ってください）
これを変数nakigoeに代入しておきます。そうすることで、このnakigoeは、鳴き声を聞き分けるための分類関数になります。鳴き声を判別するという意味で判別器nakigoeと呼びましょう。

```
nakigoe = Classify[traindata]
```

Out[]=

```
ClassifierFunction[ { Input type: Audio
Classes: うし, とり, ねこ } ]
```

Information関数では、判別器nakigoeに関する詳しい情報を出力することができます。

Information[nakigoe]
情報

Classifier information	
Data type	Audio
Classes	うし, とり, ねこ
Accuracy	(71. ± 8.)%
Method	DecisionTree
Single evaluation time	752. ms/example
Batch evaluation speed	1.48 examples/s
Loss	0.749 ± 0.11
Model memory	20.9 MB
Training examples used	21 examples
Training time	35.4 s

- Data typeはデータタイプ. Audio (音声)
- Classesは分類対象. 「うし, とり, ねこ」
- Accuracyは正解率
- . . . など

図P4-1 ClassifierFunctionのInformationの出力例

考えてみよう：この判別器nakigoeの正解率は何%でしたか？また、このnakigoeは訓練データをいくつ使っているでしょうか？（ヒント：Information関数の出力結果Classifier informationの“Training examples used”に書かれています）

3. テスト 未知のデータを使って判定する

判別器nakigoeでテストデータの鳴き声を判定してみましょう。testdataのデータは9個あるので、ここでは2番目のデータを判定してみます。testdata[[2]]で、2番目のデータの中身を確認します。

testdata[[2]]

Out[*]=

▶ 00:00

00:01

🔊

☰

→ ねこ

Data in Notebook

📄

testdata[[2,1]] とすると、testdataの2番目のデータのうち、鳴き声のデータだけを取り出すことができます。testdata[[2,1]]を判別器nakigoeに与え、判定させてみましょう。

nakigoe[testdata[[2, 1]]]

Out[*]=

とり

これは正しく判定できなかったようですね。

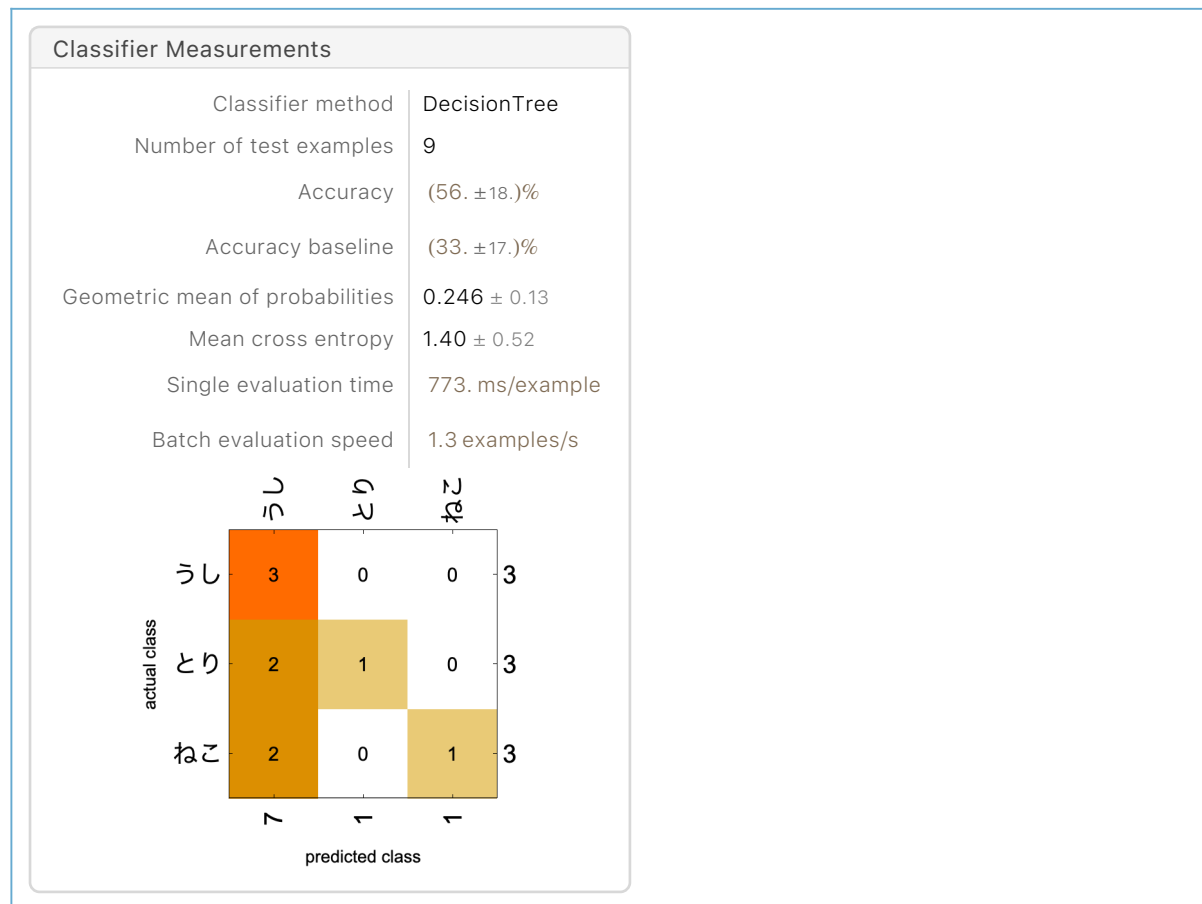
判別器の性能と混同行列

ClassifierMeasurements関数を使うと、テストデータを使った判別器の性能を確認することができます。

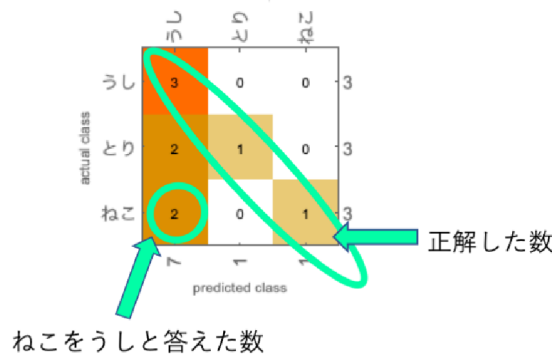
```
ClassifierMeasurements[nakigoe, testdata, "Report"]
```

分類測度

Out[]=



最後に表示されている格子図（図P4-2）は、縦軸がラベルの答え(actual class)、横軸が判別器が予測した答え(predicted class)で、判別器がどれくらい正解したか、何をどう間違ったのかを見ることができます。機械学習ではこの格子図を「混同行列」と言います。



図P4-2 判別器の性能をみる混同行列

例えば、図P4-2の対角線上の数字3,1,1は、ラベルの答えと判別器が予測した答えが一致した、いわゆる正解した数です。図P4-2では、縦軸のうしと横軸のうしが交わる格子に3という数字が入っているので、牛の鳴き声は3つとも正解しています。しかし、2行目のとり（正解）に対しては、1列目のうしに数字2が入っているので、とりの鳴き声のうち2つを、うしに間違えています。同様に、3行目のねこの行を見ると、1列目のうしに2が入っているので、2つはうしと間違えています。

考えてみよう：判別器nakigoeの性能について、Information関数やClassifierMeasurements関数の結果から考察してみましょう。例えば、格子図からどんなことが言えるでしょうか？どのような間違いが起こりやすいでしょうか？また、より良い判別器を作るにはどうしたら良いかを考えてみましょう。

教師なし学習で動物の鳴き声をグループ分けしてみよう

教師あり学習の演習と同じデータを使って、今度は動物の鳴き声をその特徴でグループ分けをしてみましょう。4章の4-5で学んだ教師なし機械学習です。

1. 準備 データを準備する

教師あり学習の1.で作成した訓練データ、テストデータをまとめて、変数animalsに代入します。


```
In[ ]:= animals = Join[traindata, testdata];
```


animalsから適当な1つを取り出してみると、以下のようになっていることがわかります。

```
RandomSample[animals, 1]
```

乱数のサンプル

Out[#]=

{  → うし }

Data in Notebook 

2. 学習 データに学習モデルを適用する

1. で準備した音声データを学習モデルに与えます。ここでは、4章の4-6で利用したFeatureSpacePlot関数を使って、音声データから抽出された特徴を数値化した値を2次元空間にプロットします。

特徴が似ているものは近くの位置に描かれます。同じ動物同士が近い位置に描かれているでしょうか？

In[]:=

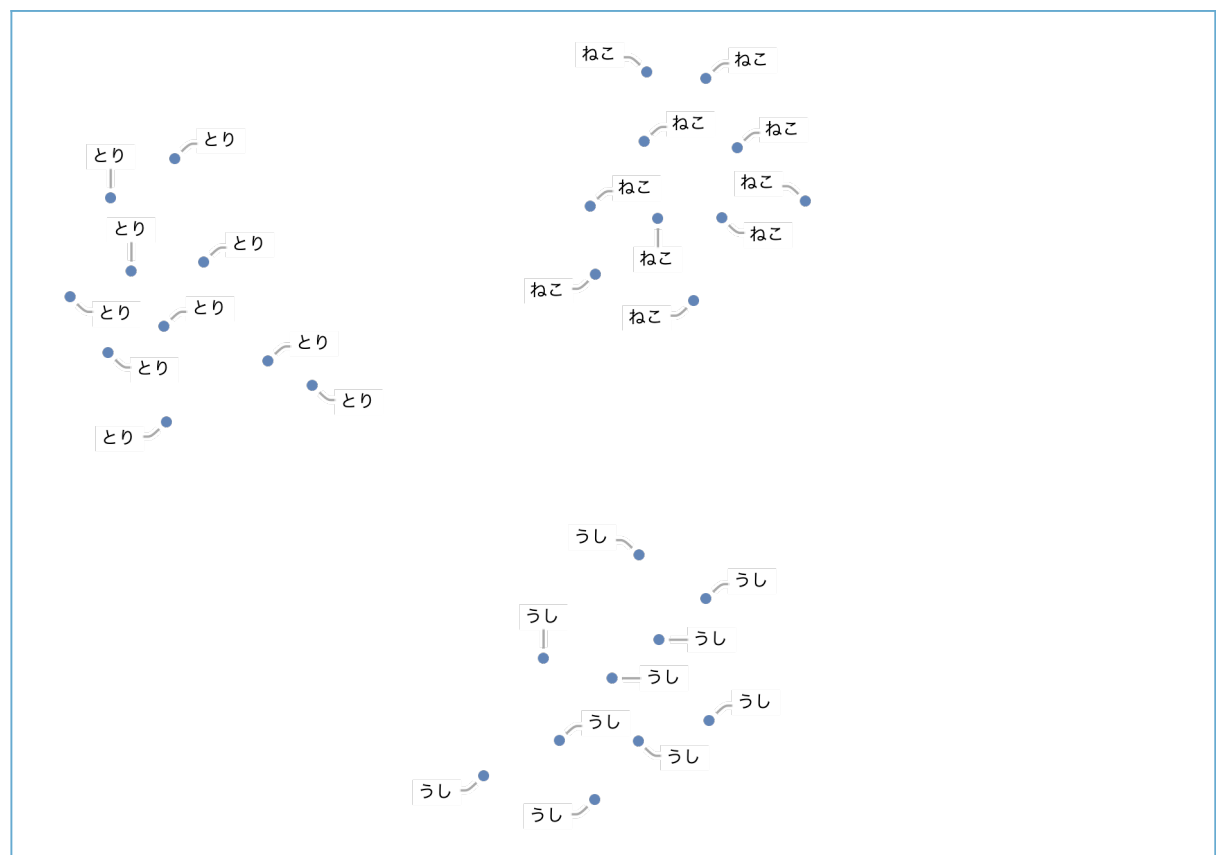
```
animals = 1;
```

```
FeatureSpacePlot[animals, LabelingFunction -> Callout]
```

[特徴空間プロット]

[ラベル付け関数]

[コールアウト]



💡ワンポイント：「教師なし機械学習」は答えを与えることができません。上記の「とり」「ねこ」「うし」の名前は、音声データを「教師なし機械学習」でグループ分けした後に、描画した後のラベルとして使われています。（名前の情報はモデルの学習には使用していないことに注意しましょう）

3. 活用 学習結果から新しい特徴を見つける

2. で出力されたグラフは私たちの考える動物の種類に分かれていて、このデータではグループ分けがうまくできたように見えます。でも、さらにグラフをよく見てみましょう。縦軸や

横軸に注目してデータを辿ると何か新しい発見があるかもしれません。

考えてみよう：教師なし学習では、どのように鳴き声をグループ化しているのでしょうか。ここまで学んだ音声データの正体や特徴を踏まえて、考えてみましょう。