scikit-learn

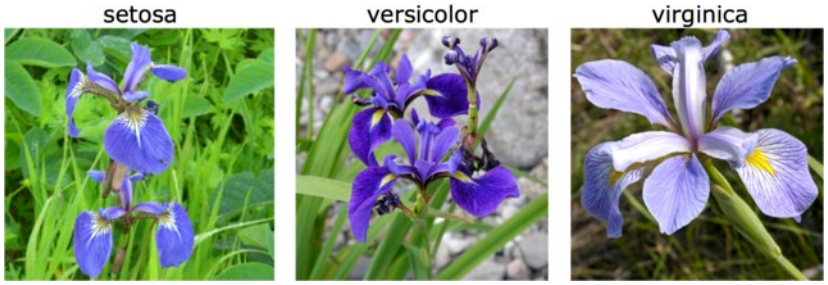
「AI と言えば Python」で、その AI＝機械学習のためのライブラリのひとつが scikit-learn です。なので、scikit-learn は Python の花形といえます。

scikit-learn は、numpy/scipy を使って動作します。「scikit」という用語も scipy toolkit の略称で、scipy の拡張モジュールとして開発がはじまった。

scikit-learn に関しても Anaconda のインストールで同時に読み込まれている。

■iris を使う

scikit-learn には「iris」（アヤメ）という学習用の花のデータセットが用意されています。「花がくの長さ」「花がくの幅」「花びらの長さ」「花びらの幅」の 4 つのデータが収められています。それぞれに、教師データとして「setosa：0」（ヒオウギアヤメ）、「versicolor：1」（ブルーフラッグ）、「virginica：2」（バージニカ）という 3 種類のラベルが付けられています。



「花がく」と「花びら」の長さと幅のデータから品種を予測するための学習用サンプルデータです。

◆iris データの読み込み

```
#1
from sklearn.datasets import load_iris

iris = load_iris()
print(iris.data.shape)
print(iris.data[:10])
print(iris.target_names)
```

iris データは、sklearn.datasets というモジュールに用意されている「load_Iris」関数で読み込みができる。

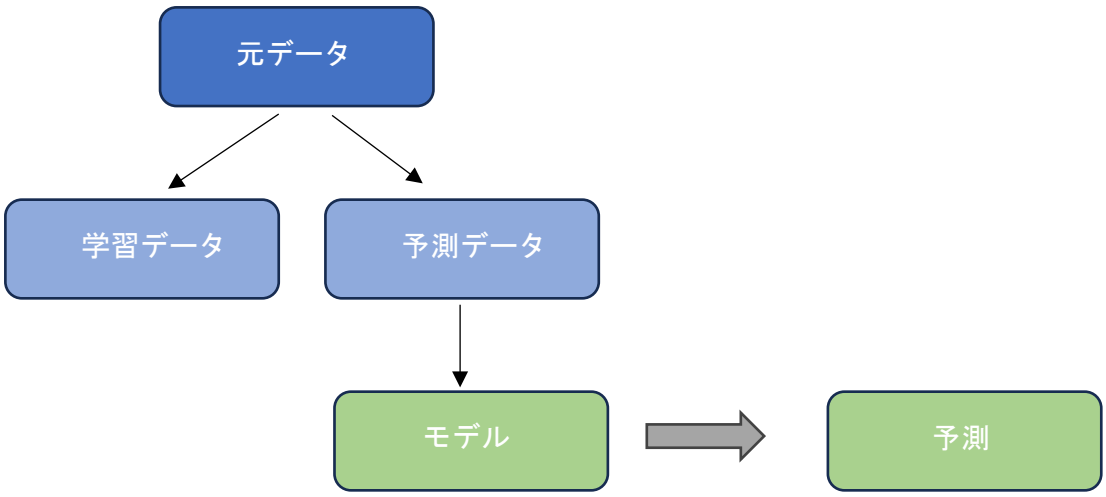
ここで表示される内容は、「データサイズ」（150×4 の行列）、「データの最初の 10 行」、「ターゲットの名前」。

```
(150, 4)
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]
 [5.4 3.9 1.7 0.4]
 [4.6 3.4 1.4 0.3]
 [5.  3.4 1.5 0.2]
 [4.4 2.9 1.4 0.2]
 [4.9 3.1 1.5 0.1]]
['setosa' 'versicolor' 'virginica']
```

◆2 種類のデータの準備

機械学習のためには、「**学習用データ**」と「**予測用データ**」の 2 つを事前に用意する必要があります。

まず、データを学習して「**モデル**」を作成します。そのモデルをもとにデータ予測を行う。



データの振り分けには、sklearn.model_selection **モジュールの** train_test_split 関数を使います。

文法：
変数 ＝ train_test_split（データ、教師データ）

```
#2
from sklearn.model_selection import train_test_split

(train_X, test_X, train_Y, test_Y) = train_test_split(iris.data,
                                        iris.target, test_size=0.2)

print(iris.target_names[test_Y])
print(test_Y)
print(test_X)
```

test_size＝0.2 は、データの 2 割を「予測用」に、残りを「学習用」に割り当てる指示です。

全部で 4 つの値を変数に格納される。

train_X　：学習用に割り当てるデータ
test_X　：予測用に割り当てるデータ
train_Y　：学習用データの教師データ
test_Y　：予測用データの教師データ

◆表示結果

・予測用の教師データの品種名

```
['versicolor' 'setosa' 'setosa' 'virginica' 'virginica' 'versicolor'
 'setosa' 'setosa' 'versicolor' 'versicolor' 'versicolor' 'setosa'
 'versicolor' 'versicolor' 'setosa' 'virginica' 'virginica' 'versicolor'
 'virginica' 'versicolor' 'setosa' 'versicolor' 'virginica' 'virginica'
 'virginica' 'versicolor' 'setosa' 'setosa' 'virginica' 'versicolor']
```

・予測用の教師データ

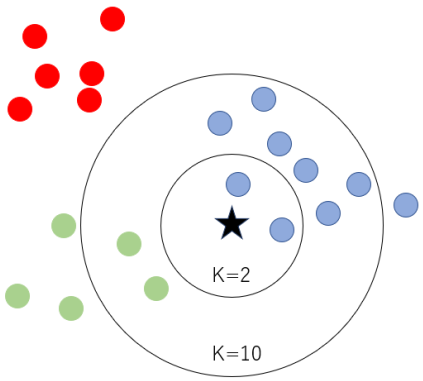
```
[1 0 0 2 2 1 0 0 1 1 1 0 1 1 0 2 2 1 2 1 0 1 2 2 2 1 0 0 2 1]
```

・予測用のデータ

```
[[5.6 2.9 3.6 1.3]
 [5.4 3.7 1.5 0.2]
 [5.2 3.5 1.5 0.2]
 [7.2 3.6 6.1 2.5]
 [6.5 3.  5.8 2.2]
 [5.7 2.6 3.5 1. ]
 [5.1 3.5 1.4 0.2]
 [5.2 3.4 1.4 0.2]
 [5.7 3.  4.2 1.2]
```

■予測を行う(1)

「K 近傍法」(k-nearest neighbor algorithm、k-nn、knn)を使った学習を行う。
k 近傍法とは、あるデータをグループ分けするとき、周囲にあるどのグループに近いかを多数決で推測する手法。



K 近傍法は sklearn.neibhbors モジュールの KNeighborsClassifier クラスとして用意されている。これを実行すら歌目には、インスタンスを作成して、学習用データを設定して学習させる。

> **文法**：
> [KNeigghborsClassifier].fit(**学習用データ、教師データ**)

ここでは、教師データは「train_Y」だが、このデータは単なる数値なので、これを品種名(target_names)にまとめたものをリストにして割り当てる。そうすれば、学習用データの答えが品種名で表示される。
fit の実行だけで、データの割り当てと、KNeighborsClassifier による学習が同時に完了する。

```
#3
from sklearn.neighbors import KNeighborsClassifier

model = KNeighborsClassifier()
model.fit(train_X, iris.target_names[train_Y])
```

これにより KNeighborsClassifier インスタンスが作成され、細かなパラメータが自動的に設定される。

結果表示は、

```
▼KNeighborsClassifier

KNeighborsClassifier ()
```

◆学習データを使った予測

学習モデル（model）を使って予測を行う。

#2 で用意した予測データ（test_X）をもとに予測を行い、その結果と解答のデータを表示する。

```
#4
from sklearn import metrics
from sklearn.metrics import
         accuracy_score,confusion_matrix,classification_report

pred = model.predict(test_X)
score = accuracy_score(iris.target_names[test_Y], pred)
print('score:%s' % score)
print(classification_report(iris.target_names[test_Y], pred))
print(confusion_matrix(iris.target_names[test_Y], pred))
```

※この実習には、データの前処理が必要なので、#1～＃3 までを事前に実行しておく必要がある

◆予測内容

precision	：正解予想の中で実際に正解だったもの割合
recall	：実際に正解だったものの中で「正解」を予測したものの割合
f1-score	：precision と recall の調和平均
support	：実際のサンプル数

表示結果(例)：

```
score:1.0
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00         9
  versicolor       1.00      1.00      1.00        12
   virginica       1.00      1.00      1.00         9

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30

[[ 9  0  0]
 [ 0 12  0]
 [ 0  0  9]]
```

◆予測内容(各値ごとに予測した数をまとめたもの)

上の結果では、setosa のデータは実際には[9 0 0]と予測された、と言うことを表わしている。
もし、この値が[0 1 9]であれば、正解が 9 で不正解が 1 を表わしている。

■表示内容について

◆精度(正解率)

score:1.0 なので、全正解となる。

機械学習による予測には、学習済みのモデルでは「predict」メソッドを使う。

```
pred = model.predict(test_X)
```

predict は、学習した内容をもとに、渡されたデータから結果（どこに分類されるのか）を返す。
これが機械学習による「予測」です。

具体的には、データに「花がくと花びらの長さ、幅」のデータ配列が用意されていた。それに対して、predict により各データの花の種類を予測し、その値を配列に返す。

◆精度計算

精度の計算に sklearn.metrics モジュールの accuracy_score 関数を使う。

```
変数 = accuracy_score（教師データ、予測したデータ）
```

※第 2 引数には、predict によって得られた予測データを指定

このサンプルでは、iris.taget_names で品種名に置き換えられたリストを使っているので、その値を指定している。

```
score = accuracy_score(iris.target_names[test_Y], pred)
```

この結果は、どれだけ精度（正解率）が高いかを示している。「1.0」に近いほど正解率は高くなる。

◆クラス分けレポート

sklearn.metrics モジュールの classification_report 関数で予測した結果のレポートを作成する。これを print（）で出力することで表示される。

precision、recall などの項目は、このレポートによって作成された。

```
変数 = classification_report（教師データ、予測データ）
```

◆予測結果の行列表示

予測した結果を直接知るには、sklearn.metrics モジュールの confusion_matrix を使う。

```
print(confusion_matrix(iris.target_names[test_Y], pred))
```

■ロジスティック回帰

ロジスティック解析は、次のコードになる。
既にデータは前準備も用意できているので、モデルを作成し、それを使って学習と予測をおこなわせればいい。

このソースコードを実行すると、iris データを使って学習、予測をおこなう。

※このコード実行のためには＃1 から＃4 の事前実行が必要

```
#5
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import
               accuracy_score,confusion_matrix,classification_report

model = LogisticRegression()
print(model)
model.fit(train_X, iris.target_names[train_Y])

pred = model.predict(test_X)

score = accuracy_score(iris.target_names[test_Y], pred)
print('score:%s' % score)
print(classification_report(iris.target_names[test_Y], pred))
print(confusion_matrix(iris.target_names[test_Y], pred))
```

◆LogisticRegression()

ロジスティック回帰は、sklearn.linear_model モジュールの LogisticRegression クラスとして用意してある。

```
model = LogisticRegression()
model.fit(train_X, iris.target_names[train_Y])
```

ここで作成した LogisticRegression に学習データを設定して、学習を実行させる。
fit の使い方は KNeighborsClassifier を同じ。そのほか、Predict による予測、レポートなど待ったくおなじなので、使用するクラスを変更するだけで、それ以外の変更はほとんどなく使うことができる。

◆出力(例)

```
LogisticRegression()
score:0.9666666666666667
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        13
  versicolor       0.86      1.00      0.92         6
   virginica       1.00      0.91      0.95        11

    accuracy                           0.97        30
   macro avg       0.95      0.97      0.96        30
weighted avg       0.97      0.97      0.97        30

[[13  0  0]
 [ 0  6  0]
 [ 0  1 10]]
```