FEATURE 特集 1

今から学ぶ 機械学習アルゴリズム

AIの全体像からPythonによるプログラミングまでを一気に理解

仕事や学業のために、いよいよAIを勉強しなければならなくなった…という方にお届けするのが本特集です。AIの歴史と全体像から、基本的な機械学習アルゴリズムのPython実装まで、幅広く解説します。

澤田 千代子 プログラミングスクール「Future Coders ∞ Kids」(川崎市)主催。湘南工科大学講師(非常勤)。プログラミングや機械学習が専門。著書は「scikit-learn データ分析 実践ハンドブック」(秀和システム)や「そろそろ常識？ マンガでわかる『Python機械学習』」(C&R研究所)など。

▷Part 1

AIと機械学習の全体像を把握しよう

AIにおける機械学習の位置付けと機械学習の種類

現在、AI(Artificial Intelligence、人工知能)の"主流"と言える方式は「機械学習」と、機械学習の発展形である「深層学習」(ディープラーニング)です。これらは今やあらゆる分野で利用されています。まずは、AIの歴史を簡単にひもときながら、機械学習と深層学習が、どのように出現し、AIの主流になっていったのかを見ていきましょう(図1)。

● AI小史：現在は第3次AIブーム

AIという言葉は、1956年頃に米国の計算機科学者であるジョン・マッカーシー氏が生み出したと言われています。この頃、最初のAIブーム(第1次AIブーム)が起こり、AIは社会のいろいろな問題を解決できる夢の技術として期待されました。

第1次AIブームの特徴は、ドラえもんのような「汎用型AI」を目指したことです。しかし、実際にはチェスのようなルールがはっきりした問題には対応できても、現実世界に存在する複雑な問題には対応できなかったため、ブームは下火になっていきました。

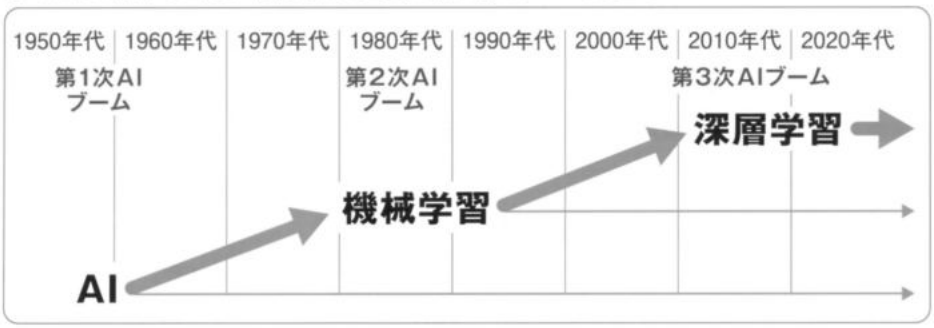
1980年代に入ると、人間がデータ分析を行うデータマイニングの技術が登場します。これに伴い、AIの中でも、「人の代わりに機械がデータを学習して、自動的に予測や分類を行う」=機械学習が注目されるようになりました。この機械学習の登場によって、第2次AIブームが起こります。以降、機械学習はAIの代名詞のようになり、機械学習のことをAIと呼ぶことが多くなりました。

また、第2次AIブームの頃のAIは、第1次AIブー

ムで志向されたような汎用型AIではなく、画像認識や音声認識といった特定の用途に対応するための「特化型AI」を目指しました。この流れは現在も続いています。

第2次AIブームの頃に、現在につながる様々な機械学習の技術が開発されましたが、大きな課題も残していました。具体的には次のようなものです。

図1●AIの簡単な歴史。機械学習の登場で第2次AIブームが起こる



- **そもそも機械学習に必要な大量のデータを集めることが難しい**
- **大量のデータの学習に時間がかかり過ぎる**

これらの課題の解決には、インターネットと高性能なコンピュータの出現を待たなければなりませんでした。

2010年代に入ると、インターネットとスマートフォンやソーシャルメディアの普及も相まって、比較的容易に収集できる大量のデータが日々生み出されるようになりました。また、比較的安価な割には高性能なGPU(Graphics Processing Unit)を機械学習の演算に使うことで、データの学習に必要な時間を大幅に短縮できるようになりました。第2次AIブームの頃の課題が解決されたわけです。

このような背景のもと、機械学習の世界に大きな技術革新が起こります。機械学習のアルゴリズムのひとつであるニューラルネットワークを発展させた深層学習の登場です。深層学習によって、従来の機械学習よりもはるかに高精度な予測や分類が可能になりました。深層学習は第3次AIブームを起こし、現在もそのブームが続いていると言えます。

●統計的ではない手法のAI

現在主流になっている機械学習は、より厳密に言うと、統計学を応用した「統計的機械学習」です。この特集も統計的機械学習の解説がメインテーマですが、「統計的ではない機械学習」と、「機械学習ではなく統計的でもないAI」のことも知っておきましょう。

統計的ではない機械学習の代表には、「強化学習」があります。強化学習は、試行錯誤を通して評価(報酬)を得られる行動や選択を学習する手法です。脳の学習メカニズムをモデルにしています。自動運転や対戦型のゲーム、二足歩行ロボットなどのAIで応用されています。

強化学習は統計的機械学習と全く無関係というわけではありません。深層学習と強化学習を組み合わせた「深層強化学習」は非常に強力な手法として知られています。例えば、プロ棋士を破った囲碁AIとして有名な「AlphaGo」(アルファ碁)は深層強化学習の手法を使っています。

機械学習ではなく統計的でもないAIとしては、「ルールベースAI」や「遺伝的アルゴリズムによるAI」などがあります。

ルールベースAIは、事前に人間が定義したルールをもとに各種の処理を行うAIです。ECサイトにおいて、女性には美容用品の広告を表示し、男性にはスポーツ用品の広告を表示する、といった処理は最も簡単なルールベースAIと言えます。

ルールベースAIは定義されたルール以上のことができないので性能的に限界がありますが、統計的機械学習などに比べると実装が簡単なのでしばしば使われます。

遺伝的アルゴリズムは、生物の進化をシミュレートするように試行錯誤を繰り返して、最適解を探索する手法です。「巡回セールスマン問題」(すべての都市を巡回する最短経路を求める問題)のように、全パターンの探索が不可能に近い問題に対して有効です。

ここまで解説したAIの手法をまとめると図2のようになります。

機械学習を手法で分類
教師あり学習と教師なし学習

ここからは統計的機械学習(以降、機械学習)をいろいろな視点で分類しながら解説します。

機械学習は、手法で分類すると、「学習」するために用いる各データに「正解」が付いている「教師あり学習」と、付いていない「教師なし学習」に大別できます。

例として、ウサギの画像かネコの画像かを分類できる機械学習を考えましょう。

●教師あり学習

教師あり学習では、ウサギとわかっている画像とネコとわかっている画像を大量に用意し、機械学習させることで、ウサギかネコかを分類できる「モデル」を作ります。機械学習のモデルとは、大雑把に言えば、数式とパラメータの集合です。

学習の中身はこの場合、画像に写る動物の特徴(形状や色など)を捉えることです。機械学習がうまくいけば、モデルはウサギかネコかを分類できる機能を持ち、教師データが付いていない未知の画像(ウサギが写っているのかネコが写っているのかわかっていない画像)に対しても、ウサギかネコか、正しく分類できるようになります(図3)。

教師あり学習では、学習用のデータ(ウサギかネコの画像)に、「教師データ」と呼ばれる正解情報(その画像がウサギかネコかを示すラベル)を、あらかじめ紐づけておきます。その作業は、基本的には人間が行うことになるので、手間がかかります。

●教師なし学習

教師なし学習は、教師データのないウサギの画像とネコの画像を大量に学習させてグループ分けのモデルを作り、ウサギの画像とネコの画像に自動的に分けるような機械学習です(図4)。ただ

図2●AIの分類。AIは機械学習とそれ以外に分類できる。また、機械学習は統計的機械学習と強化学習に分類できる

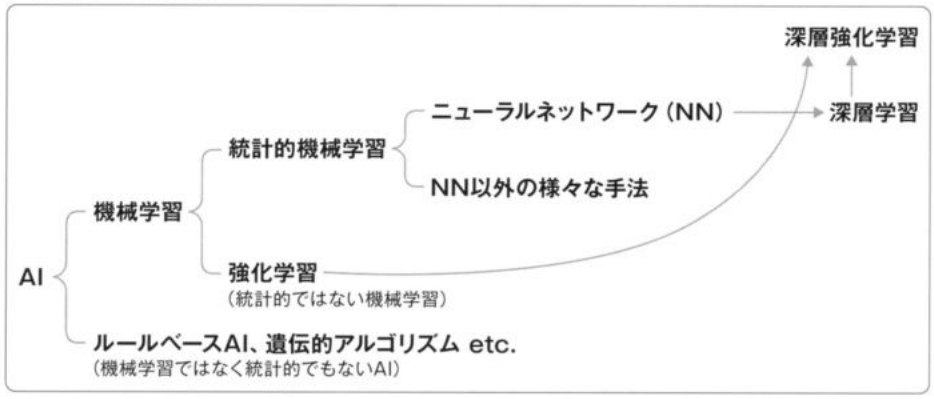
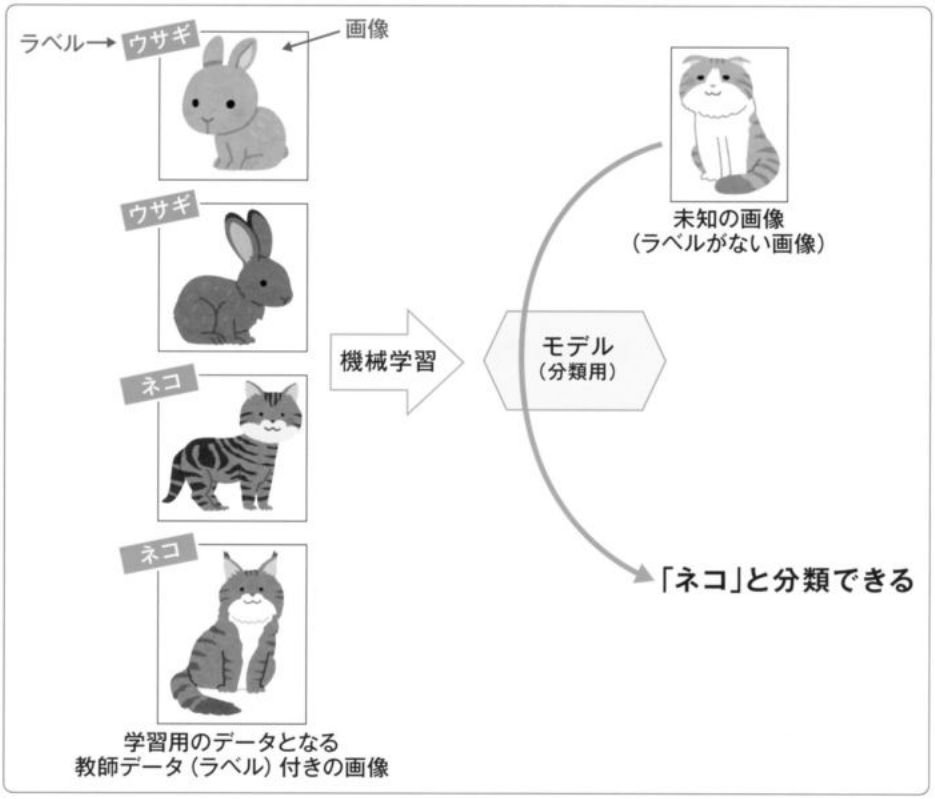


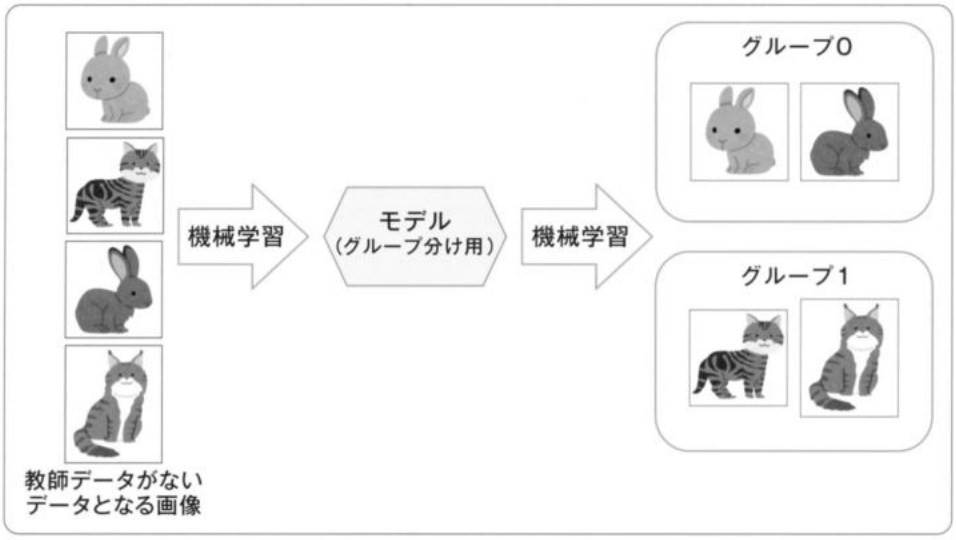
図3●教師あり学習のイメージ



し、教師なし学習のモデルは、ウサギやネコの画像が、ウサギやネコであることを認識しているわけではありません。画像の特徴だけを見て、単に、グループ0やグループ1のように、複数のグループに分けるだけです。

なお、教師なし学習が単体で使われるケースは少ないと言えます。教師なし学習は、教師あり学習の前段階で使われることが多くなっています。

図4●教師なし学習のイメージ



機械学習を用途で分類
予測、分類、グループ分け、情報の要約

今度は、機械学習を用途で分類してみましょう。用途の観点からは、「予測」「分類」「グループ分け」「情報の要約」の4種類に大別できます。

● 予測

予測とは、既存のデータから未来のデータを予測することです。例えば、過去の「気温」と「アイスクリームの売り上げ」のデータから予測モデルを作り、今年の予想気温からアイスクリームの売り上げを予測します。予測では、教師あり学習の「回帰」と呼ばれる手法がしばしば使われます。

図5は、回帰の中でも最も基本的な「線形回帰」の例です。散在する丸が、ある気温のときのアイスクリームの売り上げデータで、直線がデータを機械学習(ここでは最小二乗法)することで作った予測のモデルです。この予測のモデルは次の1次式になります。

```
売り上げ = 0.307 × 気温 + 4.397
```

この式の気温に値を入れれば、売り上げを予測できる、というわけです。例えば、気温が30度であれば、0.307 × 30 + 4.397 = 13.607万円が、予測される売り上げの金額です。

● 分類

分類は、データが属する「クラス」の予測が目的です。ここでのクラスというのはカテゴリーの意味ですが、分類は英語でclassificationなので、クラスという用語が使われます。

気温とアイスクリームの例で言うと、ある気温でアイスクリームが「たくさん売れる」か「あまり売れない」かを予測するのが分類です。この場合、クラスは「たくさん売れる」と「あまり売れない」の2種類で、結果は

図5●線形回帰の例。過去の気温とアイスクリームの売り上げの関係のデータから予測のモデルを作る

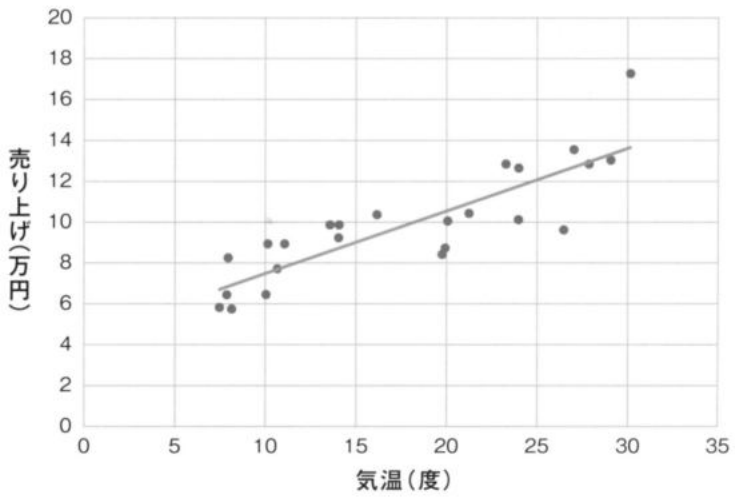
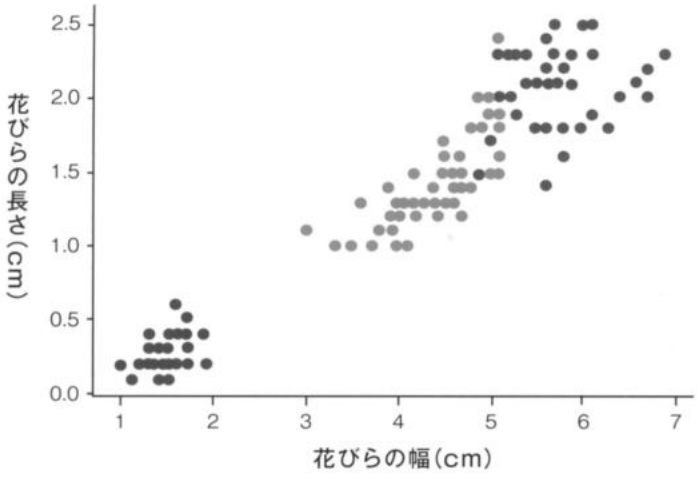


図6●クラスタリングの例。花びらの長さと幅のデータから、アヤメの花を3つのグループに分けた



必ずこのどちらかになります。分類も、教師あり学習の手法が使われます。

●グループ分け

グループ分けとは、データを似たもの同士のグループに分けることです。分類と似ていますが、あらかじめクラスを用意するわけではない点が異なります。グループ分けは、教師なし学習の「クラスタリング」と呼ばれる手法で実現します。

図6はクラスタリングの例です。散在する丸は、アヤメの花を採取して長さを測り、一つずつプロットしたものです。縦軸は花びらの長さ、横軸は花びらの幅です。これをクラスタリングによって、赤、緑、青の3つのグループに分けました。このグループ分けによって、同じだと思っていたアヤメは、実は3つの品種に分かれているのではないかと推測できます。このように、グループ分けによって見えてくることがあるわけです。

ところで、例に出したアヤメの花は、機械学習の入門で定番になっているデータです。ほかにも、「MNIST」(手書き数字の画像)や「タイタニック号の乗客リスト」といった、入門でよく出てくる定番のデータがいくつかあります。

●情報の要約

情報の要約とは、多数のデータを少数のデータに、そのデータの特徴を保ったまま要約することです。例えば、身長と体重という2つのデータ(2次元のデータ)から、肥満度という1つのデータ(1次元のデータ)を求める次のBMIの計算は、最も簡単な情報の要約の例と言えます。

```
BMI(肥満度)= 体重(kg)÷身長(m)÷身長(m)
```

情報の要約は、教師なし学習の「次元削減」と呼ばれる手法で実現します。

ここまでの説明をまとめたものが図7です。予測、分類、グループ分け、情報の要約を実現するために、教師あり学習と教師なし学習に分類される様々な機械学習のアルゴリズムがあります。図7の右側に、そのいくつかを記載しました。

この特集のPart 2以降では、これらの中から、基本的なアルゴリズムをいくつかピックアップして、Pythonのコードとともに、その実装について解説していきます。ただし、次元削減は少しハードルが高いので、ここでは扱いません。

図7●機械学習の用途と、それを実現するためのアルゴリズム

用途			アルゴリズム
機械学習	予測	教師あり学習	線形回帰、サポートベクター回帰(SVR)、リッジ回帰、ラッソ回帰、ニューラルネットワークなど
	分類	教師あり学習	k近傍法(kNN)、サポートベクターマシン(SVM)、ランダムフォレスト、ニューラルネットワークなど
	グループ分け	教師なし学習(クラスタリング)	k平均法(k-Means)、スペクトルクラスタリング、ミーンシフト、ニューラルネットワークなど
	情報の要約	教師なし学習(次元削減)	主成分分析(PCA)、特徴選択、非負値行列因子分解、ニューラルネットワークなど

▷Part 2

機械学習をAIライブラリで実装しよう

分類、回帰、クラスタリングを実現！

統計学などの専門知識があまりなくても、AIライブラリを使えば、機械学習のプログラムを手軽に実装できます。PythonにはAIライブラリがいくつもありますが、初学者向けのライブラリとしては「scikit-learn」が定番です。

そこでPart 2では、scikit-learnが用意しているアルゴリズムを使って、「分類（教師あり学習）」「回帰（教師あり学習）」「クラスタリング（教師なし学習）」のプログラムを実装してみます。

表1●「scikit-learn」で使用できる主な機械学習アルゴリズム

手法	アルゴリズム
分類	k近傍法（kNN）、サポートベクトルマシン（SVM）、ランダムフォレストなど
回帰	線形回帰、サポートベクトル回帰（SVR）、リッジ回帰、ラッソ回帰など
クラスタリング	k平均法（k-Means）、スペクトルクラスタリング、ミーンシフトなど
次元削減	主成分分析（PCA）、特徴選択、非負値行列因子分解など

scikit-learnを使う準備をしよう

scikit-learnは、基本的な機械学習のアルゴリズムを備えたライブラリです。表1に示した「分類」「回帰」「クラスタリング」「次元削減」の4種類の手法から用途に合ったアルゴリズムを選択し、機械学習のプログラムで利用できます。

Part 2で使用するのは、以下の3つのアルゴリズムです。

- k近傍法（分類）
- 線形回帰（回帰）
- k平均法（クラスタリング）

「k近傍法」は、教師あり学習の分類のアルゴリズムです。「線形回帰」は、教師あり学習の回帰のアルゴリズムで、予測を行います。「k平均法」は、教師なし学習のクラスタリングのアルゴリズムで、グループ分けを行います。

scikit-learnでは、これらのアルゴリズムの違いをあまり意識することなく、シンプルなコードで機械学習の

Pythonの代表的なAIライブラリ

Pythonには、scikit-learn以外にもよく使われるAIライブラリがあります。代表的なのは「XGBoost」「TensorFlow」「PyTorch」といったライブラリです。

XGBoostは、分類や予測を行う教師あり学習で精度の高い機械学習モデルを作れるライブラリです。「勾配ブースティング木」というアルゴリズムを使ってモデルを構築します。このアルゴリズムは、アンサンブル法の一種である「ブースティング」という手法を使っています。アンサンブル法とは、複数のモデルを組み合わせてより優れたモデルを作る手法のことです。XGBoostは、「Kaggle」などのデータサイエンスのコンペティションでも人気のライブラリです。scikit-learnでの機械学習の実装に慣れたら、次にチャレンジしてみると良いでしょう。

TensorFlowとPyTorchは、本格的な深層学習（ディープラーニング）のプログラムを実装できるライブラリです。高速演算を行うGPUを使った学習ができます。scikit-learnでも、ニューラルネットワークのアルゴリズムを使って深層学習のモデルを実装することはできますが、GPUに対応していません。しかし、精度の高い深層学習を行うためにはGPUは必須と言えます。よって、本格的な深層学習を実装する場合は、TensorFlowやPyTorchを使うのが一般的です。

プログラムを実装できます。それぞれの実装方法は、あとで詳しく説明していきます。

● 開発環境を構築する

Part 2とPart 3を通して、開発環境には「Anaconda」を利用します。Anacondaは、Pythonのプログラミングに必要な環境やツール、AI・データサイエンス向けなどの定番ライブラリをパッケージ化したものです。scikit-learnも、Anacondaに含まれています。

Anacondaには、個人ユーザー向けの「Anaconda Distribution」というエディションがありますので、これをインストールします。下記のURLから公式サイトにアクセスし、「Download」をクリックしてください。

```
https://www.anaconda.com/products/distribution
```

ダウンロードしたインストーラーを起動します。本稿執筆時点(2022年3月)では、インストーラーのファイル名は「Anaconda3-2021.11-Windows-x86_64.exe」となっています。インストーラーの画面の指示に従って作業をすれば、インストールは完了します。

図1●「Anaconda Navigator」のHOME画面

● Jupyter Notebookを使う

プログラムの実行には、Anacondaに含まれている「Jupyer Notebook」を使います。Jupyter Notebookは、Webブラウザ上で対話式にプログラムを記述・実行できるツールです。機械学習モデルの実装で、一部のパラメータだけを変更して何度も同じプログラムを実行するようなときに便利です。

Jupyter Notebookを起動しましょう。Windowsのスタートメニューから「Anaconda3」→「Anaconda Navigator(Anaconda)」をクリックし、Anaconda NavigatorのHOME画面を開きます(**図1**)。HOME画面で、Jupyter Notebookの「Launch」ボタンをクリックして起動してください。

Jupyter Notebookを起動すると、**図2**の画面が表示されます。この画面の右上の「New」→「Python 3」を選択してください(**図3**)。新規の「ノートブック」の画面が表示されます。

ノートブックの画面で、Pythonのコードの入力と実行結果の確認を行います。**図4**は、「3+5」という計算式を入力して実行し、実行結果の「8」を表示した画面です。In[1]の右の「セル」と呼ばれる部分にコードを

図2●Webブラウザで開いた「Jupyter Notebook」の画面

図3●「Python 3」を選択して新規のノートブックを作成する

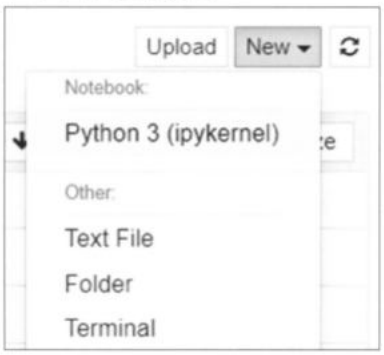
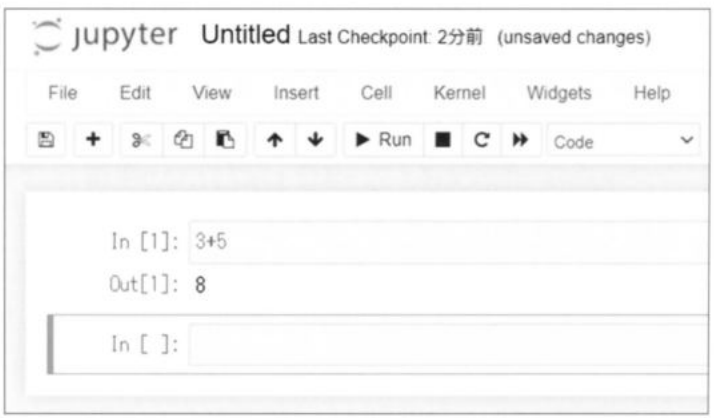


図4●セルに「3+5」と入力して「Run」をクリックすると実行結果の「8」が表示された



入力し、「Run」ボタンをクリックすることでプログラムを実行します。実行結果はセルの下（ここではOut[1]の右）に表示されます。このあとに続けてコードを記述するときは、実行結果の下のIn[　]のセルに入力します。

分類のプログラムを実装しよう

分類（k近傍法）、回帰（線形回帰）、クラスタリング（k平均法）の実装方法を順番に説明していきます。

機械学習プログラムでよく使うライブラリ

本稿では、機械学習モデルを実装するscikit-learnのほかに、「pandas」「Matplotlib」「NumPy」というPythonのライブラリを使用します。データ分析やグラフ出力のためのライブラリで、機械学習のプログラムで一般的に使われています。これらはAnacondaのパッケージに含まれているので、個別にインストールする必要はありません。

それぞれを簡単に紹介します。pandasは、データ分析向けのライブラリです。「データフレーム」という行と列で構成された表形式のデータ構造で、データを操作できます。

Matplotlibは、グラフを描画するライブラリです。機械学習のプログラムで使用するデータをグラフにすることで、視覚的にわかりやすく理解できるようになります。

NumPyは、数値計算用のライブラリです。「NumPy配列」という多次元配列を表すデータ構造を提供します。Pythonの標準のリストよりも、高速にデータを操作できます。

まずは、分類のプログラムを実装します。使用するアルゴリズムはk近傍法です。Part 1でも出てきた気温とアイスクリームの関係を例にします。ここでは「アイスクリームの売り上げ」ではなく、顧客がアイスクリームを「買う」か「買わない」かという行動のデータを使用します。

図5のように、過去の「気温」と「顧客の購買行動」（アイスクリームを「買う」か「買わない」か）のデータを学習用データとします。この学習用データから分類用のモデルを作り、過去のデータにない未知の気温（ここでは30度）のときに、顧客がアイスクリームを「買う」か「買わない」かを分類します。

この機械学習を行う目的は以下の通りです。

目的
気温30度のときに、顧客がアイスクリームを「買う」か「買わない」かを分類すること

最終的に分類したい結果が「顧客の購買行動」で、分類のために使うデータが「気温」です。

教師あり学習において、「顧客の購買行動」のように、分類（や予測）の結果として求めたいデータのことを「目的変数」と呼びます。また、「気温」のように、分類（や予測）の結果に影響するデータのことは「説明変数」と呼びます。

では、この例を実装していきましょう。機械学習の実装手順は、基本的には以下の4ステップになります。

(1) 入力データを準備する
(2) モデルに学習させる
(3) モデルを使って分類（や予測）をする
(4) モデルの精度の評価と改善をする

図5●分類のプログラムのイメージ

図6●「気温と購買行動の関係」のデータ

気温	購買行動
5	買わない
10	買わない
20	買う
22	買わない
25	買う

(2)でアルゴリズムを選択してモデルを作り、(3)でそのモデルを使って分類(や予測)を行います。順番に説明していきます。

●入力データを準備する

モデルを作るための学習用データを準備します。機械学習では大量のデータをファイルから読み込むことが一般的ですが、ここでは説明をわかりやすくするために、少量のデータを使います。Pythonのリスト(配列)で、図6のような「気温と購買行動の関係」のデータを用意します。

次のプログラムを、Jupyter Notebookのセルに入力して実行してください。

```
x = [[5], [10], [20], [22], [25]]
```

図7●k近傍法のイメージ

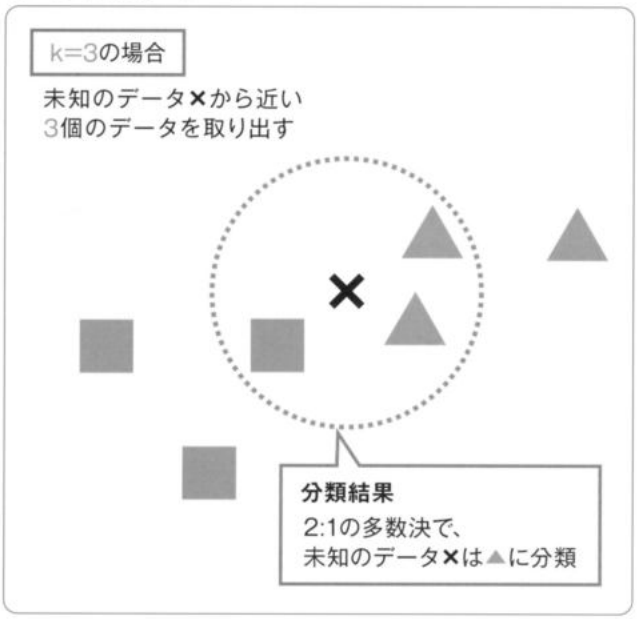


図8●k近傍法のモデルを実装した

```
In [1]: x = [[5], [10], [20], [22], [25]]
        y = ["買わない", "買わない", "買う", "買わない", "買う"]

In [2]: from sklearn.neighbors import KNeighborsClassifier
        knn = KNeighborsClassifier()
        knn.fit(x, y)

Out[2]: KNeighborsClassifier()
```

```
y = ["買わない", "買わない", "買う", "買わない",
"買う"]
```

xには説明変数である「気温」を、yには目的変数である「購買行動」を格納します。xは、2次元リストになっていることに注目してください。scikit-learnでは、説明変数を2次元リストで定義する決まりになっています。

●モデルに学習させる

学習用データを入力し、分類用のモデルを作ります。ここで使うk近傍法を説明しましょう。

k近傍法とは、未知のデータを分類するときに、未知のデータと学習用データとの距離を計算し、距離の近いk個の学習用データを取り出して多数決を行うというアルゴリズムです。kの値はあらかじめ決めておきます。図7を見てください。中央の×印が未知のデータです。その周囲の三角形や四角形のデータは学習用データを表しています。k=3とした場合、未知のデータに近い3個の学習用データを選びます。そして、選んだデータの種類(三角形か四角形か)を見て、数が多い種類に未知のデータを分類します。

k近傍法を使ったモデルを作りましょう。先ほどのプログラムの下のセルに、次のプログラムを入力して実行します。

```
from sklearn.neighbors import KNeighborsCla
ssifier
knn = KNeighborsClassifier()
knn.fit(x, y)
```

1行目では、k近傍法のモジュールをインポートしています。2行目で未学習のモデルを作成し、3行目でfit関数を呼び出して学習データをモデルに入力しています。この入力が、モデルに学習させる処理です。fit関数の引数x、yは、先ほど作成した「気温」と「購買行動」のデータです。

k近傍法であらかじめ決めておくべきkの値は、ここではデフォルトの値(k=5)で実行されます。

実行結果は図8です。KNeighborsClassifier()と表示されていれば、モデルの学習は完了です。

●モデルを使って分類する

作成したモデルに未知のデータ(ここでは気温30度)を入力して、分類を行います。次のプログラムを実行してください。

```
knn.predict([[30]])
```

predict関数で分類を行っています。predict関数の引数は、2次元リストにする必要があります。

実行結果は図9です。気温30度のときは「買わない」と分類されています。ここまでたったの6行で、簡単に機械学習のプログラムを実装し、分類結果を確認できました。しかし、この結果は正しいのでしょうか。図6の学習用データ(過去のデータ)を見ると、気温が高いときにアイスクリームを買う傾向がありそうです。気温30度のときにアイスクリームを「買わない」と分類されるのは、少し違和感があります。

機械学習は、モデルを作って分類(や予測)を行うだけで終わりではありません。モデルの精度、つまり、「作成したモデルの分類がどのくらい正しいのか?」という点を評価し、改善していくことも重要です。

図9●k近傍法のモデルで気温30度のときの購買行動を分類した

```
In [3]: knn.predict([[30]])
Out[3]: array(['買わない'], dtype='<U4')
```

図10●「気温と購買行動の関係」の実際のデータと分類結果のデータを表示した

```
In [5]: print("実際のデータ=", y)
        print("分類結果のデータ=", knn.predict(x))

実際のデータ= ['買わない', '買わない', '買う', '買わない', '買う']
分類結果のデータ= ['買わない' '買わない' '買わない' '買わない' '買わない']
```

●モデルの精度を改善する

先ほど作成したモデルの精度を、次のコードで検証してみましょう。

```
knn.score(x, y)
```

実行結果は次のようになります。

```
0.6
```

これは、このモデルを使って分類をするときの正解率が6割であるという意味です。

学習用データを使って、実際のデータと分類結果のデータを比べてみましょう。

```
print("実際のデータ=", y)
print("分類結果のデータ=", knn.predict(x))
```

1行目の「実際のデータ」では、図6の学習用データの「購買行動」をそのまま表示します。2行目の「分類結果のデータ」では、図6の学習用データの「気温」のデータを、先ほど作成したモデルに入力し、分類した結果を表示します。

図10が実行結果です。比較すると、分類結果のデータの5つのうち、先頭から3番目と5番目が実際のデータと異なっています。正解率は5分の3なので、モデルの精度は0.6です。「買う」か「買わない」かの二択の問題で0.6というのは少し精度が低いと言えます。精度を改善していきましょう。

モデルの精度を上げる方法はいくつかありますが、ここでは、アルゴリズムのパラメータ(ハイパーパラメータ)を調整する方法を説明します。

図7に示したk近傍法の手法を思い出してください。k近傍法は、未知のデータと距離が近いk個の学習用データを取り出して多数決で結果を決めるというアルゴリズムです。このアルゴリズムでは、kの値を適切に設定することが重要です。kの値のように、アルゴリズムの挙動を制御するパラメータのことを「ハイパーパラメータ」と言います。

先ほどモデルを作ったときには、kの値をプログラムで指定しなかったので、scikit-learnのk近傍法のモジュールにデフォルトで設定されている値を使いました。次のプログラムをセルに入力して実行すると、ハイパーパラメータkのデフォルトの値を表示します。

```
print(knn.n_neighbors)
```

実行結果は次の通りです。

```
5
```

先ほどのモデルは、k=5のk近傍法で作られたことを確認できました。ここで使った学習用データは、「買わない」「買わない」「買う」「買わない」「買う」の5件です。未知のデータを入力して分類するときに、5件すべてのデータを取り出して多数決を行ったということになります。5件のデータは、「買わない」が3件で「買う」が2件でした。つまり、5件すべてのデータで多数決をとると「買わない」と分類されます。なので、先ほどpredict関数で分類を行った結果が「買わない」と判定されていたのです。5件すべてのデータを取り出す(k=5)よりも、より距離の近いデータだけを取り出した方が精度が上がりそうです。kの値を小さくしてみましょう。

k近傍法のkの値を4に設定してモデルを構築します。

```
knn.n_neighbors = 4
knn.fit(x, y)
knn.score(x, y)
```

このプログラムを実行すると、次のようになります。

```
0.8
```

精度が0.8と表示されます。先ほどのモデルの精度の0.6よりも向上したことがわかります。精度が0.8のこのモデルを使って、気温30度のときの購買行動を分類してみましょう。

```
knn.predict([[30]])
```

図11●精度が0.8のモデルで分類した

```
In [8]: knn.predict([[30]])
Out[8]: array(['買う'], dtype='<U4')
```

実行結果は図11です。購買行動は「買う」に分類されています。このように、モデルを作るときは、ハイパーパラメータの設定が重要です。

回帰のプログラムを実装しよう

回帰の手法でモデルを作り、予測を行うプログラムを実装しましょう。使用するアルゴリズムは線形回帰です。引き続き、気温とアイスクリームの例で説明します。ここでは、過去の「気温」と「アイスクリームの売り上げ」のデータから予測のモデルを作り、未知の気温(30度)のときのアイスクリームの売り上げを予測します(図12)。先ほどの分類のプログラムでは、結果は「買う」か「買わない」かのどちらかを出力しました。予測のプログラムでは、例えば290万円、300万円、310万円といった売り上げ金額の数値そのものを予測し、結果として出力します。

この機械学習を行う目的は以下の通りです。

目的
気温30度のときのアイスクリームの売り上げを予測すること

説明変数は「気温」で、目的変数は「アイスクリームの売り上げ」です。分類の手法のときと同じように、4つのステップで実装を行います。

● 入力データを準備する

モデルを作るための学習用データは、CSVファイルを読み込んで使用します。本稿では「ice_sales.csv」というファイル名の「気温とアイスクリームの売り上げ

図12●回帰のプログラムのイメージ

の関係」のデータを用意しました。日経ソフトウエアのWebページ(https://nkbp.jp/nsoft)からサンプルプログラムをダウンロードしてください。サンプルプログラムが入っているフォルダーの中に、ice_sales.csvが入っています。このファイルを、ノートブックを作成したフォルダーと同じフォルダー(Windowsでデフォルトだと C:¥Users¥ユーザー名)に格納してください。

図13●「気温とアイスクリームの売り上げの関係」のデータ(ice_sales.csv)

	A	B
1	temp	sales
2	7.5	5.8
3	8	8.2
4	11.1	8.9
5	16.2	10.3
6	21.3	10.4
7	23.3	12.8
8	27.1	13.5
9	27.9	12.8
10	24	10.1
11	20.1	10
12	14.1	9.2
13	10.1	6.4
14	8.2	5.7
15	7.9	6.4
16	10.2	8.9
17	13.6	9.8
18	20	8.7
19	24	12.6
20	29.1	13
21	30.2	17.2

ice_sales.csvは、Excelで開くと**図13**のように表示されます。このCSVファイルを、データ分析ライブラリであるpandasを使ってプログラムで読み込みましょう。新しいノートブックを作成し、次のプログラムを入力して実行します。

```
import pandas as pd
df = pd.read_csv("ice_sales.csv")
df.head()
```

1行目のimport文でpandasを読み込みます。2行目のread_csv関数では、ice_sales.csvを読み込み、「df」という名前のデータフレームに格納しています。データフレームに格納することで、行と列を持つ表形式の構造でデータを扱えるようになります。3行目のhead関数では、dfの先頭5行を表示しています。

実行結果は**図14**です。図13のCSVファイルの1行目がデータフレームの列名として読み込まれ、2～6行目がデータフレームの先頭5行のデータとして表示されています。

次に、モデルの学習に使う説明変数xと目的変数yを設定します。先ほどのプログラムの下のセルに次のプログラムを入力して実行してください。

図14●ice_sales.csvを格納したデータフレームdfの先頭5行を表示した

```
In [1]: import pandas as pd
        df = pd.read_csv("ice_sales.csv")
        df.head()
```

Out[1]:

	temp	sales
0	7.5	5.8
1	8.0	8.2
2	11.1	8.9
3	16.2	10.3
4	21.3	10.4

図15●線形回帰のモデルを実装した

```
In [2]: x = df.temp.to_frame()
        y = df.sales

In [3]: from sklearn.linear_model import LinearRegression
        lr = LinearRegression()
        lr.fit(x, y)

Out[3]: LinearRegression()
```

```
x = df.temp.to_frame()
y = df.sales
```

説明変数xにdfのtemp(気温)を、目的変数yにdfのsales(アイスクリームの売り上げ)を格納しています。scikit-learnでは説明変数を2次元リストにする必要があるので、1行目のto_frame関数で2次元リストに変換します。

モデルに学習させる

学習用データで学習を行い、線形回帰のモデルを作ります。線形回帰のモデルは、「気温とアイスクリームの売り上げの関係」を表す1次関数(「y = ax + b」で表される式)を、最小二乗法という手法で求めます。最小二乗法の詳細はPart 3で説明します。

次のプログラムで線形回帰のモデルを作りましょう。

```
from sklearn.linear_model import LinearRegression
lr = LinearRegression()
lr.fit(x, y)
```

1行目で線形回帰のモジュールをインポートし、2行目では未学習のモデルを作成しています。3行目でfit関数を呼び出して、モデルの学習を行います。fit関数の引数x、yは、先ほど設定した「気温」と「アイスクリームの売り上げ」です。

図16●線形回帰のモデルで気温30度のときのアイスクリームの売り上げを予測した

```
In [4]: lr.predict([[30]])
Out[4]: array([13.62120947])
```

実行結果は図15です。LinearRegression()と表示されれば学習完了です。

●モデルを使って予測する

作成したモデルに未知のデータ(気温30度)を入力して、売り上げを予測します。次のコードを入力して実行してください。分類のときと同じpredict関数を使います。predict関数の引数は2次元リストで指定します。

```
lr.predict([[30]])
```

実行結果は図16です。気温30度のときの売り上げは、「13.62120947」万円と予測されています。

●モデルの精度を評価する

このモデルの精度を評価してみましょう。

```
lr.score(x, y)
```

このコードを実行すると、次のように表示されます。

```
0.7261874233533446
```

これは、このモデルを使って予測をするときの正解率が約7割であるという意味です。学習用データを使って、実際のデータと予測結果のデータを比べてみましょう。

```
print("実際のデータ", df.sales.values)
print("予測データ", lr.predict(x))
```

1行目の「実際のデータ」では、学習用データの「アイスクリームの売り上げ」をそのまま表示しています。2行目の「予測データ」では、学習用データの「気温」のデータを、先ほど作成したモデルに入力し、予測した結果を表示しています。

図17が実行結果です。実際のデータ5.8に対して予測データ6.70272434、実際のデータ8.2に対して予測データ6.85646845…のように対応しています。このままでは比較しにくいので、グラフにしてみます。次のコードを入力して実行してください。グラフを描画するライブラリであるMatplotlibを使っています。

```
import matplotlib.pyplot as plt
plt.scatter(df.temp, df.sales)
plt.plot(df.temp, lr.predict(x))
plt.show()
```

1行目でMatplotlibをインポートします。2行目では、scatter関数を使って、「気温」と実際のデータの「アイスクリームの売り上げ」の関係を、散布図で表示しています。3行目では、「気温」と予測データの「アイスクリームの売り上げ」の関係を、直線で表示しています。

4行目のshow関数はグラフを出力する関数ですが、Jupyter Notebookの環境で実行する場合は省略できます。

実行すると図18のようなグラフが表示されます。丸い点は、実際のデータの「気温とアイスクリームの売り上げの関係」です。直線は、予測データの「気温とアイスクリームの売り上げの関係」です。つまり、この直線は、先ほど作成したモデルそのものを表しています。これを回帰直線と言います。

モデルを1次関数「y = ax + b」の形式で表示します。次のコードを実行してください。

```
print('sales = %.3ftemp + %.3f' % (lr.coef_
, lr.intercept_))
```

lr.coef_でモデルの傾き(a)を、r.intercept_でモデルの切片(b)を表示しています。

図17●「気温とアイスクリームの売り上げの関係」の実際のデータと予測データを表示した

```
In [6]: print("実際のデータ", df.sales.values)
        print("予測データ", lr.predict(x))

実際のデータ [ 5.8  8.2  8.9 10.3 10.4 12.8 13.5 12.8 10.1 10.   9.2  6.4  5.7  6.4
  8.9  9.8  8.7 12.6 13.  17.2  9.6  8.4  9.8  7.7]
予測データ [ 6.70272434  6.85646845  7.80968196  9.37787192 10.94606189 11.56103834
 12.72949361 12.97548419 11.7762801  10.57707601  8.73214664  7.50219373
  6.9179661   6.82571963  7.53294256  8.57840253 10.54632719 11.7762801
 13.34447006 13.68270711 12.54500067 10.48482954  8.73214664  7.68668667]
```

実行結果は図19です。次の式が表示されています。

```
sales = 0.307temp + 4.397
```

傾き(a)が0.307で、切片(b)が4.397です。

この式の「temp」に気温の値を代入すると、アイスクリームの売り上げの予測結果を求められます。次のコードを実行します。

```
0.307*30 + 4.397
```

実行結果は次の通りです。

```
13.607
```

これは、先ほどpredict関数を使って予測した結果(図16)とほとんど一致しています。

回帰の手法でモデルの精度を上げる方法の一つは、説明変数の数を増やすことです。「気温」に加えて「休日かどうか」など、アイスクリームの売り上げに関連が高そうな説明変数を追加できれば、より精度の高いモデルを作成できるでしょう。このように説明変数が複数のモデルのことを「重回帰モデル」と言います。先ほどの例のように説明変数が「気温」の1つであるモデルは、「単回帰モデル」と言います。

クラスタリングのプログラムを実装しよう

クラスタリングのプログラムを実装します。クラスタリングにはいくつかのアルゴリズムがありますが、基本的なk平均法を使います。

ここでは、都道府県を、「人口」と「出生率」という特徴に基づいてグループ分け(クラスタリング)します。グループ分けといっても、前述の分類とは違います。分類は「買う」「買わない」といった分けるクラス(カテゴリー)が決まっています。一方のクラスタリングは、データの特徴を見て、その特徴によって似ているデータ同士をグループ分けするのです。

クラスタリング(教師なし学習)では、教師あり学習のような「モデルを作って未知のデータの予測を行う」という目的はありません。データの特徴やその特徴の背景を明らかにするために行います。このため、説明変数や目的変数という概念もありません。

scikit-learnによる実装の手順は、基本的にはこれまでと同じです。

● 入力データを準備する

グループ分けを行うデータを準備します。ここではサンプルとして用意した「population.csv」というファイル名の「都道府県ごとの人口と出生率の関係」のデータを使用します(図20)。このデータは日本政府が提供するe-Stat[*1]という統計データを加工したものです。

population.csvを、先ほどと同様に日経ソフトウエアのWebページからダウンロードし、ノートブックと同じ

図18●「気温とアイスクリームの売り上げの関係」の実際のデータと回帰直線をグラフで表示した

```
In [7]: import matplotlib.pyplot as plt
        plt.scatter(df.temp, df.sales)
        plt.plot(df.temp, lr.predict(x))
        plt.show()
```

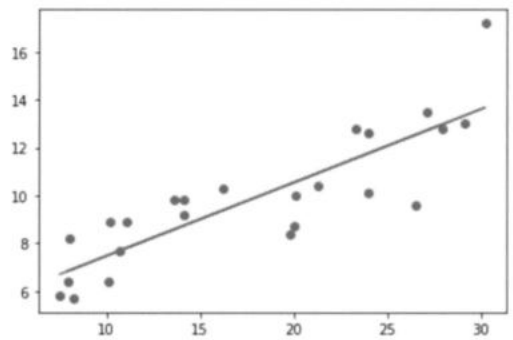


図19●線形回帰のモデルを1次関数で表した

```
In [8]: print('sales = %.3ftemp + %.3f' % (lr.coef_ , lr.intercept_))
        sales = 0.307temp + 4.397
```

*1 e-StatのWebサイトには、https://www.e-stat.go.jp/ からアクセスできます。

フォルダーに格納します。

このCSVファイルを読み込みます。新しいノートブックを作成し、次のコードを入力して実行してください。

```
import pandas as pd
df = pd.read_csv("population.csv")
df.head()
```

1行目でpandasをインポートします。2行目では、CSVファイルをdfという名前のデータフレームに読み込みます。3行目でデータフレームの先頭5行を表示します。

実行結果は**図21**です。nameが都道府県名、populationが人口、rateが出生率を示しています。

図20●「都道府県ごとの人口と出生率の関係」のデータ(population.csv)

	A	B	C
1	name	population	rate
2	Hokkaido	525	1.24
3	Aomori-ken	125	1.38
4	Iwate-ken	123	1.35
5	Miyagi-ken	231	1.23
6	Akita-ken	97	1.33
7	Yamagata-ken	108	1.4
8	Fukushima-ken	185	1.47
9	Ibaraki-ken	286	1.39
10	Tochigi-ken	193	1.39
11	Gumma-ken	194	1.4
12	Saitama-ken	735	1.27
13	Chiba-ken	626	1.28
14	Tokyo-to	1392	1.15
15	Kanagawa-ken	920	1.28
16	Niigata-ken	222	1.38
17	Toyama-ken	104	1.53
18	Ishikawa-ken	114	1.46
19	Fukui-ken	77	1.56
20	Yamanashi-ken	81	1.44
21	Nagano-ken	205	1.57

図21●population.csvを格納したデータフレームdfの先頭5行を表示した

```
In [1]: import pandas as pd
        df = pd.read_csv("population.csv")
        df.head()
```

Out[1]:

	name	population	rate
0	Hokkaido	525	1.24
1	Aomori-ken	125	1.38
2	Iwate-ken	123	1.35
3	Miyagi-ken	231	1.23
4	Akita-ken	97	1.33

●モデルに学習させる

準備したデータを使って、k平均法によるクラスタリングを実装します。

k平均法の手順を簡単に説明します。まず、いくつのクラスタ(グループ数)に分けたいかを決めます。これをkという値として設定します。ここでは、k=2として2つのクラスタに分けます。

次に、データの中からk個の任意のデータ(ここでは2つのデータ)を各クラスタの中心として選びます。そのほかのすべてのデータと2つの中心との距離をそれぞれ計算し、距離が近い方のクラスタに所属するようにグループ分けを行います(**図22**)。その後、クラスタの重心を中心にしてグループ分けを繰り返す処理を行います。詳細はPart 3で説明します。

次のプログラムを実行し、k平均法を使ったクラスリングを実装しましょう。

```
from sklearn.cluster import KMeans
km = KMeans(n_clusters = 2)
km.fit(df.drop("name", axis = 1))
```

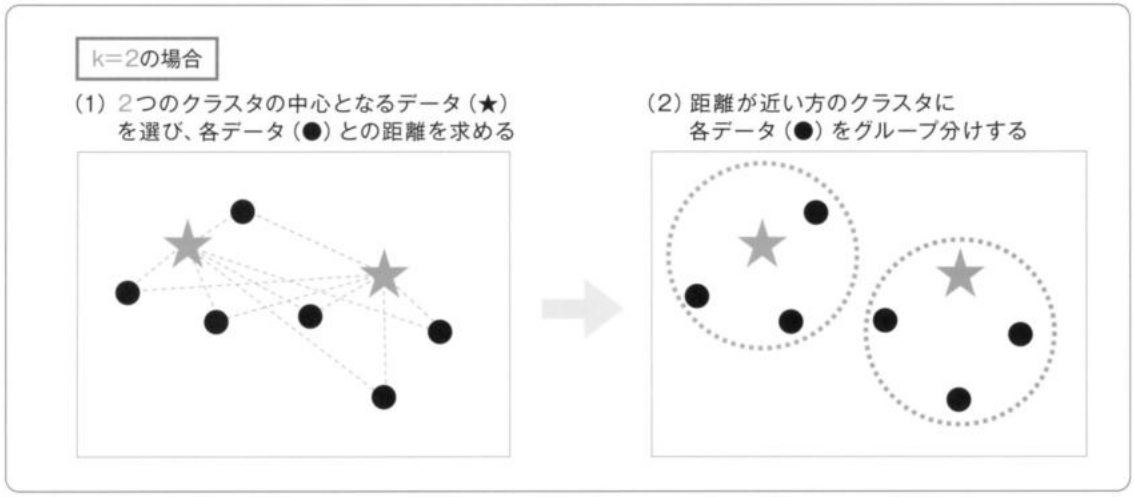
1行目でk平均法のモジュールをインポートし、2行目でオブジェクトを作成しています。引数のn_clustersには、クラスタの中心の個数kの値を指定します。n_clustersを指定しない場合は、デフォルトのk=8で実行されます。3行目でfit関数を呼び出してデータを入力します。fit関数の引数(df.drop("name", axis = 1))では、name(都道府県名)の列を取り除き、それ以外のpopulationとrateのデータを指定しています。

実行結果は**図23**です。KMeans(n_clusters=2)と表示されればクラスタリングは完了です。

●結果を散布図で表示する

クラスタリングの結果を確認します。次のコードを実行してください。

図22●k平均法の最初の処理のイメージ




```
df["group"] = km.labels_
df.head()
```

1行目では、groupという名前の列を新規に作成し、クラスタリングの結果であるkm.labels_を格納しています。2行目でkm_labels_の値を含む先頭5行を表示します。

実行結果は**図24**のように表形式で表示されます。この表の、groupの列を見てください。Hokkaido(北海道)は「1」、その他の4つの県は「0」というグループに分けられていることがわかります。

表形式では確認しにくいので、Matplotlibを使って結果を散布図で表示します。**リスト1**を入力してください。

1行目でMatplotlibをインポートしています。2行目でX軸にpopulation(人口)を、3行目でY軸にrate(出生率)を設定しています。

4行目ではグラフのサイズを設定しています。

5行目では、図24に示したgroupの値が「0」の都道府県を青色に、「1」の都道府県を赤色で表示しています。

6行目のscatter関数で、2～5行目の設定を反映した散布図を表示しています。

7行目以降で、forループを使ってグラフ上の各点に対応するname(都道府県名)を表示しています。

実行すると**図25**が表示されます。この散布図から、都道府県は以下の2つのグループに分けられていることがわかります。

図23●k平均法でクラスタリングのプログラムを実装した

```
In [2]: from sklearn.cluster import KMeans
        km = KMeans(n_clusters = 2)
        km.fit(df.drop("name", axis = 1))

Out[2]: KMeans(n_clusters=2)
```

図24●クラスタリングの結果を表形式で表示した

```
In [3]: df["group"] = km.labels_
        df.head()
```

Out[3]:

	name	population	rate	group
0	Hokkaido	525	1.24	1
1	Aomori-ken	125	1.38	0
2	Iwate-ken	123	1.35	0
3	Miyagi-ken	231	1.23	0
4	Akita-ken	97	1.33	0

リスト1●クラスタリングの結果を散布図で表示する

```
import matplotlib.pyplot as plt
x = df.population
y = df.rate
plt.rcParams["figure.figsize"] = (10, 10)
df["color"] = df.group.map({0:"blue", 1:"red"})
plt.scatter(x, y, c=df.color)
for i in range(len(x)):
  plt.text(x[i], y[i], df.name[i], fontsize=16)
```

図25●クラスタリングの結果を散布図に表示した

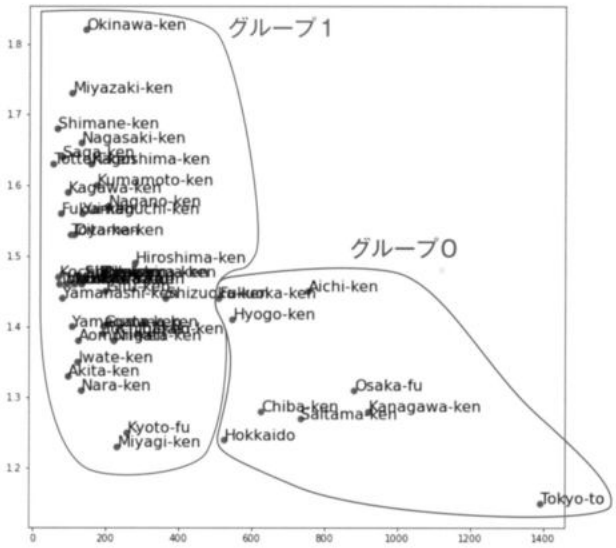


図26●k=3でクラスタリングを行った結果を散布図に表示した

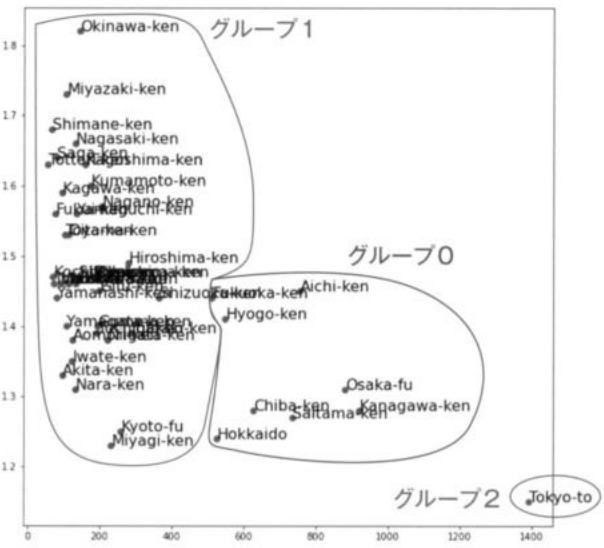
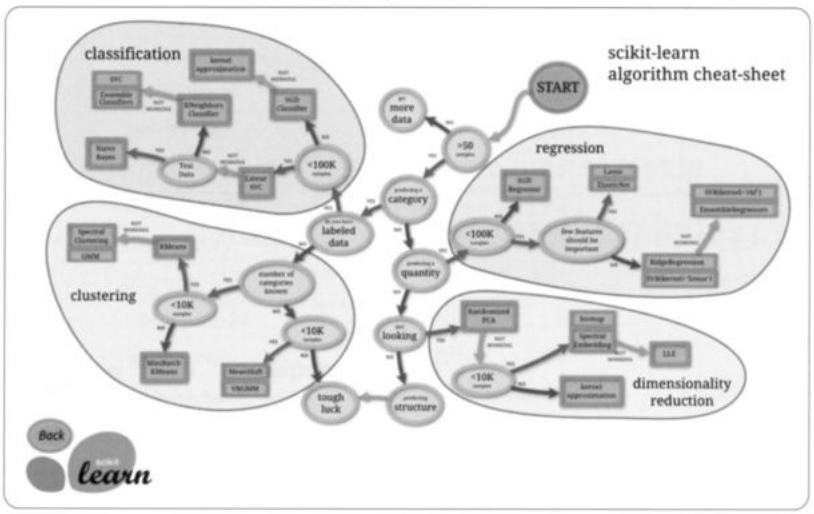


図27●scikit-learnの「アルゴリズムチートシート」(出典：https://scikit-learn.org/stable/tutorial/machine_learning_map/)



グループ0(赤丸)：出生率が低く人口が多いグループ(東京・大阪など)
グループ1(青丸)：人口が少ないグループ(沖縄・宮崎など)

グループ0は「都市型」、グループ1は「郊外型」などと名付けられそうです。

●k=3で実行してみる

教師なし学習には正解となる教師データがないので、教師あり学習のモデルのような、「モデルの精度の数値」(評価指標)はありません。

k=2の結果で納得できない場合は、kの値を変更し、クラスタ数を変えて改善を行います。例えば、k=3で3つのクラスタに分ける場合は、n_clustersの値を次のように変えます。

```
km = KMeans(n_clusters = 3)
```

さらに、この結果を散布図に反映するために、リスト1の5行目を次のように変更します。groupの値が「2」の都道府県を、緑色で表示します。

```
df["color"] = df.group.map({0:"
blue", 1:"red", 2:"green"})
```

実行すると、図26のように東京が別のグループに分けられ、緑色に表示されます。3つのクラスタにグループ分けされたことがわかります。

ここまで、k近傍法、線形回帰、k平均法を使った機械学習のプログラムを紹介しました。機械学習のアルゴリズムには、ほかにもたくさんの種類があります。どのアルゴリズムを使うべきかを判断することは、初学者には難しく感じられるでしょう。scikit-learnには、図27のようなフローチャート形式で質問に答えていくことで最適なアルゴリズムを選択できる「アルゴリズムチートシート」が付属しています。参考にしてみてください。

Part 3では、scikit-learnを使わずに、線形回帰とk平均法のアルゴリズムを実装してみます。

▷Part 3

アルゴリズムをゼロから実装！

AIライブラリを使わずに、線形回帰とk平均法を実現

機械学習のアルゴリズムは難しい…という印象をもたれがちですが、基本的なアルゴリズムは比較的シンプルなので、それほど難しくありません。

そこでこのPart 3では、scikit-learnなどのAIライブラリを使わずに、Pythonで線形回帰とk平均法を実現する方法を説明します。これらのアルゴリズムの実装方法はいくつかありますが、ここでは基本的な仕組みがよくわかるように、簡易的な方法で実装することにします。

線形回帰を実装してみよう

線形回帰の中でも最もシンプルな「単回帰」を実装してみます。

単回帰では、最小二乗法を使って、散布図の各データからの距離の2乗の総和を最小にする直線の式を求めます（図1）。計算上、距離がマイナスの値になることがあるので、すべて0以上の値にするために2乗する点が、最小二乗法の特徴です。

求める直線の式は「回帰式」と呼ばれるもので、次の通りです。

```
y = ax + b
```

つまり、直線の傾き（a）と切片（b）の値を計算することで回帰式を求めることになります。

●総当たりの力技で計算する

ここでは、各データからの距離の2乗の総和を最小にする傾き（min_a）と切片（min_b）を、総当たりで計算する方法で最小二乗法を実装します。あまり使うべきではないとされる3重ループが登場する泥臭い力技になりますが、数学的な概念があまり出て来ないので、わかりやすい方法だと思います。なお、計算の負荷を軽くするために、精度は小数点第1位までとします。

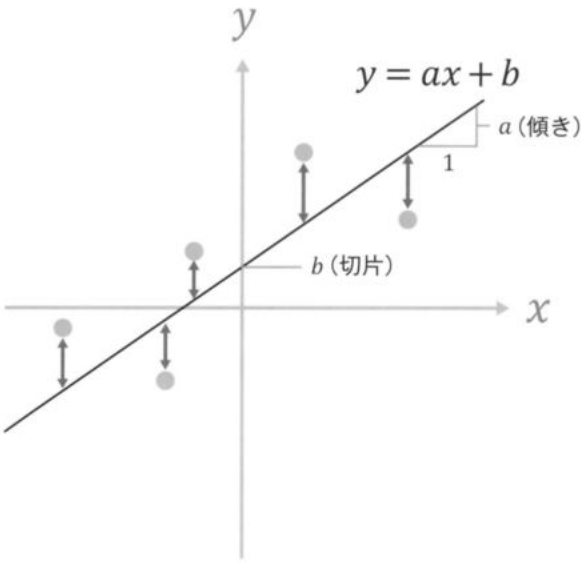
実装したいのは、次のように使える「myFit関数」です。

```
x = [2, 4, 6]
y = [2, 3, 5]

a, b = myFit(x, y)
```

データ（ここでは（2, 2）、（4, 3）、（6, 5））をxとyのリストで用意しておき、これらをmyFit関数の引数に渡すと、傾き（a）と切片（b）を返してくれる、という関数です。

図1●最小二乗法のイメージ



最小二乗法 ＝
●が与えられたとき、↕の長さの「2乗の総和」を最小にする直線を求める手法

↕の長さの2乗の総和は数式で表すと以下の通り

$$\sum_{i=1}^{n}\{y_i-(ax_i+b)\}^2$$

リスト1 ●myFit関数のプログラム

```
def myFit(x, y):
  s = 0    # 各データからの距離の2乗の総和
  min_s = 1000000    # 最小のsを初期化

  # 傾きと切片の取り得る最大値を計算
  res = []
  for i in range(len(x) - 1):
    res.append((y[i+1] - y[i]) / (x[i+1] - x[i]))
  max_a = (sum(res) / len(res))    # 傾きの取り得る最大値
  max_b = sum(y) / len(y)    # 切片の取り得る最大値

  tick_max_a = [i/10 for i in range(int(max_a+1)*10)]
  tick_max_b = [i/10 for i in range(int(max_b+1)*10)]

  # tmp_bを0から0.1ずつmax_bまで変化させる
  for tmp_b in tick_max_b:
    # tmp_aを0から0.1ずつmax_aまで変化させる
    for tmp_a in tick_max_a:
      # 各データからの距離の2乗の総和（s)を求める
      for i in range(len(x)):

        s = s + (y[i] - (tmp_a * x[i] + tmp_b))**2

      if min_s > s :    # sが最小かどうかを判定する
        min_s = s
        min_a = round(tmp_a, 1)
        min_b = round(tmp_b, 1)

      s = 0
    tmp_a = 0

  print("回帰式:y =", min_a, "* x +", min_b)

  return min_a, min_b
```

先に、myFit関数の実装例を示すと、**リスト1**のプログラムになります。中身を解説していきましょう。

冒頭の2行は、利用する変数の初期化です。

```
s = 0    # 各データからの距離の2乗の総和
min_s = 1000000    # 最小のsを初期化
```

変数sは、各データからの距離の2乗の総和を一時的に格納する変数です。変数min_sは、その時点で最小のsを記録するための変数で、十分に大きな値（ここでは1000000）で初期化しておきます。

その後、傾きと切片の取り得る最大値を計算しておきます。ここでは簡易的に、隣接する2つのデータ間の傾きの平均値を、傾きの取り得る最大値（max_a）とします。プログラムでは次のように計算しています。

```
res = []
for i in range(len(x) - 1):
  res.append((y[i+1] - y[i]) / (x[i+1] - x[i]))
max_a = (sum(res) / len(res))    # 傾きの最大値
```

簡単な実装にしているので、ここで「x[i+1] - x[i]」が0になるデータがある場合、すなわち、xで同じ値が連続する場合は、ゼロ除算のエラーが発生するので注意しましょう。これはmyFit関数の制限になります。

切片の取り得る最大値（max_b）は、各データのyの値の平均値とします。

```
max_b = sum(y) / len(y)
# 切片の最大値
```

次に、総当たりの準備をします。0から傾きの取り得る最大値まで0.1ずつ増える数列のリストtick_max_aと、0から切片の取り得る最大値まで0.1ずつ増える数列のリストtick_max_bを作ります。

```
tick_max_a = [i/10 for i in range(int(max_a
+1)*10)]
tick_max_b = [i/10 for i in range(int(max_b
+1)*10)]
```

ここではPythonのリスト内包表記を利用しています。実際には実装を簡単にするため、0から最大値よりも少し大きな値までのリストになっています。

いよいよ総当たりの実行です。3重ループを使って、tick_max_aとtick_max_b、そして引数から得たデータのxとyで総当たりの計算を行い、各データからの距離の2乗の総和が最小になる傾き（min_a）と切片（min_b）を求めます。

```
for tmp_b in tick_max_b:
  for tmp_a in tick_max_a:
    for i in range(len(x)):
```

```
      s = s + (y[i]-(tmp_a*x[i]+tmp_b))**2

    if min_s > s :
      min_s = s
      min_a = round(tmp_a, 1)
      min_b = round(tmp_b, 1)

    s = 0
  tmp_a = 0
```

「(y[i] - (tmp_a * x[i] + tmp_b))**2」が各データからの距離の2乗を計算している部分です。「s = s + (y[i] - (tmp_a * x[i] + tmp_b))**2」がその総和の計算になります。3重ループの中で、変数sの値を最小にするtmp_a(tick_max_aの中の値の1つ)とtmp_b(tick_max_bの中の値の1つ)を調べているのです。

変数sがその時点で最小になったら、min_sにその値を保存すると同時に、そのsでのtmp_aとtmp_bの値をmin_aとmin_bに保存しておきます。

3重ループが終了したら、回帰式を表示して、戻り値でmin_aとmin_bの値を返します。

それでは、リスト1をJupyter Notebookで実行して、myFit関数を定義しましょう。

その後、次のテストデータを使って、動作をチェックします。

```
# テストデータ
x = [6, 7, 8, 10, 12, 15, 19, 20]
y = [7, 3, 5,  6, 10,  3, 10, 12]

a, b = myFit(x, y)
```

実行結果は図2です。「回帰式:y = 0.4 * x + 2.2」と表示されました。ただ、傾きの0.4や切片の2.2という値は本当に正しいのでしょうか。念のため、NumPyを利用して検算してみます。

NumPyのnumpy.polyfit関数を使うと、最小二乗法による単回帰を簡単に行えます。同じテストデータで計算してみましょう。次のプログラムを実行します。

```
import numpy as np

# テストデータ
x = [6, 7, 8, 10, 12, 15, 19, 20]
y = [7, 3, 5,  6, 10,  3, 10, 12]

coe = np.polyfit(x, y, 1)
a = coe[0]
b = coe[1]
print("回帰式:y =", a, "* x +", b)
```

numpy.polyfit関数の第3引数で指定している1は、求める式が1次関数であるという意味です。戻り値はリストで、coe[0]に傾きが、coe[1]に切片が格納されます。

実行結果は図3で、「回帰式:y = 0.389402341343 19157 * x + 2.278496611213796」と表示されました。誤差はありますが、「回帰式:y = 0.4 * x + 2.2」

図2●テストデータでmyFit関数の動作をチェック

```
In [2]: # テストデータ
        x = [6, 7, 8, 10, 12, 15, 19, 20]
        y = [7, 3, 5,  6, 10,  3, 10, 12]

        a, b = myFit(x, y)

        回帰式 : y = 0.4 * x + 2.2
```

図3●numpy.polyfit関数を使って検算する

```
In [3]: import numpy as np

        # テストデータ
        x = [6, 7, 8, 10, 12, 15, 19, 20]
        y = [7, 3, 5,  6, 10,  3, 10, 12]

        coe = np.polyfit(x, y, 1)
        a = coe[0]
        b = coe[1]
        print("回帰式 : y =", a, "* x +", b)

        回帰式 : y = 0.38940234134319157 * x + 2.278496611213796
```

図4●「xとyの共分散」と「xの分散」の式

$$\text{xとyの共分散} = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})$$

$$\text{xの分散} = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2$$

$\bar{x}$ ：xの平均値
$\bar{y}$ ：yの平均値

リスト2●myFit2関数のプログラム

```
def myFit2(x, y):
  n = len(x)              # データの個数
  x_ave = sum(x) / n      # xの平均値を求める
  y_ave = sum(y) / n      # yの平均値を求める

  cov_xy = 0    # xとyの共分散の変数の初期化
  var_x =  0    # xの分散の変数の初期化

  # xとyの共分散と、xの分散を求める
  for xi, yi  in zip(x, y):
    cov_xy = cov_xy + (xi - x_ave) * (yi - y_ave)
    var_x = var_x + (xi - x_ave)**2
  cov_xy = cov_xy / n
  var_x = var_x / n

  a = cov_xy / var_x
  b = y_ave - a * x_ave

  print("回帰式:y =", a, "* x +", b)

  return a, b
```

と近い値になりましたから、myFit関数は正しく動いていると考えられます。

少し現実的なデータを利用しましょう。Part 2で使った「気温とアイスクリームの売り上げの関係」のデータである「ice_sales.csv」を読み込んで、単回帰を行います。

```
import pandas as pd

df = pd.read_csv("ice_sales.csv")
a, b = myFit(df.temp, df.sales)
```

実行すると、「回帰式：y = 0.3 * x + 4.5」と表示されます。

続いて、回帰式を使って予測するための「myPredict関数」を次のように定義します。

```
def myPredict(val, a, b):
  y = a * val + b
  return y
```

では、気温が30度のときのアイスクリームの売り上げを予測してみましょう。

```
sales = myPredict(30, a, b)
print(sales)
```

実行結果は「13.5」万円で、Part 2でscikit-learnを使って求めた予測の値の13.62120947万円と、ほぼ同じになりました。

●統計学を使う実装

参考までに、統計学で出てくる共分散（2種類のデータの関係を表す値）と分散（データの散らばり具合を表す値）を使う最小二乗法の実装例を紹介しておきます。データのxとyがあるとき、回帰式の傾き（a）と切片（b）は、次のように計算できます。

a = xとyの共分散 ÷ xの分散

b = yの平均値 - a * xの平均値

xとyの共分散とxの分散の式は**図4**です。

共分散と分散を使ってaとbを求める「myFit2関数」は**リスト2**のようになります。

では、myFit2関数を使って、気温が30度のときのアイスクリームの売り上げを予測してみましょう。Jupyter Notebookでリスト2を実行したら、続けて次のプログラムを記述します。

```
a, b = myFit2(df.temp, df.sales)
sales = myPredict(30, a, b)
print(sales)
```

実行すると、「13.621209467301526」が出力されます。先ほどのmyFit関数や、Part 2のscikit-learnを使って求めた予測の値とほぼ同じ値になっています。

k平均法を実装してみよう

AIライブラリを使わずに、k平均法によるクラスタリングを実装してみます。

k平均法では、次の2つのステップでグループ分けを行います。

ステップ1　ランダムなグループ分け

k個のクラスタを設定し、それぞれのクラスタの「中心点」をランダムに決めます。次に、各データごとにそれぞれ一番近い中心点を求め、データをそこに所属させることで、各データをk個のグループに分けます。

ステップ2　重心点によるグループ分け

グループごとに「重心点」を求めます。この重心点はグループに属する各データの座標的な平均値で、中心点とは異なる点です。その後、重心点を新たな中心点とし、各データを一番近い中心点に所属させます。この処理を、重心点の位置が変化しなくなるまで"繰り返し"ます。

ステップ1で乱数を使うので、同じデータでもグループ分けの結果が毎回少し異なる可能性がある点が、k平均法の特徴です。

各データが属する中心点は、データと中心点の"距離"を計算することで決めます。距離にはいくつかの種類がありますが、ここではユークリッド距離を使います。2点間のユークリッド距離は、いわゆる直線距離のことで、図5のように三平方の定理で簡単に計算できます。

今回はプログラムを簡単にするために、k=2にしたいと思います。つまり、データを2つのグループに分けます。また、ステップ2での"繰り返し"は重心点の位置が変化しなくなるまでではなく、3回に留める簡易的な実装にします。

●グループに分けるcalcDist関数を作る

それでは実装を始めます。初めに、2つあるグループの中心点のどちらに各データが近いのかを判定し、2つのグループに分ける「calcDist関数」を定義します。これはリスト3のようになります。簡単に解説しましょう。

calcDist関数は次の4つの引数を持ちます。

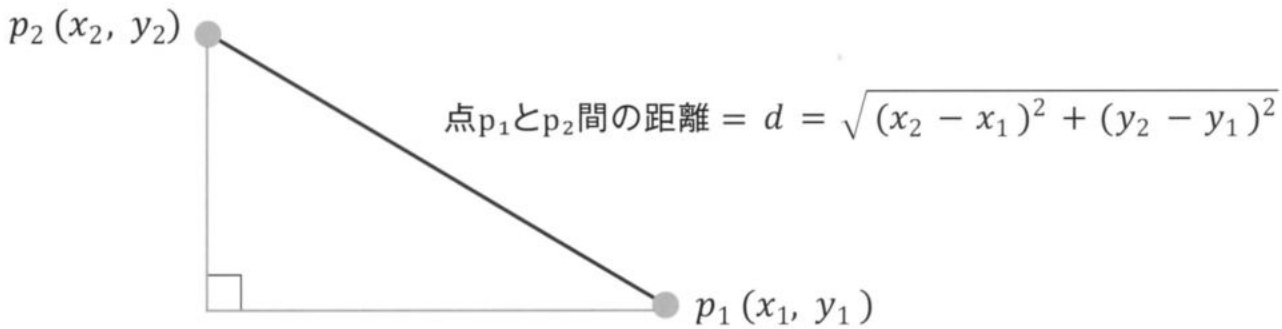
```
x   : 各データのx座標のリスト
y   : 各データのy座標のリスト
cxs : 中心点のx座標のリスト。要素数は2
cys : 中心点のy座標のリスト。要素数は2
```

各データがどのグループに属するのかは辞書で管理することにします。そこで、データを格納したリストのインデックスをキーとする辞書groupsを作ります。

```
groups = {}
groups = groups.fromkeys(list(range(len(x))
))
```

k=2にするので、2つの中心点の座標は(cxs[0], cys[0])と(cxs[1], cys[1])です。これらの中心点の座標と各データの座標を使ってユークリッド距離を計算し、どちらの中心点に近いのかを判定します。

図5●2点間のユークリッド距離は三平方の定理を使って計算できる



```
for i in range(len(x)):
  d1 = (cxs[0]-x[i])**2 + (cys[0]-y[i])**2
  d2 = (cxs[1]-x[i])**2 + (cys[1]-y[i])**2

  if d1 < d2:
    groups[i] = 0
  elif d1 > d2:
    groups[i] = 1
```

中心点(cxs[0], cys[0])に近い場合はグループ0、中心点(cxs[1], cys[1])に近い場合はグループ1とします。0または1の値を辞書groupsに設定します。なお、ここでは2つの距離を比較することだけが目的で、距離の値そのものは必要ありません。よって、正確な距離の値を出すための平方根の計算は省略しています。

最後に、辞書groupsを戻り値として返して、calcDist関数は終了します。

Jupyter Notebookでリスト3を実行したら、次のテストデータを使うプログラムでcalcDist関数をテストしましょう。

リスト3●calcDist関数のプログラム

```
def calcDist(x, y, cxs, cys):
  groups = {}    # グループ分け結果を格納する辞書
  # データのインデックスをキーにする
  groups = groups.fromkeys(list(range(len(x))))

  # 各データと中心点のユークリッド距離を求める
  for i in range(len(x)):
    d1 = (cxs[0] - x[i])**2 + (cys[0] - y[i])**2
    d2 = (cxs[1] - x[i])**2 + (cys[1] - y[i])**2

    # どちらの中心点（グループ）に近いかを判定
    if d1 < d2:
      groups[i] = 0
    elif d1 > d2:
      groups[i] = 1

  return groups
```

```
# テストデータ
x = [1.5, 3.1, 4.3, 5.6, 6]
y = [4.1, 3,   4.3, 5.2, 6]
cxs = [2, 4]   # 中心点0
cys = [4, 4]   # 中心点1
groups = calcDist(x, y, cxs, cys)
print("グループ分けの結果は", groups.items())
```

このプログラムでは、中心点の座標を(2, 4)と(4, 4)にしています。

実行すると、**図6**のように表示されます。タプルの1つ目の要素はテストデータのインデックスで、2つ目の要素が属するグループの値(0か1)です。テストデータの(1.5, 4.1)はグループ0に、(3.1, 3)と(4.3, 4.3)と(5.6, 5.2)と(6, 6)はグループ1に属するという結果になっています。

●クラスタリングを行うmyFit関数を作る

作成したcalcDist関数を使ってクラスタリングを行う、myFit関数を実装しましょう。これは**リスト4**のようになります。引数のxにはデータのx座標のリストを、引数のyにはデータのy座標のリストを与えます。

myFit関数では最初に、random.randint関数を使って、2つのクラスタの中心点となるデータをランダムに決めます。

```
c0, c1 = 0, 0
while True:
  c0 = random.randint(0, le
n(x) - 1)
  c1 = random.randint(0, le
n(x) - 1)
  if c1 != c0: break
cxs = [x[c0], x[c1]]
cys = [y[c0], y[c1]]
```

図6● テストデータでcalcDist関数をテストする

```
In [2]: # テストデータ
        x = [1.5, 3.1, 4.3, 5.6, 6]
        y = [4.1, 3,   4.3, 5.2, 6]
        cxs = [2, 4]   # 中心点0
        cys = [4, 4]   # 中心点1
        groups = calcDist(x, y, cxs, cys)
        print("グループ分けの結果は", groups.items())

        グループ分けの結果は dict_items([(0, 0), (1, 1), (2, 1), (3, 1), (4, 1)])
```

2つの中心点が同じにならないように、念のためif文でチェックしています。

続いて、calcDist関数で各データを2つのグループに分けます。

その後、forループを用意して、次の4つのリストを作成します。

```
g0x : グループ0のデータのx座標のリスト
g1x : グループ1のデータのx座標のリスト
g0y : グループ0のデータのy座標のリスト
g1y : グループ1のデータのy座標のリスト
```

これらのリストを使って、それぞれのグループの重心点を計算し、新しい中心点に設定します。

```
cxs = [sum(g0x)/len(g0x), sum(
g1x)/len(g1x)]
cys = [sum(g0y)/len(g0y), sum(
g1y)/len(g1y)]
```

そして、新しい中心点で再度、calcDist関数を実行します。

```
groups = calcDist(x, y, cxs, c
ys)
```

この一連の処理(リスト4の(1))をforループで3回繰り返して、k平均法によるクラスタリングを終了します。最後に、戻り値として変数groupsを返したら、myFit関数は完了です。

Jupyter Notebookでリスト4を実行したら、テストデータでmyFit関数を試してみましょう。

```
# テストデータ
x = [1.5, 3.1, 4.3, 5.6, 6]
y = [4.1, 3,   4.3, 5.2, 6]
groups = myFit(x, y)
print("グループ分けの結果は", groups.items())
```

実行すると、図7の結果になります。テストデータの(1.5, 4.1)と(3.1, 3)はグループ0、(4.3, 4.3)と(5.6,

リスト4●クラスタリングを行うmyFit関数のプログラム

```
import random

def myFit(x, y):
  # クラスタの中心点をランダムに2つ設定
  c0, c1 = 0, 0
  while True:
    c0 = random.randint(0, len(x) - 1)
    c1 = random.randint(0, len(x) - 1)
    if c1 != c0: break
  cxs = [x[c0], x[c1]]
  cys = [y[c0], y[c1]]

  # 各データを中心点でグループ分け
  groups = calcDist(x, y, cxs, cys)

  # 重心点を繰り返し再計算
  for i in range(3):
    g0x, g1x, g0y, g1y = [], [], [], []
    # グループごとのリストを作る
    for i in range(len(groups)):
      if groups[i] == 1:
        g0x.append(x[i])
        g0y.append(y[i])
      else:
        g1x.append(x[i])
        g1y.append(y[i])

    # 重心点を中心点にする
    cxs = [sum(g0x)/len(g0x), sum(g1x)/len(g1x)]
    cys = [sum(g0y)/len(g0y), sum(g1y)/len(g1y)]

    # 中心点でグループ分け
    groups = calcDist(x, y, cxs, cys)

  return groups
```

(1)

図7●テストデータでmyFit関数をテストする

```
In [4]: # テストデータ
        x = [1.5, 3.1, 4.3, 5.6, 6]
        y = [4.1, 3,   4.3, 5.2, 6]
        groups = myFit(x, y)
        print("グループ分けの結果は", groups.items())

        グループ分けの結果は dict_items([(0, 0), (1, 0), (2, 1), (3, 1), (4, 1)])
```

5.2)と(6, 6)はグループ1になりました。なお、k平均法は乱数を使うので、結果は実行のたびに少し変わります。

● 人口と出生率の関係のデータをクラスタリング

作成したmyFit関数を使って、Part 2で出てきた「都道府県ごとの人口と出生率の関係」のデータ(population.csv)を、2つのグループに分けてみます。

プログラムはリスト5です。グラフを描くためのコードが何行かありますが、population.csvを読み込んでk平均法を行っているのは次の2行の部分だけです。

```
df = pd.read_csv("population.csv")
groups = myFit(df.population, df.rate)
```

Jupyter Notebookでリスト5を実行しましょう。すると、図8のようなグラフが表示されます。47都道府県を赤丸のグループと青丸のグループに分けることができました。分け方はPart 2での結果と同じで、次のようになっています。

- 出生率が低く人口が多いグループ(東京、大阪など。図8の赤丸)
- 出生率が高く人口が少ないグループ(沖縄、宮崎など。図8の青丸)

k平均法は乱数を使うので、どちらのグループが赤丸(青丸)になるのかは決まっていません。実行のたびに各グループの丸の色は変わります。

☆　☆　☆

以上のように、AIライブラリを使わずに、機械学習のアルゴリズムをゼロから実装すると、より深くアルゴリズムを理解できるようになります。この記事を機会に、いろいろなアルゴリズムの実装に挑戦してみてください。

リスト5●「都道府県ごとの人口と出生率の関係」のデータ(population.csv)を読み込んで、2つのグループに分けるプログラム

```
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_csv("population.csv")
groups = myFit(df.population, df.rate)

x = df.population
y = df.rate
df["group"] = groups.values()
plt.rcParams["figure.figsize"] = (10, 10)
df["color"] = df.group.map({0:"blue", 1:"red"})
plt.scatter(x, y, c=df.color)

for i in range(len(x)):
  plt.text(x[i], y[i], df.name[i], fontsize=16)
```

図8●リスト5の実行例