Python3 応用

（機械学習）

－　目次　－

Self-Actualization for You SAY
セイ・コンサルティング・グループ

１．機械学習とディープラーニング

機械学習とディープラーニングの説明の前に、人工知能（AI:Artifical Intelligence）について触れます。

人工知能と聞くと「鉄腕アトム」のように人間の姿をして人間のように考えて振る舞うロボットを想像する人が多いと思います。
映画やアニメなどではそのように描かれてきたからでしょう。

人工知能研究の第一人者であるマッカーシー教授は人工知能学会で「人工知能」を次のように説明しています。

「知的な機械、特に知的なコンピュータプログラムを作る科学と技術のこと。人の知能を理解するためにコンピュータを使うことと関係がありますが、自然界の生物が行っている知的手段だけに研究対象を限定することはありません。」

非常に抽象的な表現ですが、これは「知能」という定義そのものが幅広いため人工知能の解釈もおのずと広くなってしまいます。

Wikipedia では知能の解説の冒頭で以下のように表現しています。
「知能は、論理的に考える、計画を立てる、問題解決する、抽象的に考える、考えを把握する、言語機能、学習機能などさまざまな知的活動を含む心の特性のことである。知能は、しばしば幅広い概念も含めて捉えられるが、心理学領域では一般に、創造性、性格、知恵などとは分けて考えられている。」

１．１ 機械学習とは

機械学習（Machine Learning）とは、人工知能における研究分野の一つです。人間の脳が自然に行なっている「学習」の能力を、コンピュータを用いて実現しようとする手法のことです。

具体的には、ある程度の量のサンプルデータを入力して解析を行います。
解析によってデータの中に一定の法則や特徴、ルールがないかを探します。
そして、見つけた法則や特徴、ルールを元にして、別のデータを分類したり、今後を予測したりします。

最近では、こうした機械学習を応用して、文字認識や音声認識、囲碁や将棋などのゲーム戦略、医療診断やロボット開発などが行われています。

1．2 ディープラーニングとは

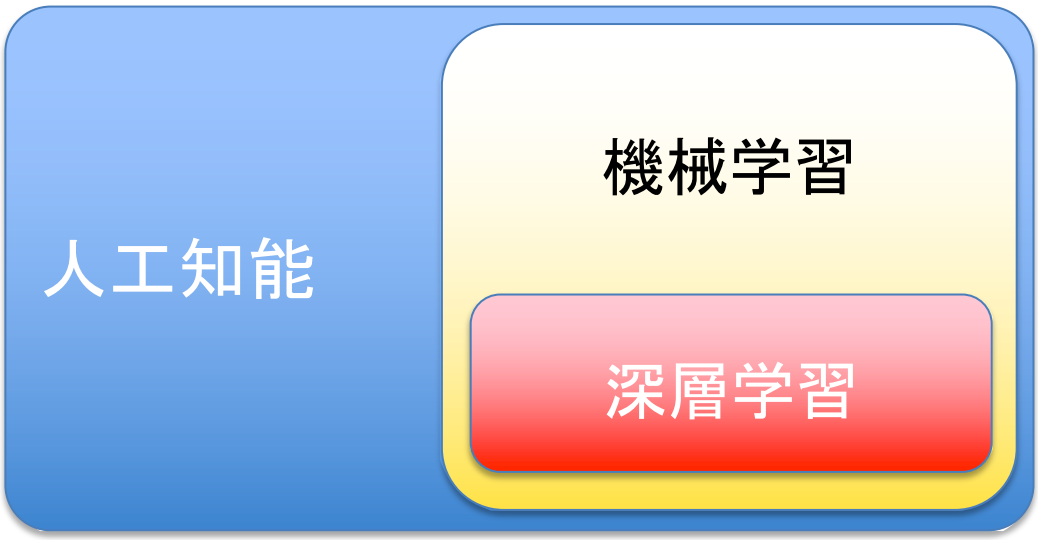
深層学習（Deep Learning）は機械学習の一分野です。
多層構造のニューラルネットワークを用いた機械学習のことです。
近年、深層学習が大きく注目を集めるようになったのは何故でしょうか。

深層学習が注目されたきっかけは、2012 年に開催された画像認識コンテスト「ILSVRC（ImageNet Large Scale Visual Recognition Competition）」であると言われています。この大会は 2010 年から毎年行われていて、大規模画像の認識・分類をソフトウェアで行わせます。
2012 年のコンテストで、カナダ・トロント大学のジェフリー・ヒントン教授が率いるチームが、深層学習を利用したシステムで、驚異的な精度を示して優勝したことがきっかけとなりました。

翌年のコンテストでも、深層学習を利用したものが上位を占めました。ちなみに 2014 年には Google が優勝しました。

さらに深層学習は、画像分野・音声認識の分野だけでなく、自然言語処理など、さまざまな分野で活用されはじめ、大きな成果を上げてきました。

実はその概念や手法は 1980 年代からあったのですが、コンピュータが高性能になったことや、ビジネス分野で大きく成功を収めたこともあり。今日のように注目されるようになりました。



１．３ 機械学習の分類

機械学習にはさまざまな分類方法がありますが、「教師あり学習（Supervised Learning）」「教師なし学習（Unsupervised Learning）」「強化学習」という 3 種類に分ける考え方があります。

教師あり学習

「教師あり学習」は、学習に使用するデータに正解が与えられ、その正解を元にして学習する手法です。

例えば、何枚かの動物の写真があり、それぞれの写真には「犬」「猫」「猿」のように正解ラベルが付けられていれば、これが学習のお手本となるデータ、すなわち「訓練データ」になります。

教師あり学習では、このような画像（または音声、テキスト）と正解ラベルが対になったデータを学習し、未知の画像が入力されたときにその画像を識別する、といった使い方が一般的です。

学習するための手段として使われるのが「分類器」と呼ばれる仕組みで、別名「モデル」とも呼ばれます。モデルは内部に「パラメータ」と呼ばれる動的変更が可能な値をいくつも持ち、学習することによってパラメータの値を「最適な値」に変えていきます。

「犬」の画像を入力して、「これは犬である」と出力するようにパラメータの値を学習するのですが、これは犬の画像を何パターンも入力することになります。そうすることで犬の画像を識別できるようにパラメータの値が更新されていきます。うまく学習できれば、いろんなタイプの犬の画像を入力しても常に「犬」と出力されるようになります。

これは教師あり学習の一例ですが、教師あり学習はこのような「識別」や「分類」という問題のために使われます。

教師あり学習は、さらに正解ラベルの種類によって「回帰」（regression）「分類」（classification）に分けられます。

教師なし学習

「教師なし学習」は、正解ラベルのないデータに対して、特徴や共通点、パターンなどを発見する手法です。

正解ラベルがないので、生のデータをそのまま訓練データとして使用することができ手間は少なくなりますが、どのような特徴やパターンが導き出されるのかは、やってみなければわからないという問題があります。

代表的な「クラスタリング」はデータの特徴を見出して似ているものどうしをグループ分けします。近年、膨大な顧客データ（ビッグデータ）を分析し、消費動向別にグループ分けする手段としてクラスタリングが活用されています。

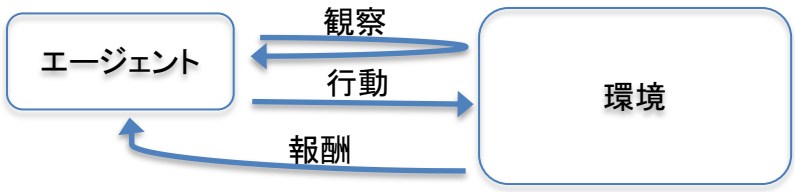
教師なし学習の手法は「クラスタリング」（clustering）や「次元削減」（dimensionality reduction）などがあります。

強化学習

「強化学習」とは、試行錯誤を通じて「価値を最大化するような行動」を学習する手法です。

教師あり学習とよく似ていますが、訓練データに完全な答えが与えられないという点が異なります。

強化学習では、エージェント（行動の主体）と環境（状況や状態）が登場します。エージェントは環境を観察し、それに基づいて意思決定を行い行動します。すると環境が変化し、エージェントに何らかの報酬が与えられます。エージェントは、よりたくさんの報酬が得られる、より良い行動を学習していきます。

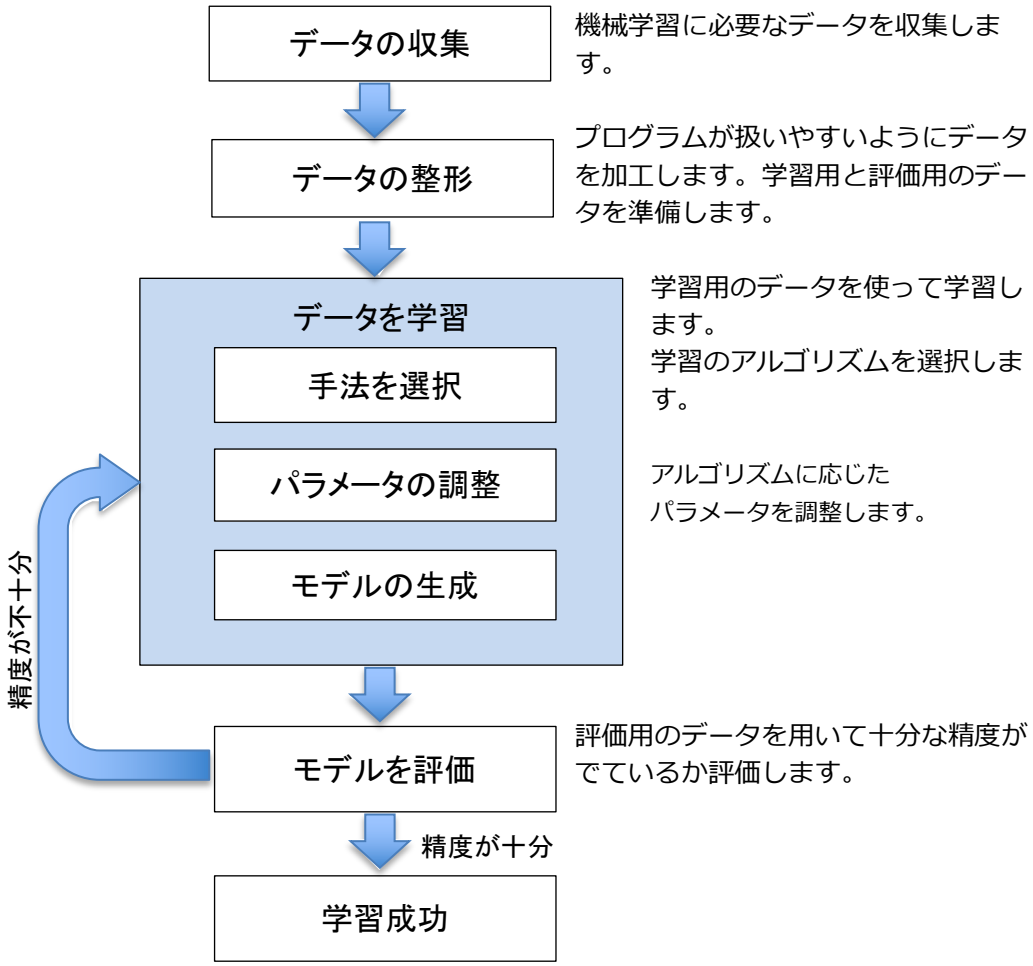


例えば、株の売買により利益を得る問題が強化学習にあたります。

この場合、持っている株をすべて売り出せば確かにその時点では最もキャッシュを得ることができますが、より長期的な意味での価値を最大化するには、株をもう少し手元に置いておいたほうが良いかもしれません。例えば「強化学習」と呼ばれる手法では、行動した結果に応じて報酬が与えられるようにすることで、徐々により良い行動へと進むように学習します。

１．４ 機械学習の流れ

機械学習は以下の手順の反復で行います。



学習モデル

機械学習の目的は、データの特徴から、予測するための「モデル」を作ることです。そのモデルを「学習モデル」と呼びます。

そして、例えば猫の画像認識ができる学習モデルができたあとに、その学習モデルに学習時の画像とは異なる画像を与えることで、その画像に猫の特徴の何パーセント含まれているのかがわかります。

過学習（過剰適合）

過学習（overfitting）とは、訓練データによる学習が過剰に適合しすぎることにより、未知のデータに適合できない状態を指します。

過学習が起きる理由は、「データが少なすぎる（あるいは偏っている）」「モデルに対して問題が複雑すぎる」ことで、前者の抜本的な解決策は、教師データの件数を増やすことです。後者の問題も含め、より適切なモデルや手法の選択を考える必要があります。

１．５ 教師あり学習のアルゴリズム

教師あり学習は「回帰」と「分類」に分けることができます。
それでは、「回帰」と「分類」の違いから説明します。

回帰（regression）

回帰とは過去のデータを元に、将来の数値を「予測」するのに利用します。
回帰により得られる結果は実際の値です。
販売予測や株価の変動、機器の異常検知などを予測することができます。

分類（classification）

与えられたデータにラベルをつけて「分類」することができます。
分類により得られる結果はカテゴリです。
例えば迷惑メールの分類や、手書き文字の認識、クレジットカードの不正検知などの判別を行うことができます。

よく利用される教師あり学習アルゴリズムとしては
・線形回帰（回帰）
・ランダムフォレスト（分類、回帰)
・ロジスティック回帰（分類、回帰）
・k 近傍法（分類）
が挙げられます。

線形回帰

線形回帰（linear regression）とは、統計学における回帰分析の一種です。
データを学習し、「直線」で表される式を使って未知の値を予測します。
線形回帰の式は、目的変数 Y と説明変数 X および係数 a と切片 b により構成される数式をモデル化したものです。

$$Y=a_1X_1+a_2X_2+\cdots\cdots+b$$

この式の係数や切片を導き出す方法として最小二乗法などがあります。

ランダムフォレスト

ランダムフォレストは決定木を複数組み合わせて、各決定木の予測結果を多数決することによって結果を得ます。
決定木は親から順に条件分岐を辿っていくことで、結果を得る手法です。

ロジスティック回帰

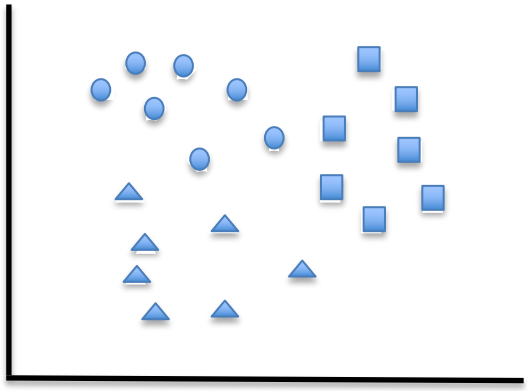
ロジスティック回帰は、クラスを分類する判断に「０」か「１」（正か負）ではなく、「確率」を用います。例えば、与えられたデータがクラス１である確率は 80%、そうでない確率は 20%のように判断します。つまり、たとえ可能性があったとしても、確率が低い場合は決定境界の重みベクトルを修正できます。

k 近傍法

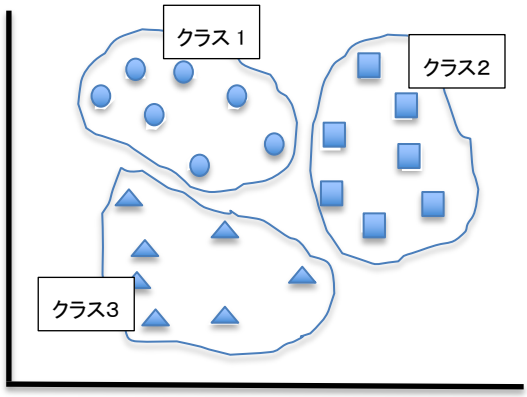
k 近傍法は、単純でわかりやすいアルゴリズムです。

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。

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。

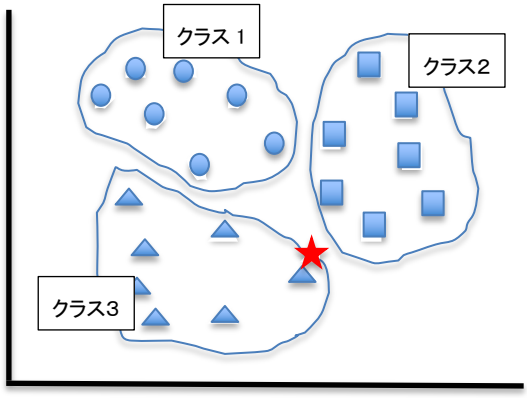


次に、与えられたデータを特徴ベクトルとしてそれぞれ近い場所にある正解ラベルのデータを使ってクラスに分類します。



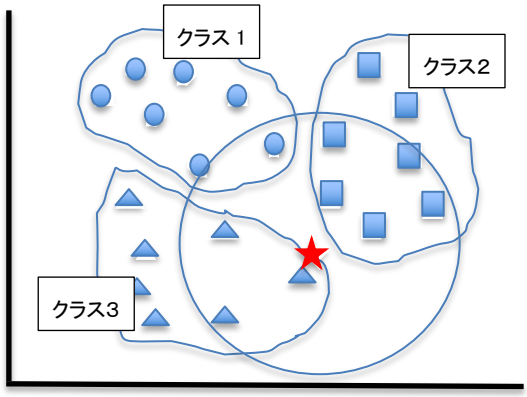
これが k 近傍法の土台となる学習モデルになります。

次に予測したいデータを学習モデルに与えます。



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９とすると、予測したいデータはクラス２に分類されます。



１．６ 教師なし学習のアルゴリズム

教師なし学習は「クラスタリング」や「次元削減」などがあります。

クラスタリング（clustering）

クラスタリングはデータの類似性を元にしてデータを複数のグループに分ける手法です。

階層的にデータをグループに分類する「階層型」と、特定のグループ数に分類する「非階層型」があり、階層型のクラスタリングには「ウォード法」、非階層型クラスタリングには「k 平均法」などのアルゴリズムがあります。

例えば、ユーザをその嗜好によってグループ分けを行い、それによって、ユーザーの嗜好に合ったダイレクトメールの送信などに活用できます。

次元削減（dimensionality reduction）

データの次元を削減して、固有の構造を見つけ出す手法です。データの特徴を維持しつつ、データ量を減らすことができます。

これは、ある種の情報を保持しつつ、高次元のデータを低次元のデータに変換することを言います。

データの可視化や、構造抽出、計算の高速化・メモリ節約などの用途に利用できます。

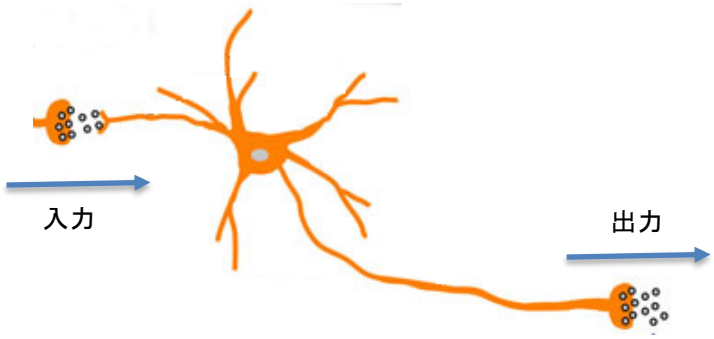
1．7 ニューラルネットワーク

ここまでは機械学習を、「教師あり学習」「教師なし学習」という分け方で解説してきましたが、その両方の学習ができ、ここまで解説してきた統計学的アプローチとは違うニューラルネットワークについて解説します。

ニューラルネットワークというのは、人の神経を模したネットワーク構造のことです。

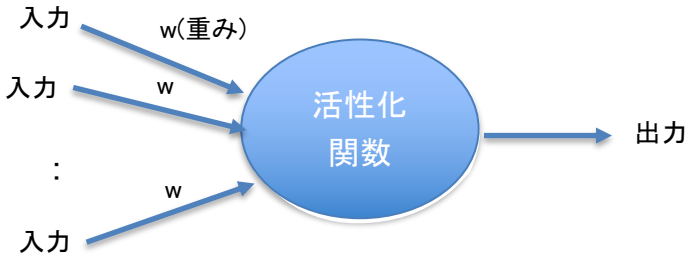
人間の脳の中には、多数の神経細胞（ニューロン）が存在しています。一つのニューロンは、他の複数のニューロンから信号を受け取り、他のニューロンに対して信号を渡します。脳はこのような信号の流れによって、さまざまな情報を伝達します。

このしくみをコンピュータで再現したのがニューラルネットワークです。



形式ニューロン

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」と呼ばれるプログラムです。



形式ニューロンは「入力（１か０）×重み」の総和が閾値を超えていると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。

このように学習データをクラス１かクラス２（１か０）のどちらかに分類する分類方法を「２クラス分類」（２値分類）と言います。

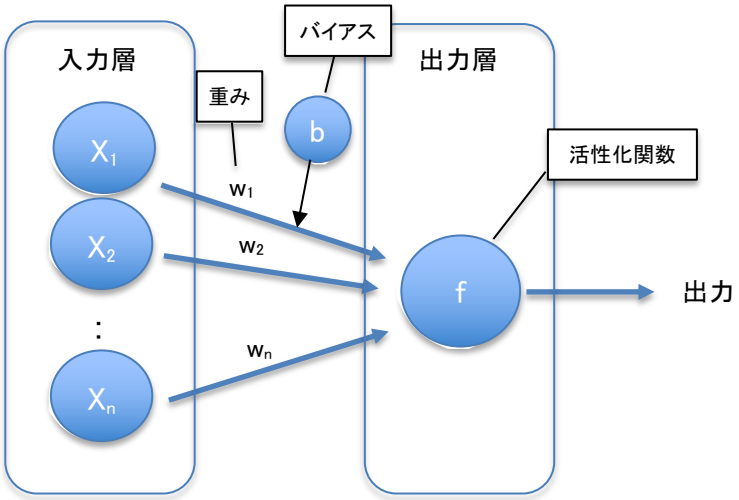
活性化関数

活性化関数は、入力信号の総和がいくつになったら出力するのかを決める関数です。結果ｔとして１か０を出力するので、一般的に活性化関数には「ステップ関数」が使われます。

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。



単純パーセプトロンは、入力に対する出力が教師データと異なる場合、「重み」を変更してから次の入力へ進みます。もし、出力が教師データと一致した場合、重みはそのままです。これを繰り返して最適な重みを見つけるのが単純パーセプトロンの学習です。

バイアス

単純パーセプトロンには、入力とひも付く「重み」がありますが、実際の計算では重みに「バイアス」という値を加えて、重みを特定の方向に偏らせることができます。

多層パーセプトロン（ニューラルネットワーク）

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ります。このような形態を「多層パーセプトロン」と呼びます。

多層パーセプトロンでは、入力部を「入力層」、出力部を「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。

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。

深層学習（ディープラーニング）

ニューラルネットを３層以上重ねたものを一般的に「ディープ・ニューラル・ネットワーク（DNN）」と呼びます。畳み込みネットワーク、RNN（再帰型ニューラルネットワーク）なども含まれます。

この DNN を使った機械学習が深層学習（ディープラーニング）です。深層学習では、大量のデータを学習することで、各ニューロン間の接続の重み付けのパラメーターを繰り返し調整します。

2．機械学習のための Python 環境

Python には機械学習のためのモジュールが提供されています。これらのモジュールを呼び出すためと、機械学習の流れのところでも説明していますが、訓練データや評価データを準備するところでも Python の知識が必要になります。

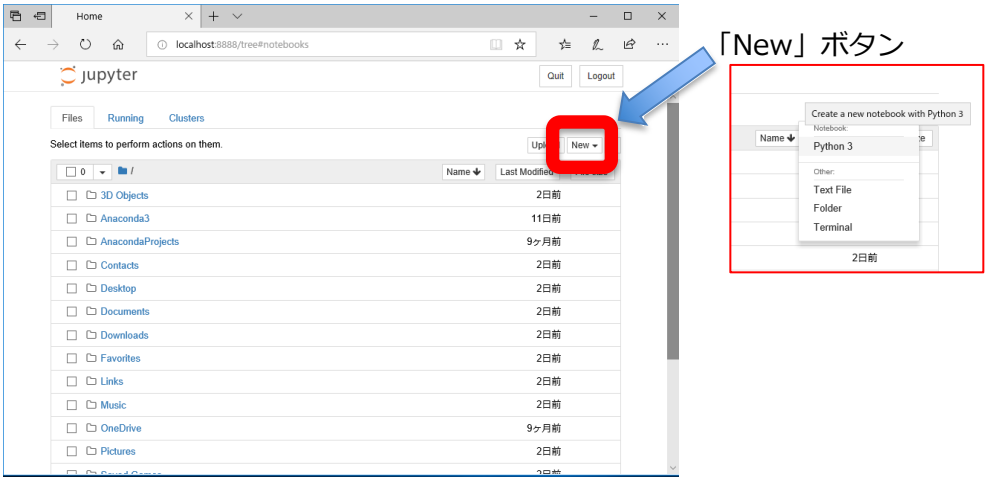
本研修では機械学習に必要なモジュールが多く組み込まれている Anaconda を使用して Python を学習します。

2．1 Jupyter Notebook の使い方

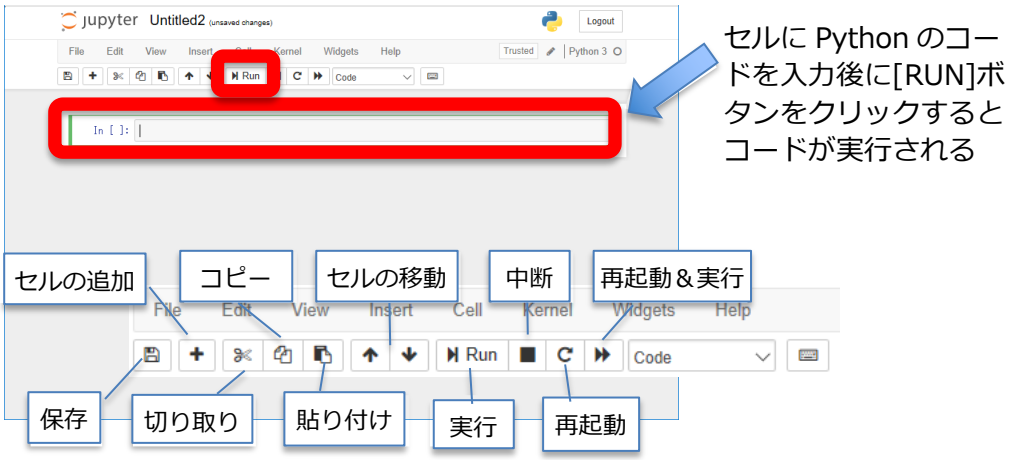
Juypter Notebook とはブラウザから Python のプログラムを実行し、その結果を記録しながらデータ分析作業をするためのツールです。

機械学習の解説では、データや Python のプログラムを実行して予測結果をグラフで表示させることが多くあります。
Jupyter Notebook は Python のコードを記述して実行することでグラフを表示してくれる機能を持っています。
Jupyter Notebook は Anaconda にデフォルトでインストールされています。

Winodws で Jupyter Notebook を起動するには、スタートメニューの「Anaconda3」のフォルダにある「Jupyter Notebook」を選択します。



右上の「New」ボタンをクリックして「Python3」を選択します。



２．２ NumPy モジュール

NumPy は、Python の数値計算のためのモジュールで、高速に数値計算ができることが特徴です。

Python は柔軟に素早く書くことができる言語として知られていますが、スクリプト言語であるため処理速度が Java や C に比べて遅いというデメリットがあります。そこで、演算処理を C で書かれたバイナリモジュールとして組み入れ Python から呼び出すことを出来るようにしたライブラリが NumPy です。NumPy を導入することで高速な数値演算ができます。

NumPy は Python の標準ライブラリにはなく、外部ライブラリのモジュールなのですが、Anaconda にはデフォルトでインストールされています。

NumPy がインストールされているか確認しましょう。
NumPy モジュールを np という名前でインポートしてみます。

```
import numpy as np
```

NumPy で配列を操作

NumPy で使われる主なクラスは np.ndarray と呼ばれる多次元を扱う配列です。
例えば、要素が 5,7,9 の配列は次のように作ります。

```
arr = np.array([5,7,9])
```

２次元配列なら次のようにします。

```
arr = np.array([[5,7,9],[1,2,3]])
```

NumPy の配列は numpy.ndarray クラスのオブジェクトなので、高速に動作する上、便利な関数や属性が多く用意されています。
例えば、各次元の要素が知りたい場合には、shape 属性を参照します。

```
arr.shape
```

次のようにすると０～１のランダムな値で、要素を初期化することができます。

```
arr = np.random.rand(2,3)
```

ベクトルと行列の演算

配列をベクトルや行列として計算することができます。
例えば、次のような配列を用意して５を掛けるとそれぞれの要素が５倍になります。

```
arr = np.array([1,2,3]) * 5
```

同じ形の配列同士の計算は、同じ場所の要素同士が計算されて返ります。

```
arr1 = np.array([[1,2,3],[4,5,6]])
arr2 = np.array([[1,4,6],[2,3,4]])
arr = arr1 + arr2
```

転置を行うには、配列のＴ属性を参照します。

```
arr3 = arr1.T
```

ベクトルの内積・行列の積

dot 関数を使うと、ベクトルの内積や行列の積を求めることができます。

【書式】numpy.dot(a, b, out = None)

パラメタ名	概要
a	左からかけるベクトルまたは行列（２次元配列）
b	右からかけるベクトルまたは行列（２次元配列）
out	結果を代入する配列
戻り値	ベクトルの内積の結果や、行列の積の結果

ベクトルの内積とは、各要素の積を全て足し合わせた値です。
例えば以下のようなベクトルの内積は１×４＋２×５＋３×６で３２です。

```
arr1 = np.array([1,2,3])
arr2 = np.array([4,5,6])
np.dot(arr1,arr2)
```

行列の積では、横の並び、縦の並びの組の同じ順番の数同士を掛けたものを足します。
例えば以下のような行列の積は、[１×５＋２×７，１×６＋２×８]，[３×５＋４×７，３×６＋４×８]で[１９，２２]，[４３，５０]になります。

```
arr1 = np.array([[1,2],[3,4]])
arr2 = np.array([[5,6],[7,8]])
np.dot(arr1,arr2)
```

配列要素の平均

配列の要素の平均は mean 関数を使うと簡単にできます。

【書式】numpy.mean(a, axis = None, dtype = None, out = None, keepdims =False)

パラメタ名	概要
a	平均を求めたい配列
axis	どの軸(axis)に沿って平均を求めるか
dtype	平均を求める際に使用するデータ型
out	計算結果を格納するための配列
keepdims	返す配列の軸(axis)の数をそのままにするかどうか
戻り値	指定した配列の要素の平均、もしくは平均を要素とする配列

０～９までのランダムな整数の配列を生成して、その平均を計算してみます。
ランダムな値の配列は、random.randint 関数を使って作ります。
ramdint 関数の第一引数は下限、第二引数は上限（この値は含まない）、第三引数は要素数です。

```
r = np.random.randint(0,10,10)
```

この配列をすべての要素の平均は次のようになります。

```
m = np.mean(r)
```

標準偏差

標準偏差は、std 関数で求めることができます。

【書式】numpy.std(a, axis=None, dtype=None, out=None, ddof=0, keepdims=False)

パラメタ名	概要
a	標準偏差を計算したい配列
axis	どの軸方向に沿って演算を進めていくか
dtype	演算を行うときに用いるデータ型
out	結果を格納する配列
ddof	データ個数 N ではなく"N-ddof"で割る
keepdims	True にすると出力される配列の次元数が保存される
戻り値	指定された範囲での標準偏差を要素とする配列、または値

先ほどの配列 r の標準偏差は、次のようになります。

```
s = np.std(r)
```

集計

集計する関数として、sum 関数があります。

【書式】numpy.sum(a, axis=None, dtype=None, out=None, keepdims=None)

パラメタ名	概要
a	和を求めたい配列
axis	どの軸方向に要素を足し合わせていくか
dtype	出力する値のデータ型と、計算を行う際に用いるデータ型
out	結果を格納する配列
keepdims	True にすると出力される配列の次元数が保存される
戻り値	和を要素とする配列

この sum 関数には axis(軸)を表す引数があり、２次元配列に関してどの向きに集計をとるのか指定できます。

```
arr = np.array([[1,2,3],[2,3,4]])
np.sum(arr,axis=0)

np.sum(arr,axis = 1)
```

ベクトルの長さ

２点間の距離（ベクトルの長さ）、いわゆる「ユークリッド距離」を求める場合は、linalg.norm 関数を使います。

【書式】numpy.linalg.norm(x, ord=None, axis=None)

パラメタ名	概要
x	長さを調べたい配列
ord	ノルム（対象）の次元数
axis	どの次元方向に和を取るか
戻り値	長さを要素とする配列

座標（x,y）として、ａ地点（2,5）からｂ地点（7,8）へユークリッド距離を求めるには次のように記述します。

```
a = np.array([2,5])
b = np.array([7,8])
np.linalg.norm(b-a)
```

2．3 Pandas モジュール

Pandas は Python でデータ分析、解析を支援するライブラリです。
機械学習では、データを Pandas の「DataFrame」に変換して、操作したり表示したりすることが多いので、ここでは DataFrame を扱います。

DataFrame は RDB（関係データベース）のテーブルや CSV、Excel のテーブルのような形式のデータ構造です。

DataFrame を作る

NumPy と Pandas をインポートします。

```
import numpy as np
import pandas as pd
```

DataFrame のコンストラクタの引数でリストを渡して、DataFrame オブジェクトを作ります。
リストは、名前、国籍、誕生日の要素で生成しています。

```
df = pd.DataFrame([["Edison","USA","2/11"],
                   ["Hiraga","Japan","12/18"],
                   ["Einstein","Germany","3/14"]])
```

さらに、表の列名を columns 関数でつけます。

```
df.columns = ["Name","Country","Birthday"]
```

表の行名は index 属性にリストをセットします。

```
df.index = [1,2,3]
```

作成した DataFrame を表示すると、表形式になっていることがわかります。

```
print(df)
```

	Name	Country	Birthday
1	Edison	USA	2/11
2	Hiraga	Japan	12/18
3	Einstein	Germany	3/14

DataFrame を CSV ファイルにする

Pandas の DataFrame は、CSV ファイルにエクスポートしたり、CSV ファイルをインポートしたりできます。

CSV ファイルへエクスポートするには、to_csv 関数を使います。

【書式】DataFrame.to_csv(path_or_buf=None, sep=', ',
float_format=None, columns=None,
header=True, index=True, encoding=None,
compression=None, quotechar='"',
line_terminator='¥n', date_format=None,
escapechar=None, decimal='.')

パラメタ名	概要
path_or_buf	出力するファイル名（省略した場合は、文字列として出力）
sep	区切り文字
float_format	浮動小数点数の書式文字列
columns	出力する列名
header	列名を保存するか否か
index	行名を保存するか否か
encoding	出力文字コード（'utf-8','shift_jis','euc_jp','ascii'など）
compression	ファイルの圧縮（'gzip','bz2','zip','xz'）
quotechar	引用文字
line_terminator	改行文字
date_format	日時の書式文字列
escapechar	エスケープ文字
decimal	小数点の記号
戻り値	長さを要素とする配列

temp.csv という名のファイル名で保存します。

```
df.to_csv("temp.csv")
```

実行すると、temp.csv というファイルができます。開くと、CSV ファイルになっています。

```
,Name,Country,Birthday
1,Edison,USA,2/11
2,Hiraga,Japan,12/18
3,Einstein,Germany,3/14
```

CSV ファイルから DataFrame を作る

CSV ファイルから、DataFrame を作ることもできます。
temp.csv を次のように修正して保存します。

```
Name,Country,Birthday
Edison,USA,2/11
Hiraga,Japan,12/18
Einstein,Germany,3/14
```

CSV ファイルからの読み込みは、read_csv 関数を使います。

【書式】pandas.read_csv(filepath_or_buffer, sep=', ', delimiter=None,
header='infer', names=None, index_col=None,
dtype=None, skiprows=None, quotechar='"',
escapechar=None, comment=None,
encoding=None, skipfooter=0)

パラメタ名	概要
filepath_or_buffer	入力用のファイル名
sep	区切り文字
delimiter	sep の代わりとなる区切り文字
header	ヘッダー行の行数
names	ヘッダー行のリスト
index_col	行のインデックス番号
dtype	各行のデータタイプ（'a'：np.float64、'b'：np.int32 など）
skiprow	先頭から読み込みをスキップする行数
skipfooter	末尾から読み込みをスキップする行数
encoding	入力文字コード（'utf-8','shift_jis','euc_jp','ascii'など）
quotechar	引用文字
escapechar	エスケープ文字
comment	コメント行の行頭文字
戻り値	CSV ファイルを読み込んだ DataFrame オブジェクト

```
dft = pd.read_csv("temp.csv")

print(dft)
```

	Name	Country	Birthday
0	Edison	USA	2/11
1	Hiraga	Japan	12/18
2	Einstein	Germany	3/14

２．４ matplotlib

matplotlib は Python とその科学計算用ライブラリ NumPy のためのグラフ描画ライブラリです。２次元や３次元で様々な種類のグラフを描画する能力を持っています。

matplotlib を使えば、データを図にプロットできるようになりデータの状態を確認しやすくなります。この matplotlib も、デフォルトで Anaconda に含まれています。

まずは、plt という名で matplotlib.pyplot をインポートします。
また、グラフなどを Jupyter Notebook のセルの下に直接表示するため、インラインの指定をします。

```
import matplotlib.pyplot as plt

%matplotlib inline
```

グラフの表示

matplotlib.pyplot モジュールの plot 関数、show 関数を使用します。
plot 関数の引数で x 軸、y 軸を指定してプロットを行い、show 関数で表示します。x 軸、y 軸の指定は配列やリストで渡します。

練習として、sin 波を表示してみます。
sin 波の X 座標は NumPy の linespace 関数で等差数列を生成し、Y 座標は sin 関数で算出します。

【書式】numpy.linspace(start, stop, num = 50, endpoint = True, retstep = False, dtype = None)

パラメタ名	概要
start	数列の始点
stop	数列の終点
num	生成する ndarray の要素数（デフォルトは５０）
endpoint	生成する数列において、stop を要素に含むかどうか
retstep	生成された配列の後に公差を表示するかどうか
dtype	出力する ndarray のデータ型を指定（ない場合 float）
戻り値	num 等分された等差数列を要素とする ndarray オブジェクト

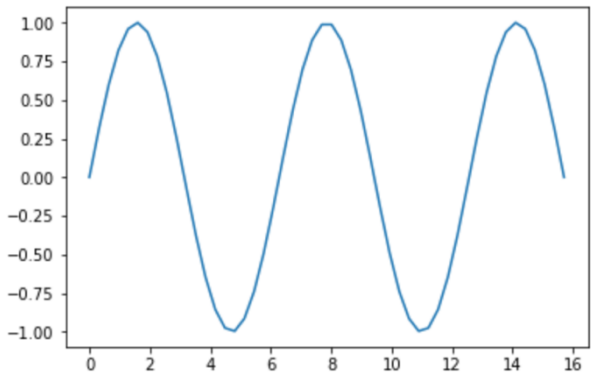
【書式】numpy.sin(x[, out])

パラメタ名	概要
x	ラジアン単位の角度
out	結果を格納する配列
戻り値	三角関数のサインの値

math モジュールの pi で円周率を取得し、X 座標、Y 座標を計算した結果を matpotlib の plot 関数に渡してグラフを生成し、show 関数で表示します。

```
import math
import numpy as np

x = np.linspace(0,5 * math.pi)
y = np.sin(x)
plt.plot(x,y)
plt.show()
```



グラフにタイトルや軸ラベルの設定

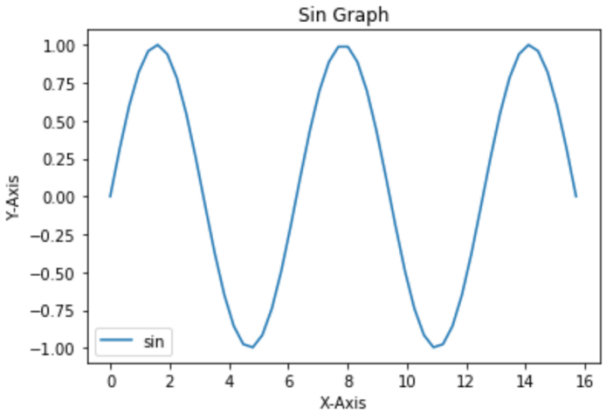
title 関数でグラフのタイトル、xlabel 関数、ylabel 関数でグラフの軸ラベルを設定できます。

グラフの凡例は plot 関数の引数 label で凡例名を指定し、legend 関数で描画します。

ちなみに日本語を表示するするには、matplotlib の設定ファイルを書き換える必要があります。

今回は、グラフを確認するだけなので設定ファイルの書き換えは行いません。

```
plt.title('Sin Graph')
plt.xlabel('X-Axis')
plt.ylabel('Y-Axis')
plt.plot(x,y,label='sin')
plt.legend()
plt.show()
```



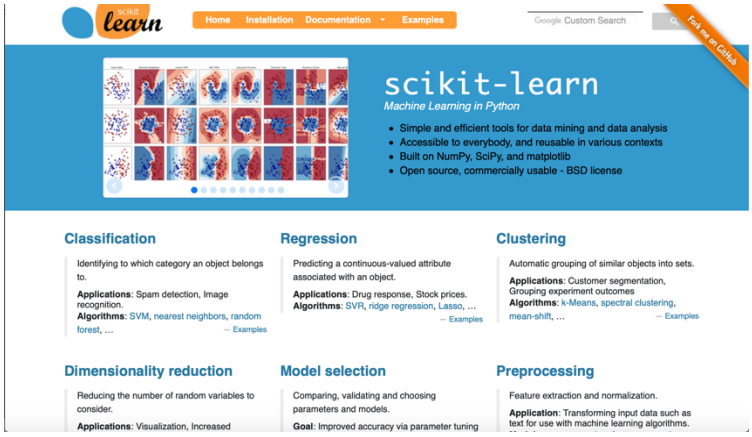
3．scikit-learn による機械学習

3．1 scikit-learn とは

scikit-learn（サイキットラーン）は、オープンソース（BSD ライセンス）のライブラリで、機械学習に必要な回帰、分類、クラスタリングなどのアルゴリズムを揃えています。また、Python の数値計算ライブラリである NumPy や SciPy を内部で利用しています。

公式サイトには、scikit-learn についての詳しいドキュメントがあります。

`http://scikit-learn.org`



scikit-learn の機械学習アルゴリズム

scikit-learn には多くのアルゴリズムが実装されています。

アルゴリズムの選択の参考になる「アルゴリズムマップ」が公式サイトにあります。

`http://scikit-learn.org/stable/tutorial/machine_learning_map/`

代表的なアルゴリズムとして

回帰（regression）

実数値をデータで学習し、実数値の予測をします。

scikit-learn が搭載するアルゴリズムには、SGD 回帰、LASSO 回帰などがあります。

分類（classification）

正解ラベルとそのデータを学習し、データに対してのラベルを予測します。

scikit-learn が搭載するアルゴリズムには、カーネル近似、k 近傍法などがあります。

クラスタリング（clustering）

データの似ている部分をグループにして、データの特徴やパターンを見つけ出します。scikit-learn が搭載するアルゴリズムには、K 平均法、スペクトラルクラスタリングがあります。

次元削減（dimensionality reduction）

データの次元を削減して、固有の構造を見つけ出す手法。または、他の手法の入力に利用します。scikit-learn が搭載するアルゴリズムには、主成分分析（PCA）、カーネル PCA などがあります。

3．2 回帰分析

回帰分析は、「結果データ」と「結果に影響を及ぼすデータ」の関連性を統計的に求める手法です。

この回帰分析を行うモデルとして、scikit-learn には「線形回帰モデル」があります。式は「最小 2 乗法」を用いて求めます。

説明変数が 1 つの場合を「単回帰分析」、複数の場合を「重回帰分析」を呼びますが、scikit-learn では、どちらの回帰分析も行うことができます。

単回帰分析

scikit-learn では「linear_mode.LinearRegression」というクラスを用いることで、線形回帰モデルを作ることができます。「linear_mode.LinearRegression」を使ってモデルを生成するときの書式は

【書式】sklearn.linear_model.LinearRegression(fit_intercept=True,
normalize=False, copy_X=True, n_jobs=1)

パラメタ名	概要
fit_intercept	切片を求める計算を含めるか否か
normalize	説明変数を事前に正規化するか否か
copy_X	メモリー内で上書きしないように X を複製してから実行するか否か
n_jobs	計算に使うジョブの数 －1 にすると、利用可能な CPU をすべて使う

今回の単回帰分析の処理で必要なモジュールをインポートします。

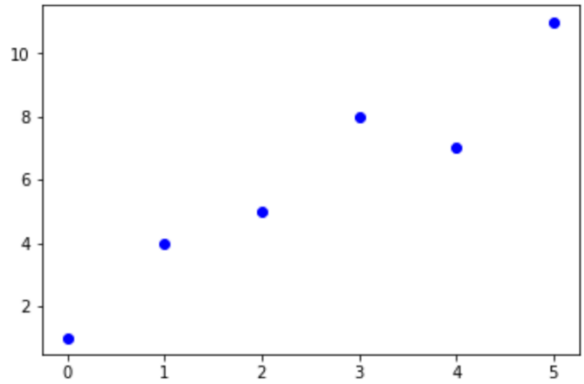
```
import numpy as np
import matplotlib.pyplot as plt
from pandas import DataFrame
from sklearn import linear_model
%matplotlib inline
```

サンプルとして次のような X 軸と Y 軸のデータを pandas の DataFrame 形式で用意します。

```
X = DataFrame([0,1,2,3,4,5])
Y = DataFrame([1,4,5,8,7,11])
```

matplotlib でプロットして散布図を表示させます。

```
plt.scatter(X,Y,color='blue')
```



scikit-learn の LinearRegression で、線形回帰モデルを生成します。

```
model = linear_model.LinearRegression()
```

作成したモデルに fit 関数で X と Y のデータを渡し学習を実行します。

```
model.fit(X,Y)
```

ここまでの操作で、X と Y のデータを学習した線形回帰モデルが完成しました。
このモデルに X の値を与えて予測した Y の値をプロットします。

まず、NumPy の arange 関数を使い、X の最小値から最大値まで 0.01 刻みの配列を生成します。

```
tmp=np.arange(float(X.min()),float(X.max()),0.01)
```

NumPy の newaxis 関数を使い、X 座標を 2 次元配列に変換します。
これは「linear_mode.LinearRegression」というクラスは重回帰分析にも使用することができるクラスで、入力の値 X は複数格納できるよう 2 次元配列にする必要があります。

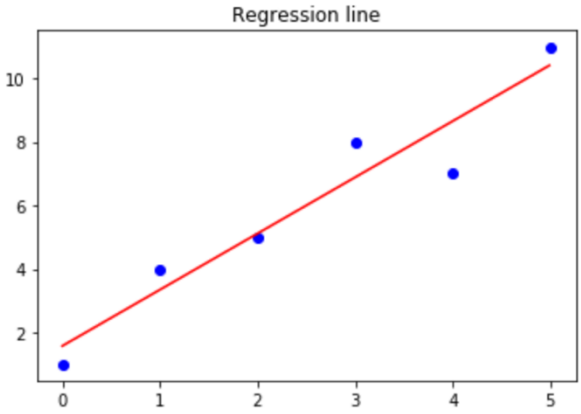
```
px = tmp[:,np.newaxis]
```

X 座標を線形回帰モデルの predict 関数に渡して、Y 座標を作ります。

```
py = model.predict(px)
```

では、元のデータと作成した回帰直線を一緒に表示してみます。

```
plt.scatter(X,Y,color='blue')
plt.plot(px,py,color='red')
plt.title("Regression line")
plt.show()
```



導き出した回帰係数はモデルの coef_属性に、切片は intercept_属性に格納されています。

```
print("回帰係数：",model.coef_)
print("切片：",model.intercept_)
```

３．３ 機械学習用のデータ

本格的な機械学習の実験を行うためには、それなりのデータ量が必要になります。
インターネット上には、機械学習用のサンプルデータがいくつかありますが、まずは scikit-learn に付属しているデータを使うことにします。

scikit-learn には、iris という「アヤメ」（花）のデータセットと、digits という手書き数字のデータセットがあります。

iris データセット

iris データセットは、統計分析や機械学習のサンプルとして有名で、機械学習の入門には、よく取り上げられるデータセットです。
iris データセットの読み込みは、専用の load_iris 関数があるのでこれを利用します。
以下の URL に load_iris 関数の説明があります。

```
http://scikit-learn.org/stable/modules/generated/
                                    sklearn.datasets.load_iris.html
```

```
from sklearn.datasets import load_iris
iris_dataset = load_iris()
```

scikit-learn のドキュメントによると、iris データは、scikit-learn で定義されている Bunch クラスのオブジェクトとして返されます。

キーとして'data'、'target'、'target_names'、'feature_names'を与えると、学習データ、正解ラベル、ラベルの意味、特徴の情報が取得でき、'DESCR'キーで iris データセットの説明が表示されるとあります。

```
print(iris_dataset['DESCR'])
```

表示をみると、アヤメのデータは「１５０個」、単位は「センチメートル」と書いてあります。「Iris-Setosa」「Iris-Versicolour」「Iris-Virginica」の３種類のアヤメの「Sepal（がく片）」と「Petal（花弁）」の「length（長さ）」および「width（幅）」を計測したデータになっています。

'data'キーは、アヤメの特徴データです。列の説明は'feature_name'で表示されます。

```
print(iris_dataset['data'])
print(iris_dataset['feature_names'])
```

'target'キーを表示すると、アヤメの種類を表す０、１、２の３種類の数字が格納されています。'target_names'キーで数字に対応するアヤメの種類名が格納されています。

```
print(iris_dataset['target'])
print(iris_dataset['target_names'])
```

Pandas の DataFrame で表示してみましょう。

```
import pandas as pld
pld.DataFrame(iris_dataset.data,columns=iris_dataset.feature_names)
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
5	5.4	3.9	1.7	0.4
6	4.6	3.4	1.4	0.3
7	5.0	3.4	1.5	0.2
8	4.4	2.9	1.4	0.2
9	4.9	3.1	1.5	0.1
10	5.4	3.7	1.5	0.2

手書き数字のデータセット

「digits」は手書き数字の画像データと、各画像に付けられたラベルのデータです。

digits データセットの読み込みは、専用の load_digits 関数があるのでこれを利用します。

以下の URL に load_digits 関数の説明があります。

https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_digits.html

digits データセットをロードしてみましょう。

```
from sklearn.datasets import load_digits
digits = load_digits()
```

データとその数字を表示してみます。

```
print(digits.data)
print(digits.data.shape)
```

データ１件あたり８×８＝６４個の値が NumPy 配列になっていて、データ件数は 1797 件あることがわかります。

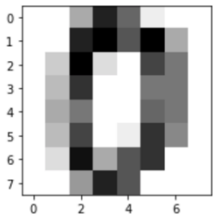
1797 件分の正解ラベルは（０～９）は、target に入っています。

```
print(digits.target)
```

digits.images に８×８のピクセルデータがあるので、matplotlib の imshow 関数で画像を描画してみます。

```
import matplotlib.pyplot as plt

plt.figure(figsize=(3,3))
plt.imshow(digits.images[0],cmap=plt.cm.gray_r)
plt.show()
```



３．４ 教師あり学習（k 近傍法）

k 近傍法は、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です。

データの分割

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評価するためです。

訓練用のデータを評価用にも使ってしまうと、すでに知っているデータなので、正解率は当然高くなり適切な評価ができません。過学習を見逃してしまうことになります。

そこでデータセットを訓練用と評価用に分割してから使います。

データの分割には、scikit-learn の「train_test_split」という関数を使います。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。

【書式】model_selection.train_test_split(*arrays,**options)

パラメタ名	概要
*arrays	特徴行列、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（デフォルトは True）
返り値	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

https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

はじめに iris データセットを読み込みます。

```
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。

train_d を訓練用データ、test_d をテスト用データ、train_l を訓練用の正解ラベル、test_l を評価用の正解ラベルとして戻り値を受けます。

引数 test_size を 0.3 にしているので、評価用データが 3 割、残りが訓練用のデータになります。

```
from sklearn.model_selection import train_test_split
train_d,test_d,train_l,test_l = train_test_split(
                iris_dataset['data'],iris_dataset['target'],
                test_size=0.3,random_state=0)
```

訓練用のデータである train_d 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。

```
import pandas as pld
pld.DataFrame(train_d,columns=iris_dataset.feature_names)
```

	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
3	6.0	2.2	5.0	1.5
4	6.7	2.5	5.8	1.8
5	5.6	2.5	3.9	1.1
6	7.7	3.0	6.1	2.3
7	6.3	3.3	4.7	1.6
8	5.5	2.4	3.8	1.1
9	6.3	2.7	4.9	1.8
101	6.3	2.9	5.6	1.8
102	5.8	2.7	4.1	1.0
103	7.7	3.8	6.7	2.2
104	4.6	3.2	1.4	0.2

105 rows × 4 columns

評価用のデータ test_d には、残りの 45 個が入っています。

```
pld.DataFrame(test_d,columns=iris_dataset.feature_names)
```

正解ラベルも分割されているか確認しておきます。

```
train_l
test_l
```

機械学習モデルの構築

k 近傍法でクラス分類を行う学習モデルを構築します。
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】sklearn.neighbors.KNeighborsClassifier(n_neighbors=5,
weights='uniform', algorithm='auto',
leaf_size=30, p=2, metric='minkowski',
metric_params=None, n_jobs=None, **kwargs)

パラメタ名	概要
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す。KNeighborsClassifier クラスをインポートします。

```
from sklearn.neighbors import KNeighborsClassifier
```

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

```
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 関数に読み込ませて学習させます。

```
knn.fit(train_d,train_l)
```

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。
これで学習が完了しました。

先ほど分割した評価用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もっているかを評価します。
評価は正解率を算出して確認します。

評価用データを学習モデルにセットします。

```
pred = knn.predict(test_d)
```

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。

```
pred
```

```
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, 2, 1, 1,
2, 0, 2, 0, 0])
```

では正解ラベルはどうなっているか確認してみると

```
test_l
```

```
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])
```

１箇所間違えています。精度を計算してみます。

```
np.mean(pred == test_l)
```

```
0.9777777777777777
```

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

それでは、k の値を５（n_neighbors=5）にして学習モデルを作成し直して、学習データを読み込ませて、正解率を上げられないかチューニングしてみましょう。

k の値をいくつか変えてみても、結果は同じでした。評価用データに１つ外れ値が含まれていると考えた方が良いかもしれません。
そもそも評価用データ数が４５で１つの不正解なので９７%の正解率は高いと評価して良いのでしょう。

モデルを使って予測

それでは、学習データを使って予測してみます。
今回は学習用データ中からデータを１つピックアップして使います。
予測したいデータを作ります。要素はがく片の長さ、がく片の幅、花弁の長さ、花弁の幅です。

```
import numpy as np
new_data = np.array([[5.0,2.9,1.0,0.2]])
```

では、クラスを予測させます。

```
pred = knn.predict(new_data)
```

予測の結果は pred に返ってきます。
学習モデルが予想したラベルを表示させます。

```
print(iris_dataset['target_names'][pred])

['setosa']
```

'setosa'のデータを与えたので予測は正解でした。

３．５ 教師あり学習（パーセプトロン）

パーセプトロンには「単純パーセプトロン」と「多層パーセプトロン」の２種類があります。「多層パーセプトロン」の方は、ニューラルネットワークやディープラーニングへつながる技術です。

scikit-learn のパーセプトロン

パーセプトロンの考え方がわかったところで、iris のデータセットを機械学習してみます。

scikit-learn には、パーセプトロンを実装した linear_mode.Perceptron があります。

【書式】sklearn.linear_model.Perceptron(penalty=None, alpha=0.0001,
fit_intercept=True, max_iter=None, tol=None,
shuffle=True, verbose=0, eta0=1.0,
n_jobs=None, random_state=0,
early_stopping=False, validation_fraction=0.1,
n_iter_no_change=5, class_weight=None,
warm_start=False, n_iter=None)

パラメタ名	概要
max_iter	トレーニングデータに対する試行の最大回数（別名エポック）バージョン 0．１９の新機能（推奨）
n_iter	トレーニングデータに対する試行回数（非推奨）
eta0	更新時に乗算される定数
tol	打ち切るための許容誤差の基準
shuffle	トレーニングデータをシャッフルするか否か
random_state	データをシャッフルするときの擬似乱数ジェネレータのシード
返り値	k 近傍法の学習モデルのオブジェクト

この Perceptron から学習モデルを生成して、iris データセットの識別を行います。

まずは、iris データセットを読み込み、訓練データと評価データを生成します。

```
from sklearn.datasets import load_iris
iris_dataset = load_iris()

from sklearn.model_selection import train_test_split
train_d,test_d,train_l,test_l = train_test_split(
                iris_dataset['data'],iris_dataset['target'],
                test_size=0.3,random_state=0)
```

パーセプトロンの学習モデルを生成します。
パーセプトロンでは、重みベクトルである w が少しずつ変更され、決定境界の直線が最適化されていきます。パラメータ eta0 は一度にどれくらい傾かせるかを決める値なので、この値が小さいほど、最適値に到達しやすくなります。最適値に到達した状態を「収束した」と言います。
ただし、一度に傾く量が僅かだと、試行回数が増えて処理が遅くなります。
ここでは、最大試行回数である max_iter を 1000、eta0 を 0.1 にしています。

```
from sklearn.linear_model import Perceptron
ppn = Perceptron(max_iter=1000,eta0=0.1,random_state=0,
                                                    shuffle=True)
ppn.fit(train_d,train_l)
```

これで学習が完了したので精度を評価します。

```
pred = ppn.predict(test_d)
import numpy as np
np.mean(pred == test_l)
```

非線形分離可能

パーセプトロンは線形分離可能なデータに対してクラスを分離できますが、直線で分離できない場合は対応できません。
決定境界が曲線だったり、データが混ざっている場合、直線を使ってクラスを分類することができない状態を「非線形分離可能」（線形分離不可能）と呼びます。
非線形分離可能な問題に対処できるアルゴリズムとして「ロジスティック回帰」と「サポートベクタマシン」があります。

３．６ ロジスティック回帰

ロジスティック回帰は、クラスを分離する判断に「０」か「１」（正か負）ではなく「確率」を用います。
例えば、与えられたデータがクラス１である確率は 80%、そうでない確率は 20%のように判断します。
つまり、たとえ可能性があったとしても、確率が低い場合は決定境界の重みベクトルを修正できます。

ロジスティック回帰で機械学習を行うには、scikit-learn の LogisticRegression を使います。

【書式】sklearn.linear_model.LogisticRegression(penalty='l2',
dual=False, tol=0.0001, C=1.0, fit_intercept=True,
intercept_scaling=1, class_weight=None,
random_state=None, solver='warn', max_iter=100,
multi_class='warn', verbose=0, warm_start=False,
n_jobs=None)

パラメタ名	概要
penalty	正則化の種類（デフォルトは L2 ノルム正則化）
C	Regularization（正則化）の強さ

LogisticRegression の引数にある「penalty」と「C」は「正則化項」と呼ばれます。
正則化とは、モデルの「複雑さ」が過剰に増えないように「ペナルティ」を設ける手法です。過学習にならないように、正則化を行います。
モデルの「複雑さ」を表す指標に「L1 ノルム正則化」と「L2 ノルム正則化」があり、係数である引数 C の値を多くすると正則化が強く働きます。

今回は正則化を行わずに機械学習を行なってみます。
データには digits データを使用します。

```
from sklearn.datasets import load_digits
digits_dataset = load_digits()

from sklearn.model_selection import train_test_split
train_d,test_d,train_l,test_l = train_test_split(
               digits_dataset['data'],digits_dataset['target'],
               test_size=0.3,random_state=0)
```

ロジスティック回帰の学習モデルを生成し、学習を行います。

```
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression()
logreg_model = logreg.fit(train_d,train_l)
```

これで、学習が完了したので、精度を評価します。

```
pred = logreg_model.predict(test_d)

import numpy as np
np.mean(pred == test_l)
```

３．７ サポートベクターマシン

サポートベクターマシン（SVM）は２クラス識別のアルゴリズムでは「強力」と言われていますが、アルゴリズムが複雑です。

SVM の特徴とには「マージン最大化」と「カーネルトリック」があります。

「マージン」とは、識別面と２つのクラスの間の距離のことで、マージンが最大になるようにすることで、「汎化能力を最大」にします。

学習量が少なくても評価データ（未知のデータ）の判別精度が高い学習モデルのことを「汎化性能が高い」と言います。

SVM はマージン最大化によって、未知のデータに対して判別精度が高いアルゴリズムです。

ただし、マージン最大化は、あくまでも線形分離可能なデータに対してのみ有効な手法であるので、SVM は「カーネルトリック」という手法を取り入れることで、線形分離が難しい問題に対処しています。

「カーネルトリック」とは、高次元空間上で線形分析を行う機能です。これまでのような２次元の分析ではなく、３次元、４次元へと写像して線形分析を行います。

非線形な識別を可能にする学習モデルを作るには、svm.SVC を使います。

【書式】sklearn.svm.SVC(C=1.0, kernel='rbf', degree=3,
gamma='auto_deprecated', coef0=0.0,
shrinking=True, probability=False, tol=0.001,
cache_size=200, class_weight=None, verbose=False,
max_iter=-1, decision_function_shape='ovr',
random_state=None)

パラメタ名	概要
C	どれだけの誤分類を許容するか
kernel	カーネルの種類 （linear,poly,rbf,sigmoid,percomputed）
gamma	カーネルの種類が rbf,poly,sigmoid のとき使用

機械学習の手順はロジスティック回帰と同じです。

今回もデータには digits データを使用します。

```
from sklearn.datasets import load_digits
digits_dataset = load_digits()

from sklearn.model_selection import train_test_split
train_d,test_d,train_l,test_l = train_test_split(
                digits_dataset['data'],digits_dataset['target'],
                test_size=0.3,random_state=0)
```

svm をインポートして、svm.SVC の学習モデルを生成します。
パラメータの gamma=0.001 は、kernel が'rbf'（デフォルト）の場合の決定境界の複雑度合いを表し、大きいほど複雑になります。
また、C=100. は誤分類を許容する尺度です。

```
from sklearn import svm
clf = svm.SVC(gamma=0.001,C=100.)
clf_model = clf.fit(train_d,train_l)
```

これで、学習が完了したので、精度を評価します。

```
pred = clf_model.predict(test_d)

import numpy as np
np.mean(pred == test_l)
```

3．8 ニューラルネットワーク

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。

scikit-learn の MLPClassifier クラスを使って機械学習を行います。

MLPClassifier クラスは、多重パーセプトロン（MLP）方式で実装されています。

【書式】sklearn.neural_network.MLPClassifier(
hidden_layer_sizes=(100,), activation='relu',
solver='adam', alpha=0.0001, batch_size='auto',
learning_rate='constant', learning_rate_init=0.001,
power_t=0.5, max_iter=200, shuffle=True,
random_state=None, tol=0.0001, verbose=False,
warm_start=False, momentum=0.9,
nesterovs_momentum=True, early_stopping=False,
validation_fraction=0.1, beta_1=0.9, beta_2=0.999,
epsilon=1e-08, n_iter_no_change=10)

パラメタ名	概要
hidden_layser_sizes	隠れ層の数とニューロンの数
activation	活性化関数'identifify','logistic','tanh','relu'
slover	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	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

MLPClassifier を使って、iris データセットの分類を行います。

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。

```
from sklearn.datasets import load_iris
iris_dataset = load_iris()

from sklearn.model_selection import train_test_split
train_d,test_d,train_l,test_l = train_test_split(
               iris_dataset['data'],iris_dataset['target'],
               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。

```
from sklearn.neural_network import MLPClassifier
mlpc = MLPClassifier()
mlpc.fit(train_d,train_l)
```

評価用データを分類して正解率を表示します。

```
pred=mlpc.predict(test_d)
import numpy as np
np.mean(pred == test_l)
```

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。

```
from sklearn.datasets import load_digits
digits_dataset = load_digits()

from sklearn.model_selection import train_test_split
train_d,test_d,train_l,test_l = train_test_split(
               digits_dataset['data'],digits_dataset['target'],
               test_size=0.3,random_state=0)


from sklearn.neural_network import MLPClassifier
mlpc = MLPClassifier()
mlpc.fit(train_d,train_l)

pred=mlpc.predict(test_d)
import numpy as np
np.mean(pred == test_l)
```

３．９ 教師なし学習

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りますがが、今回は「クラスタリング」（クラスター分析）を行います。

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、「k 平均法」（k-means clustering）を使用します。

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 平均法は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。

k 平均法で観測データを３つのクラスに分離する場合、次の手順で行います。

（１）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。
（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。
（３）再び重心を計算します。重心に変更があった場合には、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、再び重心をもとめるという作業を繰り返します。

重心に変化がなくなるとクラスタリングが完了します。

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ります。

k 平均法を scikit-learn で実行します。
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】sklearn.cluster.KMeans(n_clusters=8, init='k-means++',
n_init=10, max_iter=300, tol=0.0001,
precompute_distances='auto', verbose=0,
random_state=None, copy_x=True, n_jobs=None,
algorithm='auto')

パラメタ名	概要
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
percompute_distances	距離を事前に計算するか否か

今回は iris データセットを 3 つのクラスに分類します。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。

```
from sklearn.datasets import load_iris
iris_dataset = load_iris()
iris_dataset['target']
```

iris_dataset の中には 3 種類のアヤメデータが入っています。

それでは、KMeans の学習アルゴリズムに iris の計測データだけを流して 3 つのクラスに分離させます。

```
from sklearn.cluster import KMeans
kme = KMeans(n_clusters=3)

kme.fit(iris_dataset['data'])
```

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。
クラスタリングのラベルを確認します。

```
kme.labels_
```

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、たいだい最初の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に分割できていそうです。