教師あり学習（k近傍法）

Part 1

3日目 k近傍法

今日は機械学習のアルゴリズムを使って、初めての「機械学習」に挑戦します。利用するアルゴリズムは、「k近傍法」です。

k近傍法（k-Nearest Neighbor algorithm）は、機械学習において「教師あり学習」で、分類問題を解くためのアルゴリズムです。

分類問題とは、学習データを「クラス」と呼ばれるグループに分類しておき、テストデータがどのクラスに分類されるのかを予測する手法のことです。

なぜ、k近傍法を最初に試そうと思ったのか…。1つはアルゴリズムが単純で分かりやすかったから、もう1つは機械学習の入門書に必ず登場するアルゴリズムだったから…です。

では、k近傍法を使った学習モデルが、どのように予測を行うかを説明します。まず最初に、学習データをベクトル空間上に正解ラベルと共に配置します。教師あり学習なので、正解ラベルが必要です（**図1**）。

次に、与えられたデータを「特徴ベクトル」として、それぞれ近い場所にある正解ラベルのデータを使ってクラスに分類します（**図2**）。

図1●k近傍法の考え方（その1）

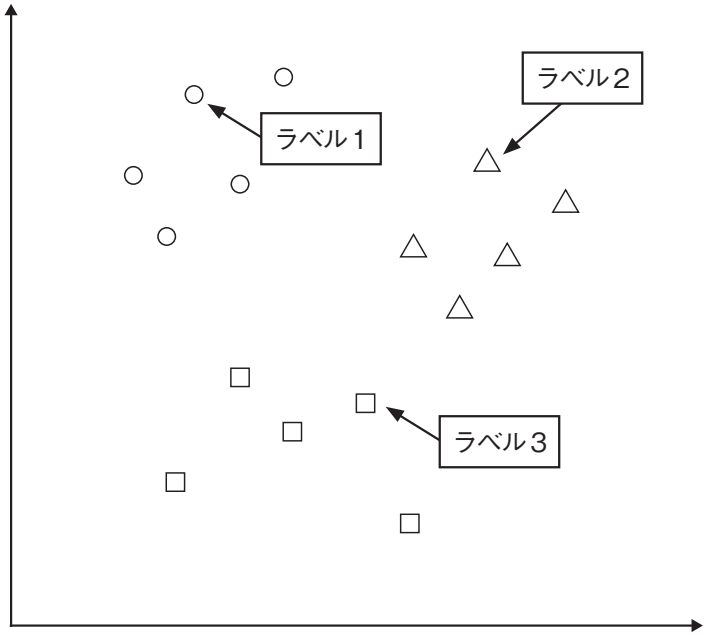
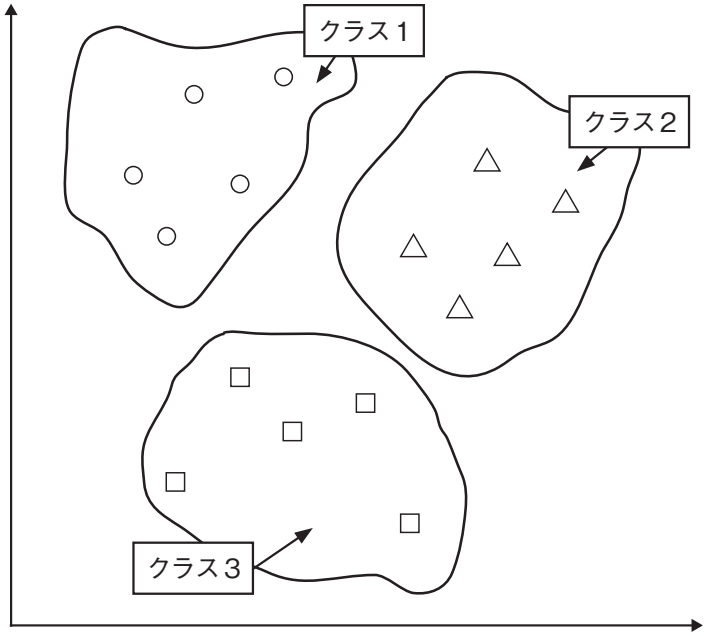
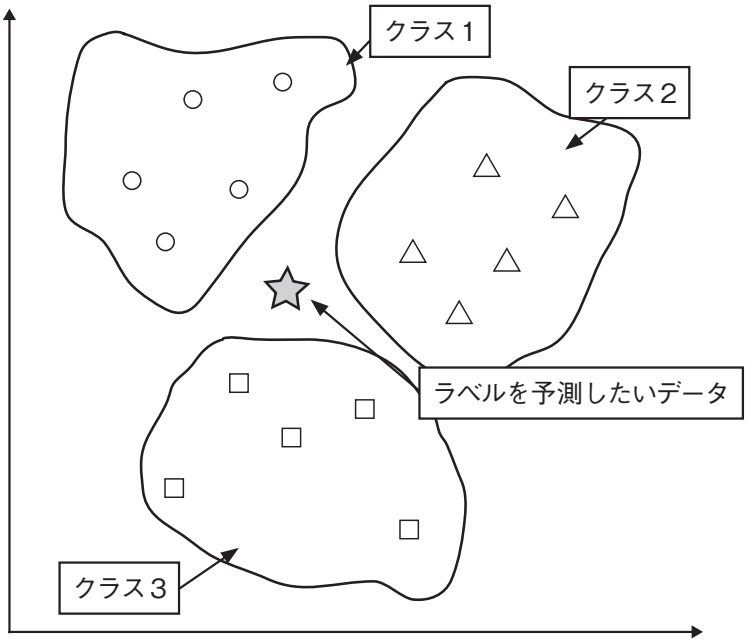


図2●k近傍法の考え方（その2）



これが、k近傍法の土台となる学習モデルになります。次に正解ラベルを知らせずに、予測したいデータを学習モデルに与えます（**図3**）。

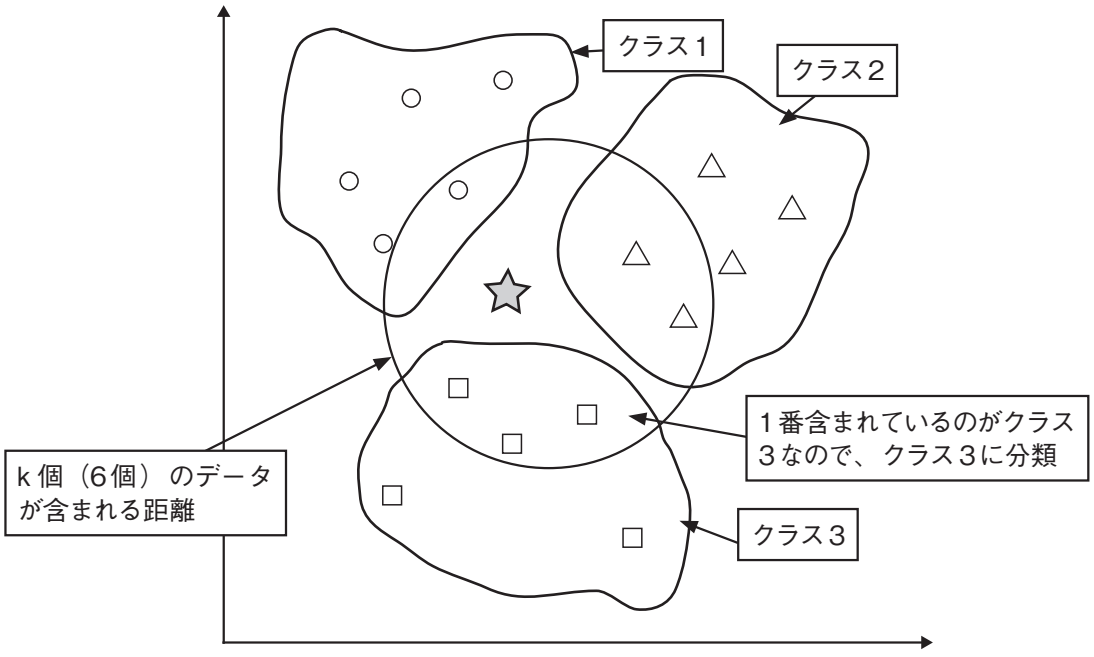
図3●k近傍法の考え方（その3）



学習モデルは、与えられたデータと存在するクラスとの近さ（距離）をもとに、そのデータがどのクラスに属するかを予測します。k近傍法では、「最も近いk個の点がどのクラスに一番多く存在するか」をもとに、属するクラスを決定します。

例えばkが6だったとすると、予測したいデータはクラス3に属することになります（**図4**）。

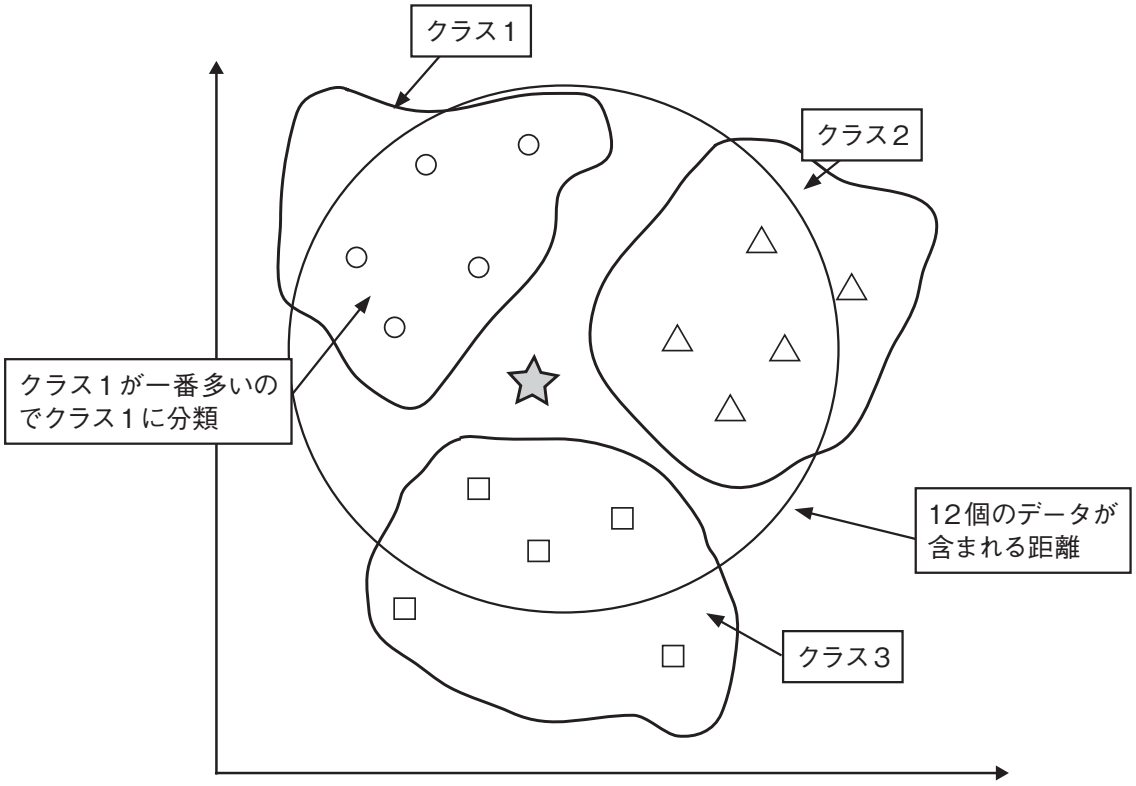
図4●k近傍法の考え方（その4）



ただし、この場合のkは任意なので、変更すると予測結果が変わる場合があります。例えば、kを12にした場合、予測したいデータは、クラス1に分類されます（**図5**）。

そこで、kの値は予測した結果の正解率から最適な個数を求めます。

図5●k近傍法の考え方（その5）



Part 2

3日目 データの分割

まず最初に、irisデータセットを訓練用のデータと評価用のデータに分割します。これは、生成した学習モデルの正解率を正しく判断するためです。訓練用のデータを評価用にも使ってしまうと、既に知っているデータなので、当然、正解率はとても高くなります。そこで、データセットを訓練用（training data）と評価用（test data）に分割してから使います。

ちなみに、学習データを訓練用と評価用に分割して、学習と評価を繰り返す方法を「クロスバリデーション」と呼びます。

データの分割には、scikit-learnの「train_test_split」という関数を使います。書式は**図1**のようになります。

図1●model_selection.train_test_split関数の書式。引数は抜粋

```
model_selection.train_test_split(*arrays, **options)
```

引数	説明
*arrays	トレーニング用の特徴行列、評価用の特徴行列、トレーニング用の目的変数、評価用の目的変数
test_size	テストデータのサイズ（1で100%）
random_state	乱数ジェネレータによって使用されるシード値
shuffle	データをシャッフルするか否か（デフォルトTrue）
返り値	分割されたリスト

train_test_split関数を使うと、データセットをランダムにシャッフルして、好きな割合で分割できます。

それでは、データの分割をやってみましょう。まず、irisデータセットを読み込みます。

```
In [1]: from sklearn.datasets import load_iris
        iris_dataset = load_iris()
```

次に、train_test_split関数でデータと正解ラベルを訓練用と評価用に分割します。分割後のリストは、次のように命名しました。scikit-learnでは、データを大文字、正解ラベルを小文字で表現するようなのでそれにならっています。

```
X_train   訓練用のデータ
X_test    評価用のデータ
x_train   訓練用の正解ラベル
x_test    評価用の正解ラベル
```

では、70%を訓練データ、30%を評価用データに分割してみます。

```
In [2]: from sklearn.model_selection import train_test_split
        X_train, X_test, y_train, y_test = train_test_split(ir
is_dataset['data'], iris_dataset['target'], test_size=0.3,  ra
ndom_state=0)
```

引数test_sizeを0.3にしているので、評価用のデータが3割、残りが訓練用のデータになります。正しく分割されているか確認してみましょう。

訓練用のデータであるX_trainをPandasのDataFrameで表示してみると、105個のデータが入っていることがわかります。データの順番はシャッフルされています。

```
In [3]: import pandas as pld
        pld.DataFrame(X_train, columns=iris_dataset.feature_na
mes)
Out[3]:
             sepal length (cm)  sepal width (cm)  petal length
(cm)  petal width (cm)
        0                  5.0               2.0
3.5               1.0
        1                  6.5               3.0
5.5               1.8
        2                  6.7               3.3
5.7               2.5
...
105 rows × 4 columns
```

評価用のデータX_testには、残りの45個が含まれています。

```
In [4]: pld.DataFrame(X_test, columns=iris_dataset.feature_nam
es)
Out[4]:
             sepal length (cm)  sepal width (cm)  petal length
 (cm)  petal width (cm)
        0                  5.8               2.8
5.1               2.4
        1                  6.0               2.2
```

```
4.0                1.0
        2                  5.5                4.2
1.4                0.2
...
       44                  5.4                3.7
1.5                0.2
```

正解ラベルも分割されているか確認します。

```
In [5]: y_train
Out[5]: array([1, 2, 2, 2, 2, 1, 2, 1, 1, 2, 2, 2, 2, 1, 2, 1,
0, 2, 1, 1, 1, 1, 2,
               0, 0, 2, 1, 0, 0, 1, 0, 2, 1, 0, 1, 2, 1, 0, 2,
2, 2, 2, 0, 0, 2, 2,
               0, 2, 0, 2, 2, 0, 0, 2, 0, 0, 0, 1, 2, 2, 0, 0,
0, 1, 1, 0, 0, 1, 0,
               2, 1, 2, 1, 0, 2, 0, 2, 0, 0, 2, 0, 2, 1, 1, 1,
2, 2, 1, 1, 0, 1, 2,
               2, 0, 1, 1, 1, 1, 0, 0, 0, 2, 1, 2, 0])

In [6]: y_test
Out[6]: array([2, 1, 0, 2, 0, 2, 0, 1, 1, 1, 2, 1, 1, 1, 1, 0,
1, 1, 0, 0, 2, 1, 0,
               0, 2, 0, 0, 1, 1, 0, 2, 1, 0, 2, 2, 1, 0, 1, 1,
1, 2, 0, 2, 0, 0])
```

ちゃんと分割されているようです。

Part 3 3日目 散布図をプロットしてみる

ここで、k近傍法のデータセットを散布図でプロットして、可視化してみましょう。irisデータセットの4つの特徴量（がく片の長さと幅、花弁の長さと幅）から、品種の分類ができそうか目視で確認します。

ここで、irisデータセットの散布図をプロットするコードを調べていると、「mglearn」というライブラリを使ってカラーマップを指定している例が見つかりました。

mglearnは、matplotlibのヘルパーライブラリのようですが、Anacondaには含まれていません。そこで、Anaconda Promptから「pip」を使ってインストールしておきます。

Anaconda Promptを起動して、「pip install mglearn」とコマンドを入力しましょう。

```
(C:¥ProgramData¥Anaconda3) C:¥Users¥User>pip install mglearn
Collecting mglearn
  Downloading mglearn-0.1.6.tar.gz (541kB)
    100% |████████████████████████████████| 542kB 6.5MB/s

...
Successfully built mglearn
Installing collected packages: mglearn
Successfully installed mglearn-0.1.6

(C:¥ProgramData¥Anaconda3) C:¥Users¥User>
```

インストールできたら、Jupyter Notebookに戻り、mglearnをインポートします。散布図のプロットには、matplotlib.pyplotを使うので一緒にインポートします。

```
In [10]:import mglearn
         import matplotlib.pyplot as plt
```

X_train（訓練用データ）からDataFrameを作成します。

```
In [11]:iris_dataframe = pld.DataFrame(X_train, columns=iris_d
```

```
ataset.feature_names)
```

それでは、データセットを散布図でプロットします。プロットデータの生成には、pandas.plottingのscatter_matrix関数を使います（**図1**）。

図1●scatter_matrix関数の書式。引数は抜粋

```
pandas.tools.plotting.scatter_matrix(frame, alpha=0.5, figsize=
None, ax=None, grid=False, diagonal='hist', marker='.', density
_kwds=None, hist_kwds=None, **kwds)
```

引数	説明
frame	pandas のデータフレーム
alpha	透明度。0（透明）～1（不透明）の間の数値
s	サイズ（デフォルト値: 20）
figsize	ウインドウサイズ（インチ）
marker	マーカーの形（デフォルト値: ‘o’= 円）
hist_kwds	柱グラフ（hist 関数）に渡されるプロットキーワード
cmap	カラーマップ
返り値	k 近傍法の学習モデルオブジェクト

scatter_matrix関数を使うと、一連の変数のペアプロットを作成して散布図にすることができます。なお、カラーマップの指定では、ラベルごとにmglearnの3色で着色しています。

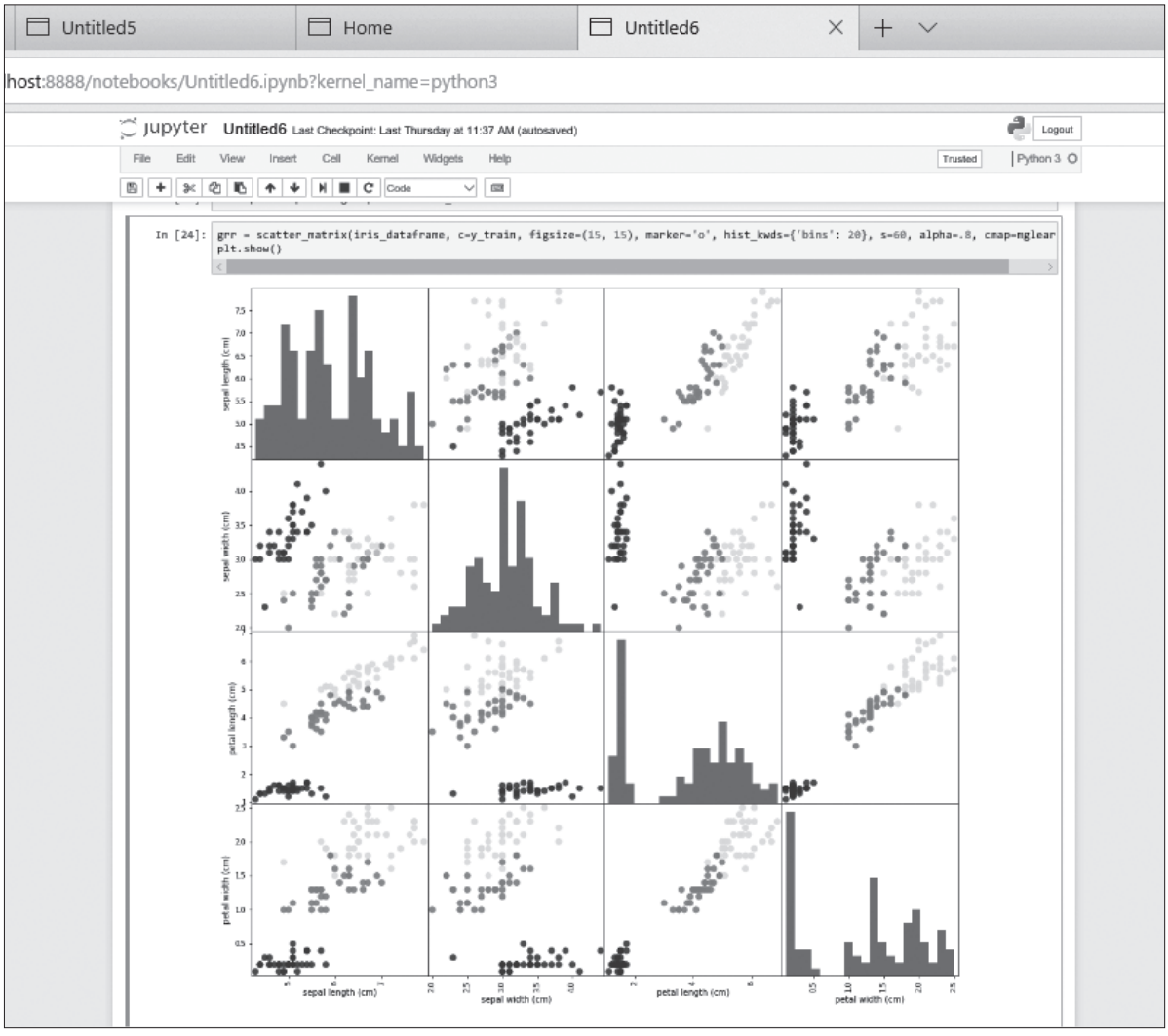
それでは、scatter_matrix 関数をインポートして散布図を表示してみましょう。

```
In [12]: from pandas.plotting import scatter_matrix
```

```
        grr = scatter_matrix(iris_dataframe, c=y_train, figsi
ze=(15, 15), marker='o', hist_kwds={'bins': 20}, s=60, alpha=.
8, cmap=mglearn.cm3)
        plt.show()
```

図2のように表示されます。3種類の品種（Setosa、Versicolour、Virginica）がラベルにしたがって色が付けられ、がく片と花弁の測定結果で分離しています。これなら、3つのクラスにグループ分けすることができそうです。

図2●散布図をプロットして可視化



3日目 Part 4 機械学習モデルの構築

準備が整ったので、k近傍法でクラス分類を行う学習モデルを構築しましょう。k近傍法のアルゴリズムは、scikit-learnのKNeighborsClassifierクラスに実装されています。書式は**図1**のようになります。

図1●KNeighborsClassifierの書式。引数は抜粋

```
sklearn.neighbors.KNeighborsClassifier(n_neighbors=5, weights='
uniform', algorithm='auto', leaf_size=30, p=2, metric='minkowski',
metric_params=None, n_jobs=1, **kwargs)
```

引数	説明
n_neighbors	k の値を指定（デフォルトは k=5）
weights	近傍までの距離を考慮するか否か（デフォルトは距離を考慮しない）
algorithm	最も近い近傍を計算するために使用されるアルゴリズム
n_jobs	並列ジョブの数。 －1の場合、利用可能な CPU の数に設定される
返り値	k 近傍法の学習モデルのオブジェクト

それでは、訓練データから学習モデルを構築しましょう。KNeighborsClassifierクラスをインポートします。

```
In [13]: from sklearn.neighbors import KNeighborsClassifier
```

KNeighborsClassifierオブジェクトを生成します。kの値は、1にしています。

```
In [14]: knn = KNeighborsClassifier(n_neighbors=1)
```

これで、学習モデルが生成されました。次に、訓練データをfit関数を使い読み込ませて学習させます。

```
In [15]: knn.fit(X_train, y_train)
Out[15]  : KNeighborsClassifier(algorithm='auto', leaf_size=30
, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=1, p=2,
           weights='uniform')
```

KNeighborsClassifierオブジェクトを生成したときのパラメータが表示されます。n_neighborsが1以外は、すべてデフォルトなのがわかります。これで学習は完了しました。

それでは、試しに学習データの中からデータを1つ与えて、正しく学習しているか確認してみましょう。

新しいデータを作ります。要素は、setosaのがく片の長さ、がく片の幅、花弁の長さ、花弁の幅です。

```
In [16]: import numpy as np
         X_new = np.array([[5.0, 2.9, 1.0, 0.2]])
```

では、クラスを予測させます。

```
In [16]: prediction1 = knn.predict(X_new)
```

予測の結果はprediction1に返ってきます。学習モデルが予想したラベルを表示してみましょう。

```
In [17]: print(iris_dataset['target_names'][prediction1])
         ['setosa']
```

setosaのデータを与えたので、正解です。

別のデータで確認しましょう。

```
In [18]: X_new = np.array([[6.0, 2.7, 5.1, 1.6]])
         prediction1 = knn.predict(X_new)
         print(iris_dataset['target_names'][prediction1])
         ['virginica']
```

こちらも正解でした。ただし、これは学習データからデータを抜き出して渡しているので、ほぼほぼ正解になるはずです。

正しく動作しているようなので、評価用データを使って正解率を出してみましょう。

▶モデルの評価

先ほど分割したテストデータを使って、作成した学習モデルがどのくらいの精度を持っているかを評価します。予測結果と評価用データの比較により、正解率を算出します。評価用データを学習モデルにセットします。

```
In [19]: y_pred = knn.predict(X_test)
```

評価用データに対して予測した品種ラベルは、次のようになりました。

```
In [20]: y_pred
Out[20]: array([2, 1, 0, 2, 0, 2, 0, 1, 1, 1, 2, 1, 1, 1, 1, 0
, 1, 1, 0, 0, 2, 1, 0,
                0, 2, 0, 0, 1, 1, 0, 2, 1, 0, 2, 2, 1, 0, 1, 1
, 1, 2, 0, 2, 0, 0])
```

では、正解ラベルはどうなっているか、もう一度確認してみると…。

```
In [21]: y_test
Out[21]: array([2, 1, 0, 2, 0, 2, 0, 1, 1, 1, 2, 1, 1, 1, 1, 0
, 1, 1, 0, 0, 2, 1, 0,
                0, 2, 0, 0, 1, 1, 0, 2, 1, 0, 2, 1, 1, 0, 1, 1
, 1, 2, 0, 2, 0, 0])
```

1カ所、間違えていますね。精度を計算してみます。

```
In [22]: np.mean(y_pred == y_test)
0.97777777777777775
```

つまり、この学習モデルでは、テストデータに対する精度は約97%の正解率ということです。

そこで、kの値を3（n_neighbors=3）にして学習モデルを作成し、学習データを読み込ませましたが、正解率は同じでした。そもそもk=1で97%の正解率なので、kを増やしても大きな違いはないということのようです。

なるほど、機械学習のデータとプログラムは、このように作るのか…。

少しわかって…、いやいや、まだまだですね。k近傍法はわかりやすいアルゴリズムですが、大量で複雑なデータには向かない気がします。明日は、別のアルゴリズムを試してみましょう。