ニューラルネットワークとクラスタリング

Part 1

5日目 ニューラルネットワーク

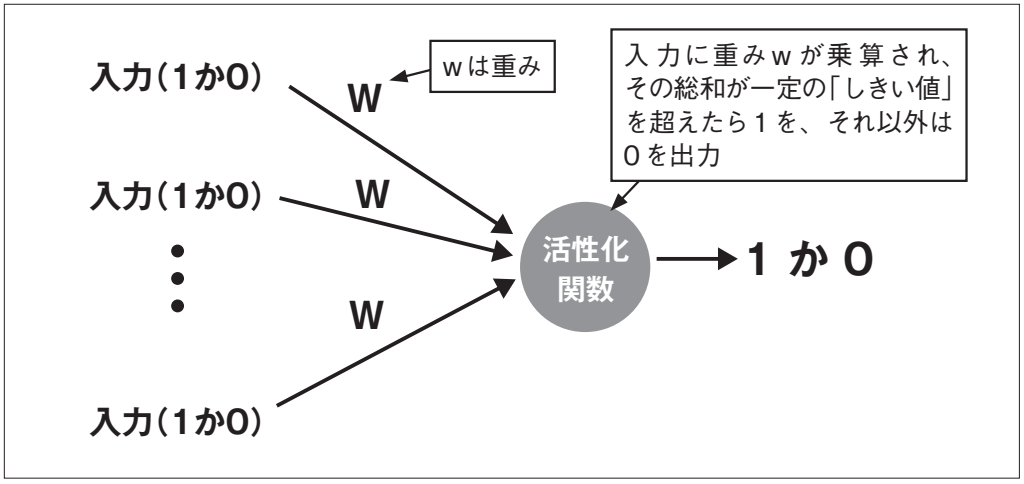
最終日は、ニューラルネットワークとk平均法を試します。まずは、ニューラルネットワークの説明からです。

▶形式ニューロン

ニューラルネットワークとは、人間の脳にある「ニューロン」という神経細胞を模したプログラムを、複数接続した機械学習システムのことです。

人間のニューロンは、外部から様々な電気刺激を受けて興奮し、やがて別のニューロンに電気信号を出力することで興奮が収束するという性質があります。このニューロンを模して作られたのが「形式ニューロン」(人工ニューロン)と呼ばれるプログラムです(**図1**)。

図1●形式ニューロン



形式ニューロンは、「入力(1か0)× 重み」の総和がしきい値を超えていると1を出力し、超えていなければ0を出力するという単純なプログラムです。この計算によって1を出力するか0を出力するかを決める関数を「活性化関数」と呼びます。

実は、昨日登場した「単純パーセプトロン」は、この形式ニューロンを機械学習に利用したものです。

▶単純パーセプトロン

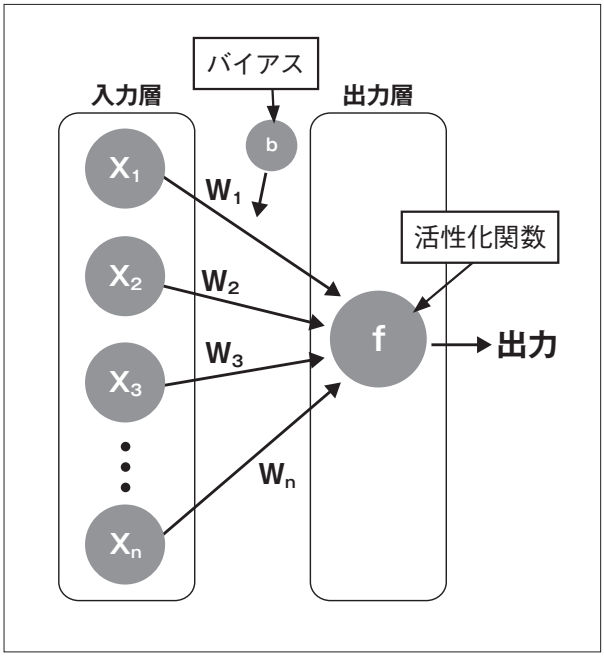
形式ニューロンは通常、入力が1か0のどちらかですが、入力を実数に対応させ、か

つ、学習により重みを更新できるように改良したものが「単純パーセプトロン」です。

単純パーセプトロンの構造は形式ニューロンと似ていますが、入力部分を「入力層」、出力部分を「出力層」と呼び、それぞれを構成するパーツを「ニューロン」と呼びます（**図2**）。

単純パーセプトロンでは、学習することで入力したデータを1か0に分類する「2クラス分類」（2値分類）が可能です。ただし、クラスを分離する境界に直線を用いるため、線形分離可能なデータしか判別できません。したがって、入力データによっては、1に分類されるか0に分類されるか判別できない場合があります。この単純パーセプトロンが持つ欠点を「XOR問題」といいます。

図2●単純パーセプトロン



例えば、データと正解ラベルに、ORの関係がある学習データがあったとします。この学習データから、**図3**のようにラベル1をクラス1、ラベル0をクラス2に分離する決定境界が引けます。

では、**図4**のような学習データではどうでしょうか。

ANDとNANDは、分離するクラスが逆になるだけで、決定境界の直線は同じです。

図3●XOR問題（その1）

入力(x1,x2)	正解ラベル
0,0	0
1,0	1
0,1	1
1,1	1

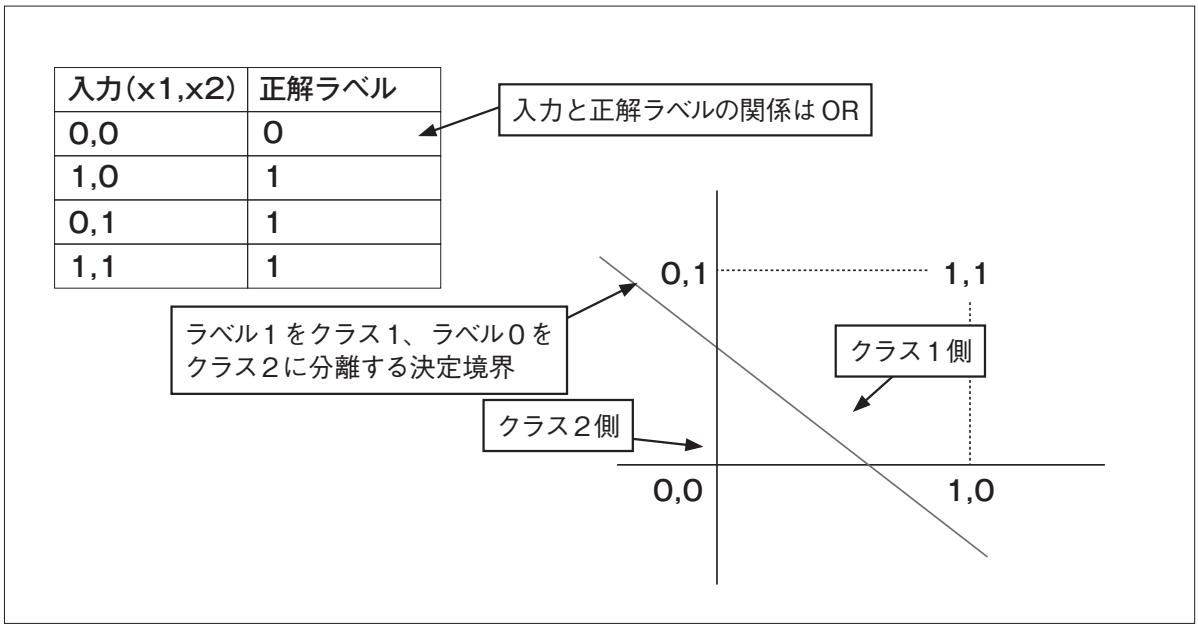
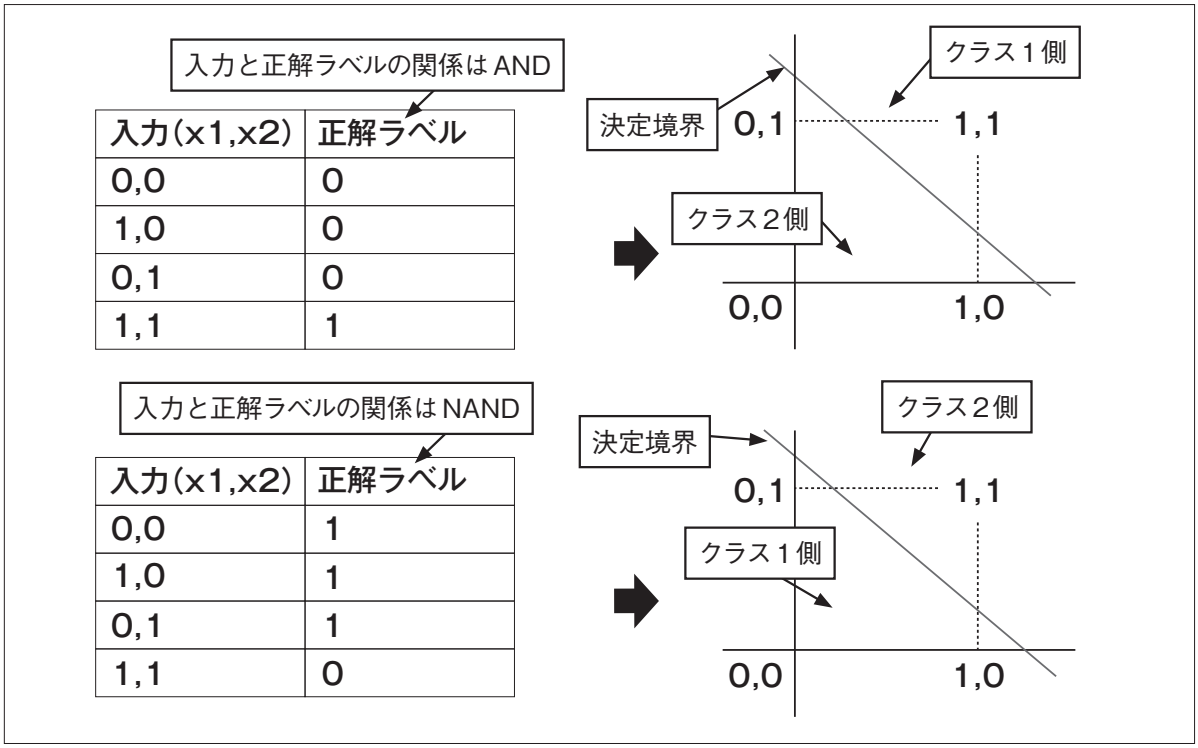


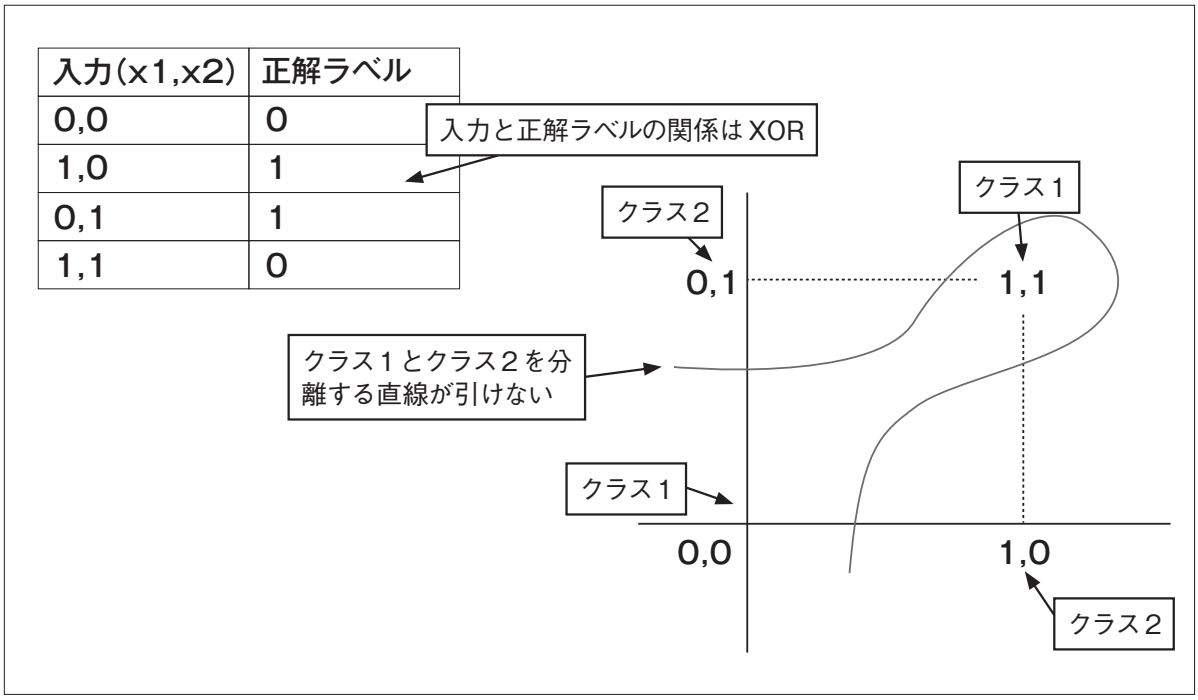
図4●XOR問題（その2）



OR、AND、NANDときたら、最後はXORですね。どうでしょうか（**図5**）。

このように、XORの学習データから直線の決定境界は引けません。これが単純パーセプトロンの「XOR問題」です。

図5●XOR問題（その3）



▶多層パーセプトロン

ところが、OR、AND、NANDの判別をする形式ニューロンを多層に組み合わせると、XORの識別ができることがわかっています（**図6**）。

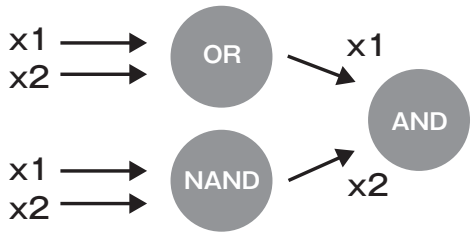
図6●XOR問題（その4）

(x1,x2)	OR
0,0	0
1,0	1
0,1	1
1,1	1

(x1,x2)	NAND
0,0	1
1,0	1
0,1	1
1,1	0

(x1,x2)	AND
0,0	0
1,0	0
0,1	0
1,1	1

入力(x1,x2)
0,0
1,0
0,1
1,1



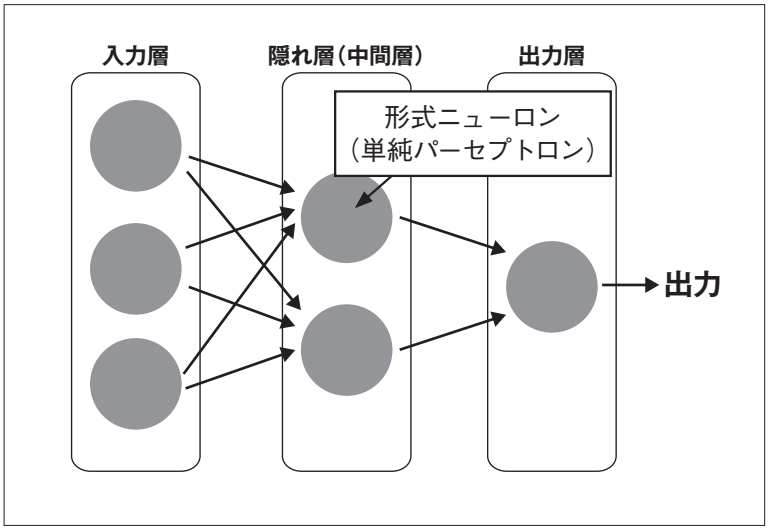
出力(XOR)
0
1
1
0

つまり、形式ニューロン（単純パーセプトロン）を何層も組み合わせることで、複雑な決定境界を引けるようになる、これが「多層パーセプトロン」です。

多層パーセプトロンでは、入力部分を「入力層」、出力部分を「出力層」、その間にあるパーセプトロンを「隠れ層」（中間層）と呼び、この隠れ層を2重3重と増やすことで複雑な決定境界を引くことができるようになります（**図7**）。

そして、このような多層パーセプトロンを利用した機械学習システムを「ニューラルネットワーク」と呼びます。

図7●多層パーセプトロン



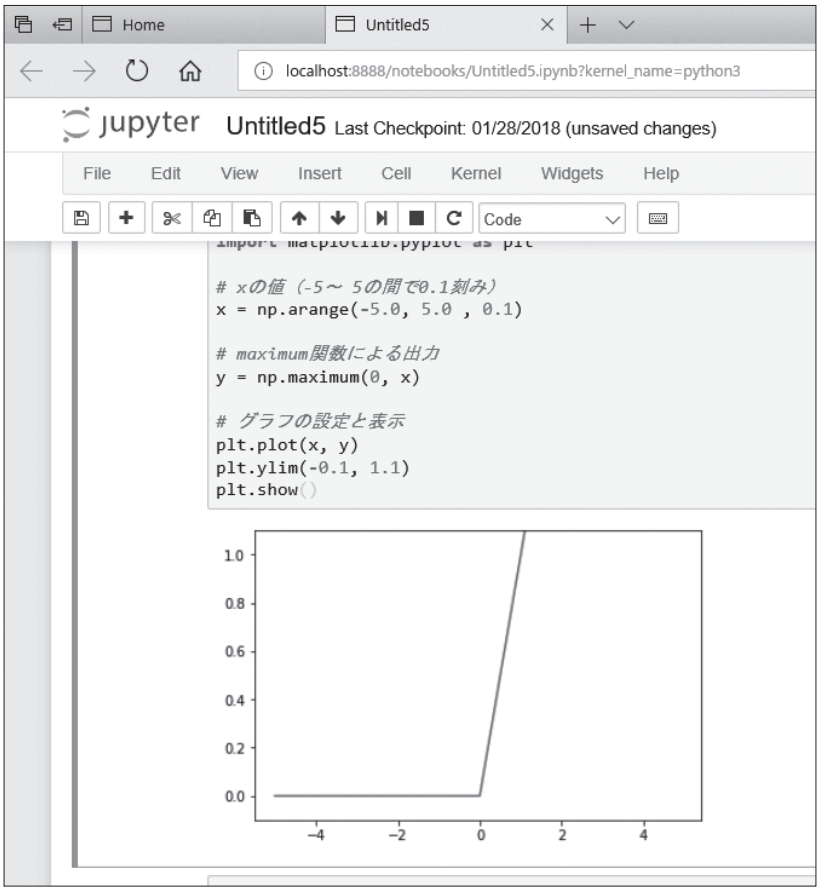
▶多層パーセプトロンの活性化関数

ニューラルネットワークの活性化関数には、ロジスティック回帰で登場した「シグモイ

ド関数」が利用されていましたが、最近は「ReLU（Rectified Linear Unit）関数」が用いらます。

ReLU関数は、入力が0を超えていれば入力をそのまま出力し、0以下なら0を出力する関数です（**図8**）。

図8●ReLU関数

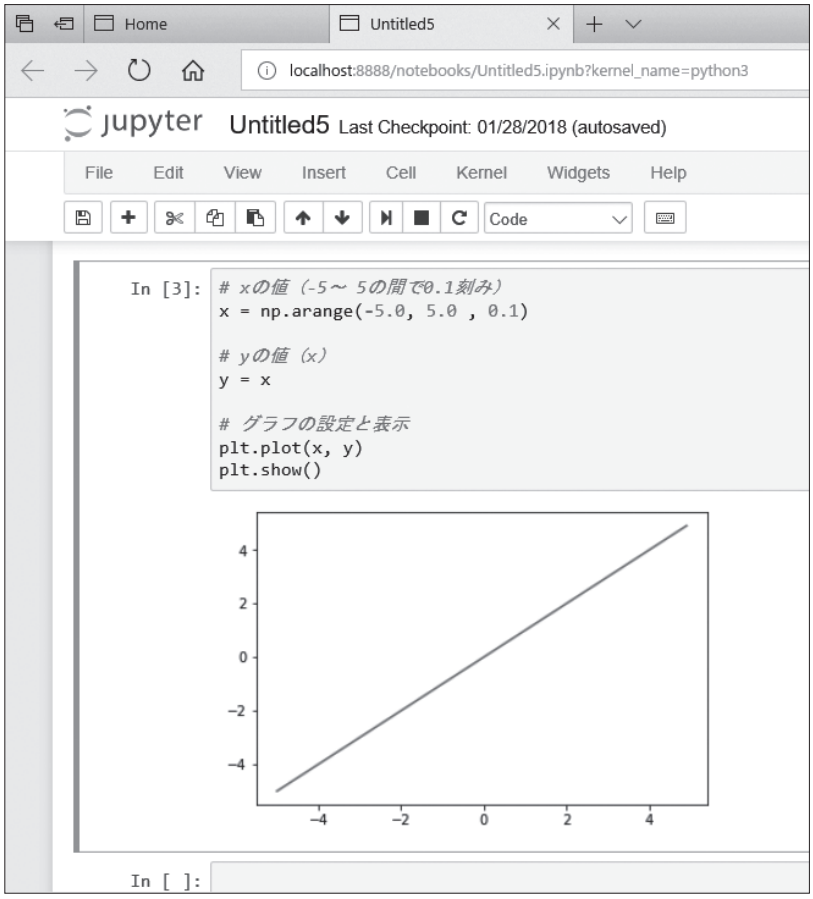



```
In [1]: import numpy as np
        import matplotlib.pyplot as plt
        %matplotlib inline

        # xの値（-5 ～ 5の間で0.1刻み）
        x = np.arange(-5.0, 5.0 , 0.1)

        # yの値（0とxの要素を比較した大きい方）
        y = np.maximum(0, x)

        # グラフの設定と表示
```

```
        plt.plot(x, y)
        plt.ylim(-0.1, 1.1)
        plt.show()
```

ちなみに、入力をそのまま出力する関数のことを「恒等関数」と呼びます（図9）。

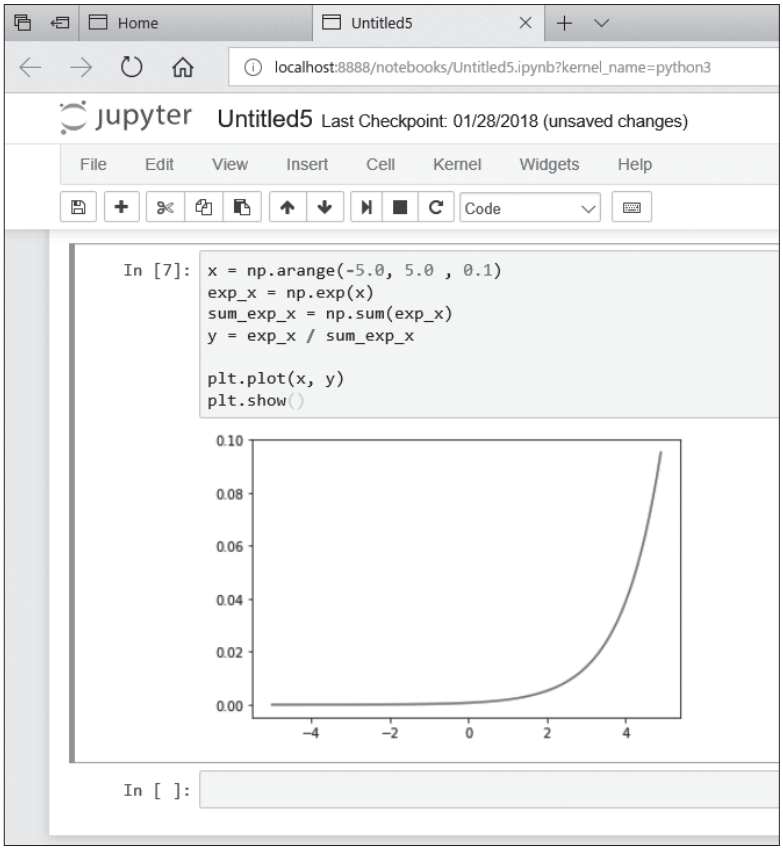
図9●恒等関数




```
In [2]: x = np.arange(-5.0, 5.0 , 0.1)
        y = x

        plt.plot(x, y)
        plt.ylim(-0.1, 1.1)
        plt.show()
```

ニューラルネットワークで分類を行う際、出力層の活性化関数に「ソフトマックス関数」（正規化指数関数）を採用することがあります。ソフトマックス関数は、出力がどのクラスに分類されるのかを、確率で出力するために使われます（図10）。

図10●ソフトマックス関数




```
In [3]: x = np.arange(-5.0, 5.0 , 0.1)
        exp_x = np.exp(x)
        sum_exp_x = np.sum(exp_x)
        y = exp_x / sum_exp_x

        plt.plot(x, y)
        plt.show()
```

▶誤差逆伝播

単純パーセプトロンでは更新する重みベクトルは1つですが、多層パーセプトロンの場合、どの順番でパーセプトロンが判断したのかによって、更新する重みが変わります。

ニューロンを逆にたどりながら重みを更新するアルゴリズムとしては「誤差逆伝播」（バックプロパゲーション、Backpropagation）が有名です。

とりあえず、ニューラルネットワークの考え方はわかりました。次は、scikit-learnで試してみましょう。

Part 2 5日目 MLPClassifierクラス

scikit-learnでは、バージョン0.18.0からニューラルネットワークを利用できるようになりました。そこで、scikit-learnのMLPClassifierクラスを使って機械学習を行ってみます。

MLPClassifierクラスは、多層パーセプトロン(MLP)方式で実装されていて、書式は**図1**のようになります。

図1●MLPClassifierの書式。引数は抜粋

```
MLPClassifier(hidden_layer_sizes=(100, ), activation='relu', sol
ver='adam', alpha=0.0001, batch_size='auto', learning_rate='cons
tant', learning_rate_init=0.001, power_t=0.5, max_iter=200, shuf
fle=True, random_state=None, tol=0.0001, verbose=False, warm_sta
rt=False, momentum=0.9, nesterovs_momentum=True, early_stopping=
False, validation_fraction=0.1, beta_1=0.9, beta_2=0.999, epsilo
n=1e-08)
```

引数	説明
hidden_layer_sizes	隠れ層の層の数とニューロンの数
activation	活性化関数 'identity'、'logistic'、'tanh'、'relu'
solver	最適化手法 'lbfgs'、'sgd'、'adam'
alpha	L2正則化のパラメータ
learning_rate_init	重みの学習率の初期値
learning_rate	重みの学習率の更新方法 'constant'、'invscaling'、'adaptive'
max_iter	試行回数の最大値
shuffle	学習を反復するごとに学習データをシャッフルするかどうか
random_state	乱数のシード値
warm_start	2回目のfit関数を呼ぶ際、学習済みの重みを引き継ぐか否か

▶ニューラルネットワークでアヤメを分類する

まずは、MLPClassifierを使って、irisデータセットの分類を行ってみましょう。irisデータセットを読み込み、訓練用のデータと評価用のデータに分けます。

```
In [4]: from sklearn.datasets import load_iris
        iris = load_iris()
        X = iris.data
        y = iris.target

        from sklearn.model_selection import train_test_split
        X_train, X_test, y_train, y_test = train_test_split(X,
y, test_size=0.3, random_state=0)
```

MLPClassifierで学習モデルを生成して、fit関数で学習します。パラメータは、すべてデフォルトを使用しています。

```
In [5]: from sklearn.neural_network import MLPClassifier
        mlpc = MLPClassifier()
        mlpc.fit(X_train, y_train)
        C:¥ProgramData¥Anaconda3¥lib¥site-packages¥sklearn¥neu
ral_network¥multilayer_perceptron.py:564: ConvergenceWarning:
Stochastic Optimizer: Maximum iterations (200) reached and the
optimization hasn't converged yet.
        % self.max_iter, ConvergenceWarning)

Out[5]: MLPClassifier(activation='relu', alpha=0.0001, batch_s
ize='auto', beta_1=0.9,
        beta_2=0.999, early_stopping=False, epsilon=1e-08,
        hidden_layer_sizes=(100,), learning_rate='constant',
        learning_rate_init=0.001, max_iter=200, momentum=0.9,
        nesterovs_momentum=True, power_t=0.5, random_state=Non
e,
        shuffle=True, solver='adam', tol=0.0001, validation_fr
action=0.1,
```

```
        verbose=False, warm_start=False)
```

何やらワーニング（警告）が出ました。評価用データを分類して正解率を表示してみます。

```
In [6]: pred = mlpc.predict(X_test)
        import numpy as np
        np.mean(pred == y_test)
Out[6]: 0.9555555555555556
```

少し正解率が低いようです。どうやらワーニングで表示されているパラメータ「max_iter」（試行回数の最大値）がデフォルトでは少ないのかもしれません。パラメータの値を増やしてもう一度学習させましょう。

```
In [7]: from sklearn.neural_network import MLPClassifier
        mlpc = MLPClassifier(max_iter=1000)
        mlpc.fit(X_train, y_train)
Out[7]: MLPClassifier(activation='relu', alpha=0.0001, batch_
size='auto', beta_1=0.9,
        beta_2=0.999, early_stopping=False, epsilon=1e-08,
        hidden_layer_sizes=(100,), learning_rate='constant',
        learning_rate_init=0.001, max_iter=1000, momentum=0.9,
        nesterovs_momentum=True, power_t=0.5, random_state=Non
e,
        shuffle=True, solver='adam', tol=0.0001, validation_fr
action=0.1,
        verbose=False, warm_start=False)

In [8]: pred = mlpc.predict(X_test)
        import numpy as np
        np.mean(pred == y_test)
Out[8]: 0.97777777777777775
```

正解率が上がりました。ニューラルネットワークは、デフォルトのパラメータ設定がシビアですね。

▶簡易手書き数字データの認識

digitsデータセットでも確認します。digitsデータセットを読み込み、訓練用と評価用に分けます。

```
In[9]: from sklearn.datasets import load_digits
       digits = load_digits()

       from sklearn.model_selection import train_test_split
       X_train, X_test, y_train, y_test = train_test_split(dig
its['data'], digits['target'], test_size=0.3,  random_state=0)
```

MLPClassifierの学習モデルを生成して学習を行います。パラメータのmax_iterは、1000にしておきます。

```
In [10]: from sklearn.neural_network import MLPClassifier
         mlpc = MLPClassifier(max_iter=1000)
         mlpc.fit(X_train, y_train)
Out[10]: MLPClassifier(activation='relu', alpha=0.0001, batch_s
ize='auto', beta_1=0.9,
         beta_2=0.999, early_stopping=False, epsilon=1e-08,
         hidden_layer_sizes=(100,), learning_rate='constant',
         learning_rate_init=0.001, max_iter=1000, momentum=0.9,
         nesterovs_momentum=True, power_t=0.5, random_state=Non
e,
         shuffle=True, solver='adam', tol=0.0001, validation_
fraction=0.1,
         verbose=False, warm_start=False)
```

これで学習が完了したので、精度を評価します。

```
In [11]: pred = mlpc.predict(X_test)
         import numpy as np
         np.mean(pred == y_test)
Out[11]: 0.97037037037037033
```

どうでしょう。他のアルゴリズムより、若干成績が良い気がします。どのように分類したの

か、数字ごとに見てみましょう。

```
In [12]: from sklearn.metrics import confusion_matrix
         confusion_matrix(y_test, pred, labels=digits['target_
names'])

Out[12]: array([[45,  0,  0,  0,  0,  0,  0,  0,  0,  0],
                [ 0, 51,  0,  0,  0,  0,  0,  0,  1,  0],
                [ 0,  0, 52,  0,  0,  0,  0,  1,  0,  0],
                [ 0,  0,  0, 53,  0,  0,  0,  0,  1,  0],
                [ 0,  0,  0,  0, 47,  0,  0,  1,  0,  0],
                [ 0,  0,  0,  0,  0, 53,  2,  0,  0,  2],
                [ 0,  1,  0,  0,  0,  0, 59,  0,  0,  0],
                [ 0,  0,  0,  0,  0,  0,  0, 53,  0,  0],
                [ 0,  2,  0,  0,  0,  1,  0,  0, 56,  2],
                [ 0,  0,  0,  0,  0,  1,  0,  1,  0, 55]], dty
pe=int64)
```

こちらもパラメータを色々と変更しながら、ミスを犯しやすいデータの特徴などを調べ、試行錯誤する必要があるでしょう。ただ、確実にわかったことは、scikit-learnを使えば、ニューラルネットワークをはじめとした様々な機械学習アルゴリズムを、ほとんど同じコードで簡単に試せるということです。だから、scikit-learnが人気なのでしょう。

Part 3 教師なし学習

ここまで、scikit-learnで試してきた機械学習の手法は、正解ラベルが学習データに付いている「教師あり学習」と呼ばれるものです。最後は、もう1つの機械学習である「教師なし学習」に挑戦します。

教師なし学習とは、回帰や分類による「予想」とは異なり、ラベルなし学習データから「特徴を分析」するための手法です。「クラスター分析」「主成分分析」といった分析手法があるようですが、今回は「クラスタリング」（クラスター分析）を調べます。

▶クラスタリングのアルゴリズム

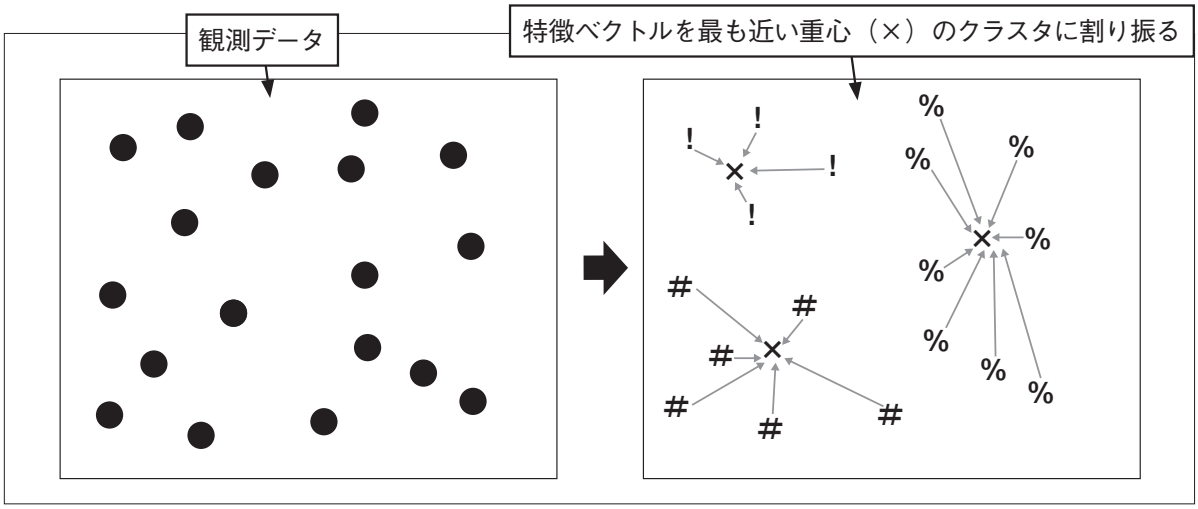
クラスタリングは、大きく分けて「階層的」アルゴリズムと「非階層的」アルゴリズムに分けられ、それぞれに複数のアルゴリズムが存在します。今回は、数あるクラスタリングアルゴリズムの中で、scikit-learnが実装していて、かつ、わかりやすかった(というか、私でも理解できた)アルゴリズムの「k平均法」(k-means clustering)を試します。

▶k平均法

k平均法は、非階層型クラスタリングです。非階層型クラスタリングは、あらかじめ観測データをいくつに分割するのか、その数(クラスタ数)を指定します。例えば、クラスタ数3でクラスタリングを行うと、似た特徴を持ったグループ(クラスタ)に3分割されます。

k平均法のアルゴリズムでは、特徴ベクトルを最も近い重心のクラスタに割り振ることで分割を行います(**図1**)。

図1●k平均法(その1)



k平均法では、次の手順で観測データを3つのクラスタに分離します。まず、各データに3つのランダムなクラスタを割り当てます。そして、クラスタごとに重心(座標の平均)を求めます(**図2**)。

求まった重心を仮のクラスタの重心として、一番近いデータを重心のクラスタに変更します。再び重心を計算します(**図3**)。

重心に変更があった場合は、仮のクラスタの重心として一番近いデータを重心のクラスタに変更し、再び重心を求めるという作業を繰り返します(**図4**)。

重心に変化がなくなるとクラスタリングが完了します(**図5**)。

図2●k平均法（その2）

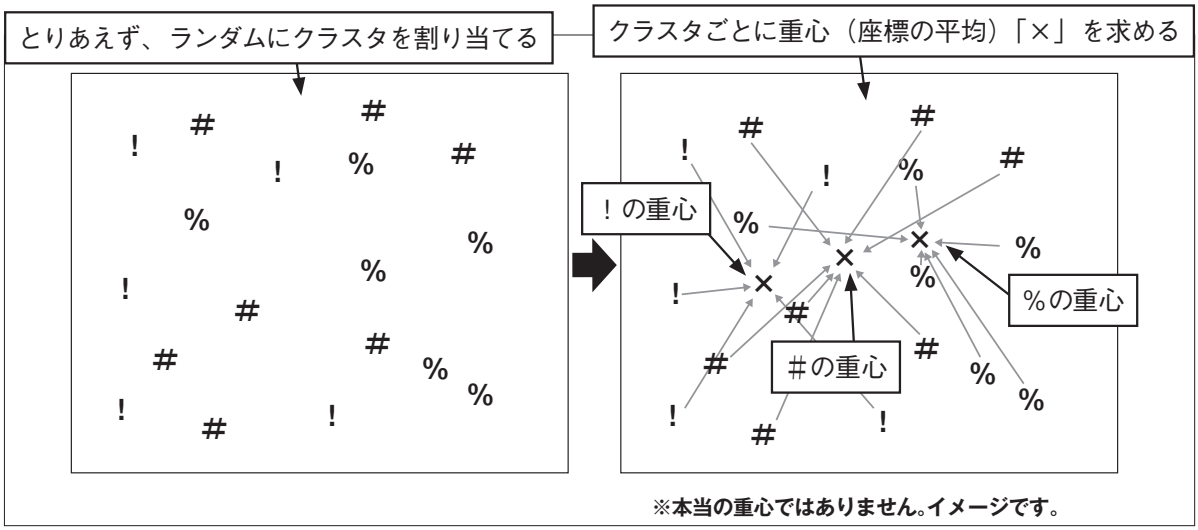


図3●k平均法（その3）

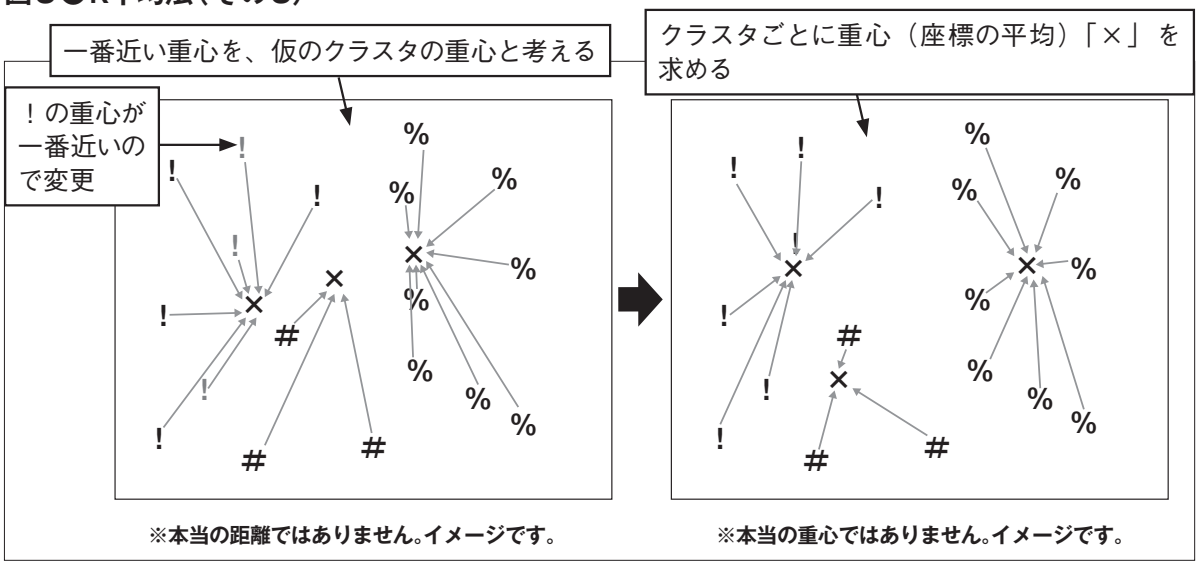


図4●k平均法（その4）

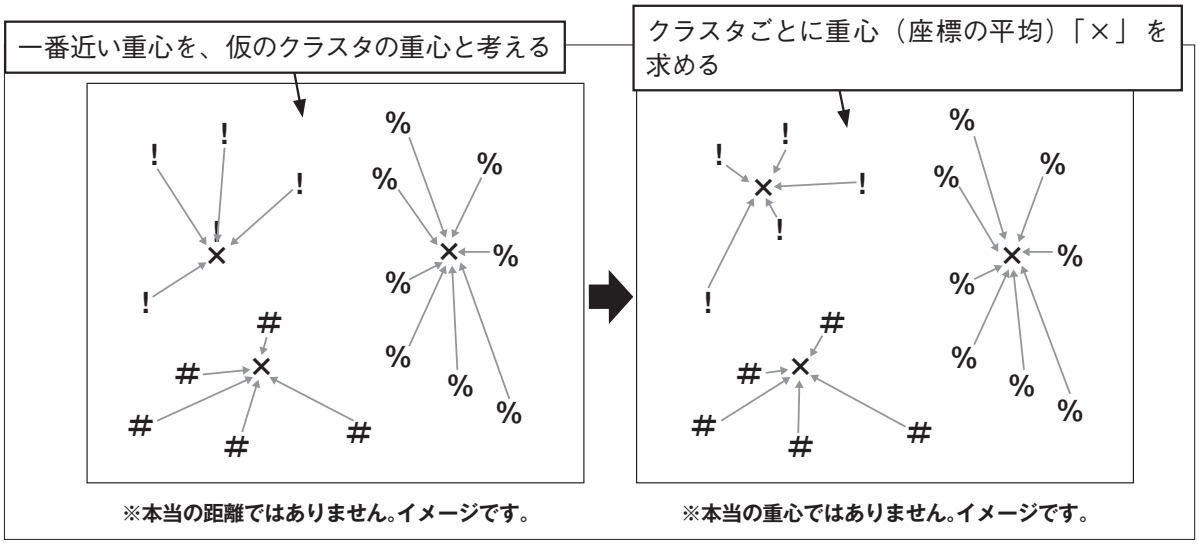
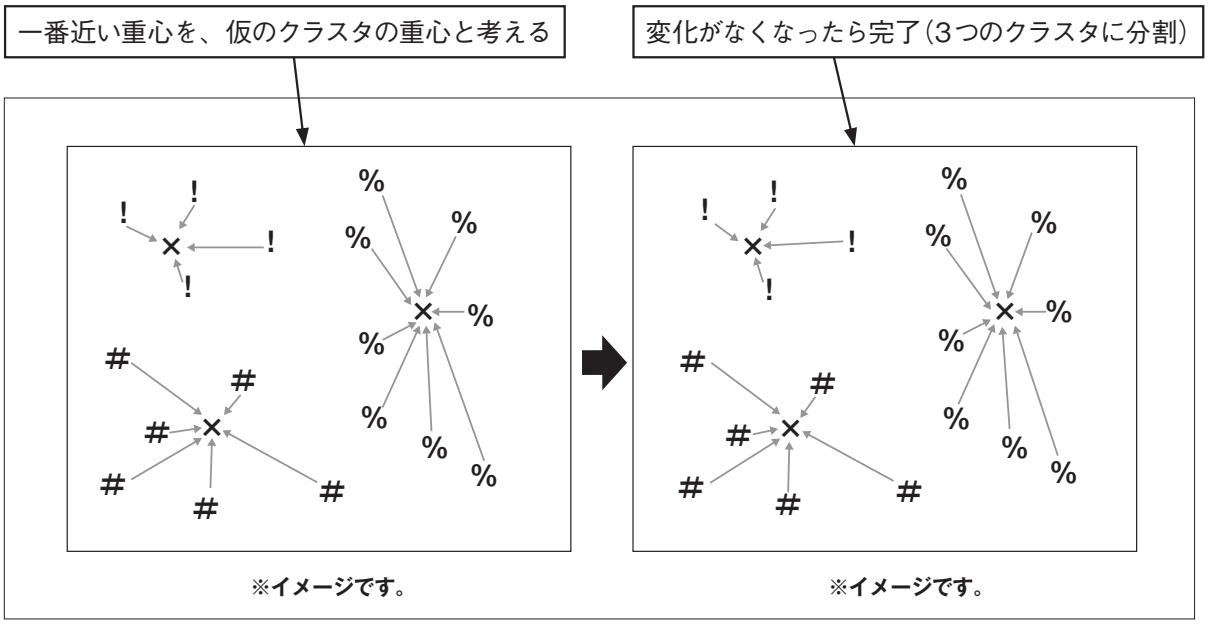


図5●k平均法（その5）



これが、k平均法のアルゴリズムです。直感的でわかりやすいアルゴリズムですが、初期値のランダムな割り振りによる揺れ幅が大きいので、何度も実行して結果の平均を取るのが良いそうです。

5日目 Part 4 k平均法を試す

それでは、k平均法をscikit-learnで試しましょう。scikit-learnには、cluster.KMeansクラスがあり、k平均法によるクラスタリングを実行できます。書式は、**図1**のようになります。

図1●KMeansの書式。引数は抜粋

```
KMeans(n_clusters=8, init='k-means++', n_init=10, max_iter=300,
tol=0.0001, precompute_distances='auto', verbose=0, random_stat
e=None, copy_x=True, n_jobs=1)
```

引数	説明
n_clusters	クラスタ数
max_iter	k-means アルゴリズムの最大反復回数

n_init	初期の重心を選ぶ処理の数
init	初期化方法 'k-means++' 'random' 'ndarray'
tol	収束判定に用いる許容可能誤差
precompute_distances	距離を事前に計算するか否か

▶クラスタリングの実施

今回は、irisデータセットを3つのクラスタに分類します。最初にirisデータセットをロードして、正解の分類を確認しておきます。

```
In [12]: from sklearn.datasets import load_iris
         iris_dataset = load_iris()
         iris_dataset['target']
Out[12]: array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2
, 2, 2, 2, 2, 2, 2, 2,
                2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2
, 2, 2, 2, 2, 2, 2, 2,
                2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

このように、irisデータセットには3種類のアヤメのデータ（ラベル0×50個、ラベル1×50個、ラベル2×50個）が入っています。それでは、KMeansの学習モデルにirisの計測データだけを渡して、うまく3つのクラスタに分離できるか試しましょう。

```
In [13]: # k-means学習モデルの生成（クラスタ数は3を指定）
         from sklearn.cluster import KMeans
         kme = KMeans(n_clusters=3)

         # クラスタリング開始
         kme.fit(iris['data'])
Out[13]: KMeans(algorithm='auto', copy_x=True, init='k-means++
', max_iter=300,
             n_clusters=3, n_init=10, n_jobs=1, precompute_dis
tances='auto',
             random_state=None, tol=0.0001, verbose=0)
```

これで、irisデータセットをk平均法によって、3つのクラスタに分けました。クラスタリング結果のラベルを表示してみます。

```
In [14]: kme.labels_
Out[14]: array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 2, 2, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2
, 2, 2, 2, 2, 2, 2, 2,
                2, 2, 2, 2, 2, 2, 2, 2, 0, 2, 2, 2, 2, 2, 2, 2
, 2, 2, 2, 2, 2, 2, 2,
                2, 2, 2, 2, 2, 2, 2, 2, 0, 2, 0, 0, 0, 0, 2, 0
, 0, 0, 0, 0, 0, 2, 2,
                0, 0, 0, 0, 2, 0, 2, 0, 2, 0, 0, 2, 2, 0, 0, 0
, 0, 0, 2, 0, 0, 0, 0,
                2, 0, 0, 0, 2, 0, 0, 0, 2, 0, 0, 2])
```

ラベル番号は、最初に割り当てたランダムな番号なので正解とは異なりますが、だいたい最初の50個がクラスタ1、次の50個がクラスタ2、最後の50個がクラスタ0に分割できそうです。

いかがだったでしょうか。もし、この結果を見て「そもそも、クラスタ数がわからない時はどうするのか?」とか、「クラスタ数もデータから分析できないのか?」といった疑問を抱くよう

でしたら、「クラスタリングとは何か?」の意味を、正しく理解していると思います。実際、クラスタ数を推定するアルゴリズムなどもあるので調べてみてください。

私の「人工知能プログラミング入門」は、ここまでとします。もし、致命的な間違いなどがあれば、ご指摘いただければ幸いです。

■筆者紹介

中島 省吾

有限会社メディアプラネット代表取締役。テクニカルライターとして、ネットワークやプログラミング関連の記事を執筆するほか、IT企業向けのセミナーや新人研修の講師なども手掛ける。最近は、ボランティアで子どもたちにもプログラミングを教えている。著書に「わかりすぎるC言語の教科書」（エスシーシー）などがある。

5日でわかる
人工知能プログラミング入門

著者●中島 省吾
編集●日経ソフトウエア
発行人●南浦 淳之
編集長●久保田 浩
デザイン・制作●JMCインターナショナル
表紙イラスト●ぷちめい

日経BP社