岐阜県産業技術総合センター
Gifu Prefectural Industrial Technology Center

MAKE NEW STANDARDS.
東海国立大学機構

岐阜大学

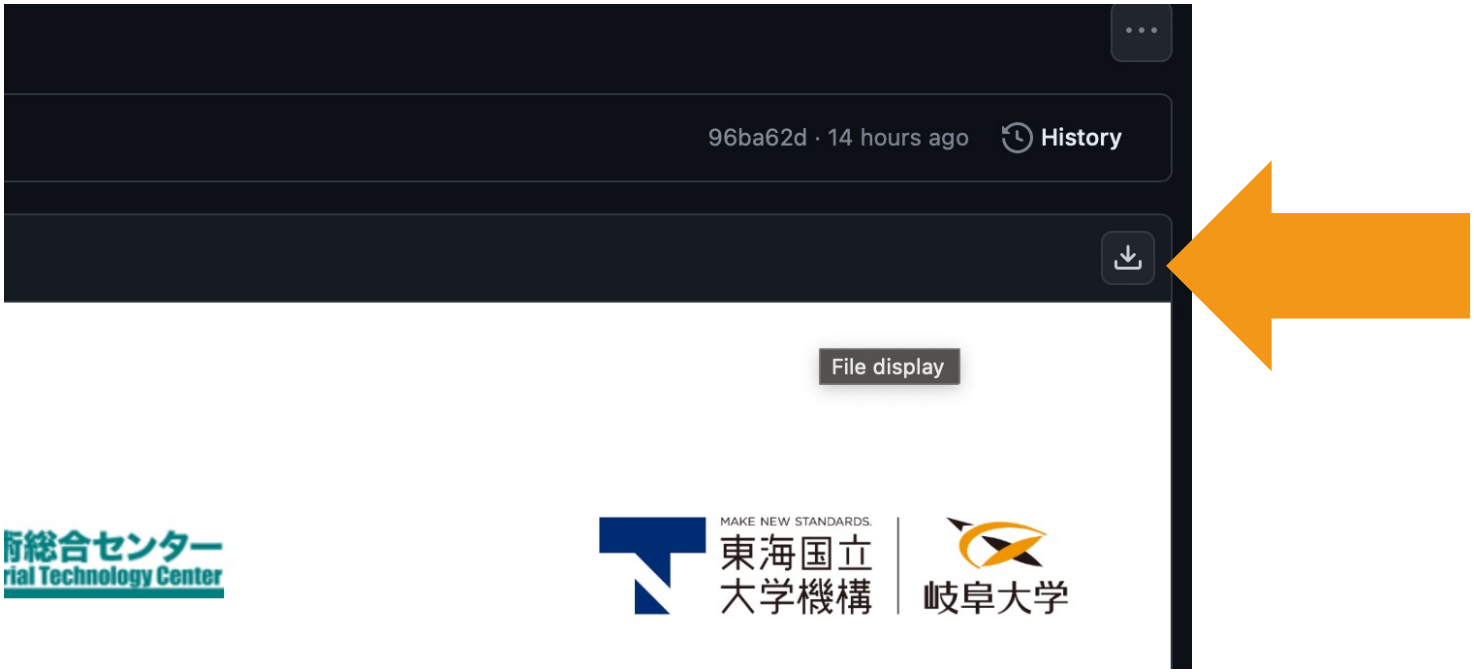
データサイエンス講習会

講義資料

岐阜大学 工学部 電気電子・情報工学科　鈴木 優

本日の資料

- https://github.com/yu-suzuki/ds2023



SageMaker Studio Labとは?

- Amazon が開発したプログラム開発環境
  - ブラウザだけでアクセス可能
  - 自分でPythonの環境を作らなくても良い
- 1日あたりの使用制限
  - CPU: 8時間/日
  - GPU: 4時間/日（今回は使わない）
- 一回あたりの使用可能時間: 4時間
  - 4時間経過するといったん切断されます

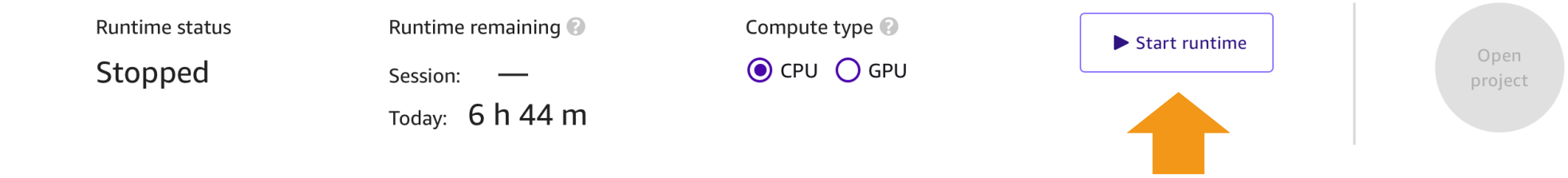
SageMaker Studio Labの使い方

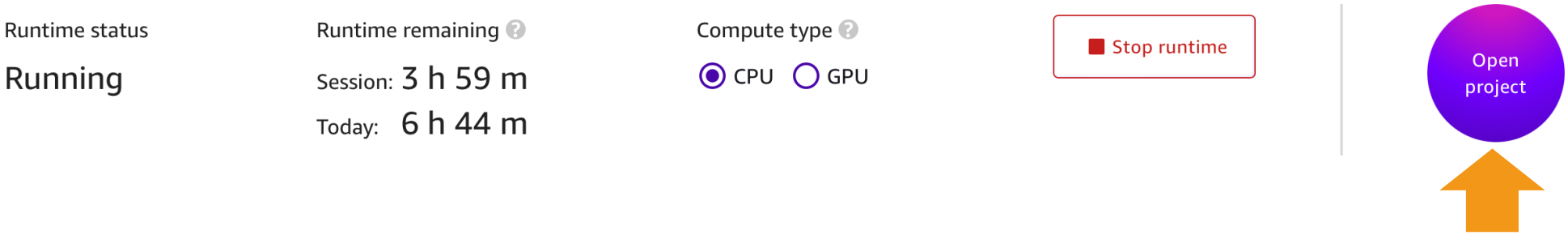
- SageMaker Studio Lab のページ
  - https://studiolab.sagemaker.aws
- アカウント作成資料
  - https://github.com/aws-sagemaker-jp/awesome-studio-lab-jp/blob/main/README_usage.md
- リファラルコードは次の通りです

gifu2024-AEBD5

準備

- ログインしてからボタンを押してスタート





モジュール

モジュールとは

- 様々な機能をまとめて提供する仕組み
  - 機能を組み込む ＝ モジュールをインポートする
  - 様々なモジュールが提供されている
  - 一つのモジュールには複数の機能が実装されている
- モジュールのインストール方法
  - %pip install モジュール名
- 2種類のモジュールのインポート方法
  - 全部使う
    - import モジュール名
  - 一部だけ使う
    - from モジュール名 import 機能名

モジュールのインポート

- モジュールの機能全てをインポート

  import モジュール名

- インポートされたモジュール中の機能(関数)を使用

  モジュール .(ドット) 機能

  (例) mathモジュールのsqrt機能を使うとき → math.sqrt

```
[4]: import math

[5]: math.sqrt(2)

[5]: 1.4142135623730951
```

モジュール名に別名をつける

- モジュールの機能全てを別名付きでインポート
  import モジュール名 as 別名
- インポートされたモジュール中の機能(関数)を使用
  別名 .（ドット） 機能

（例）mathモジュール(別名m)のsqrt機能を使うとき → m.sqrt

```
[6]: import math as m

[8]: m.sqrt(2)

[8]: 1.4142135623730951
```

特定の機能だけをインポート

- モジュールの特定の機能だけインポート

  from モジュール名 import 機能

- インポートされたモジュール中の機能(関数)を使用

  機能

  (例) mathモジュールのsqrt機能を使うとき → sqrt

```
[9]: from math import sqrt

[10]: sqrt(2)

[10]: 1.4142135623730951
```

特定の機能だけを別名付きインポート

- モジュールの特定の機能だけ別名付きでインポート

  from モジュール名 import 機能 as 別名

- インポートされたモジュール中の機能(関数)を別名で使用

  別名

  (例) mathモジュールのsqrt機能を使うとき → sq

```
[11]: from math import sqrt as sq

[12]: sq(2)

[12]: 1.4142135623730951
```

numpy

数値計算モジュール

numpy

- ベクトルや行列の計算を行うライブラリ

- numpy のインストール

```
%pip install numpy
```

- numpy を 別名 np でインポート

```
import numpy as np
```

numpyで配列を扱う

- ndarray(別名 array)
  - 中身を指定して1次元配列を作成

    a = np.array([1, 2, 3, 4])

1	2	3	4

  - 中身を指定して2次元配列を作成
    - a = np.array([[1,2,3,4], [5,6,7,8]])

1	2	3	4
5	6	7	8

等差数列

- 数列のうち差が一定のもの
  - 1, 2, 3, 4, 5, …
  - 2, 4, 6, 8, 10, …
- 作成方法1
  - 0から終了値-1まで1ずつ増える等差数列を作る

np.arange(終了値)

```
[4]: np.arange(10)

[4]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

最初の値が0

最後の値が終了値-1

等差数列

- 作成方法2
  - 開始値から終了値-1まで1ずつ増える等差数列を作る
    np.arange(開始値, 終了値)

```
[6]: np.arange(5,10)

[6]: array([5, 6, 7, 8, 9])
```

最初の値が開始値

最後の値が終了値-1

等差数列

- 作成方法3
  - 開始値から終了値-1までnずつ増える等差数列を作る

np.arange(開始値, 終了値,ステップ幅)

```
[7]: np.arange(5,6,0.1)

[7]: array([5. , 5.1, 5.2, 5.3, 5.4, 5.5, 5.6, 5.7, 5.8, 5.9])
```

最初の値が5

ステップ幅ずつ増える

最後の値が終了値-ステップ幅

6-0.1

演習

- 0から7まで1ずつ増える等差数列を作成せよ
- 5から9まで1ずつ増える等差数列を作成せよ
- 0から1まで0.1ずつ増える等差数列を作成せよ

配列のサイズ

- size: 全要素数を数える
- shape: 配列の形(行数と列数)を数える

```
[8]: a = np.array([[1,2,3,4],[5,6,7,8]])

[9]: a.size

[9]: 8
```

全部で要素は8個

```
[10]: a.shape

[10]: (2, 4)
```

2行4列

1	2	3	4
5	6	7	8

多次元配列を作成

- 単位行列(対角線の要素が1)
  - np.eye(次元数)
- ゼロ行列(全部0)
  - np.zeros(次元数)
- 全部の要素が1の行列
  - np.ones(次元数)

- 次元数の指定方法
  - 1次元: 個数
  - 2次元以上: (0次元目, 1次元目, …)

```
[13]: a = np.zeros((2,4))

[14]: a

[14]: array([[0., 0., 0., 0.],
             [0., 0., 0., 0.]])
```

要素へアクセス

- i行目j列目へのアクセスは a[i, j]

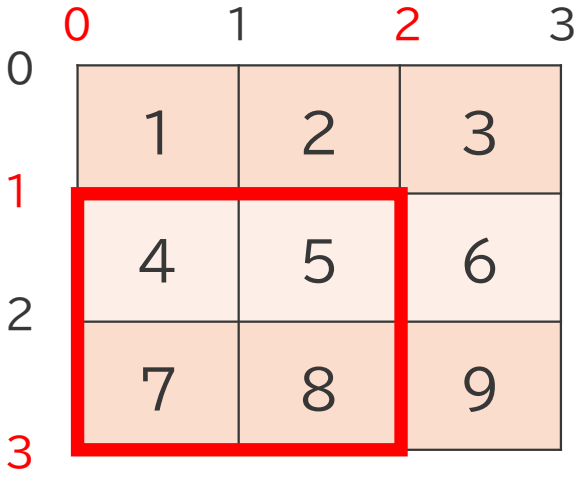
	0	1	2	3
0	1	2	3	4
1	5	6	7	8

0から始まることに注意

```
[15]: a = np.array([[1,2,3,4],[5,6,7,8]])

[16]: a

[16]: array([[1, 2, 3, 4],
             [5, 6, 7, 8]])

[17]: a[1,1]

[17]: 6

[18]: a[1,3]

[18]: 8
```

要素へアクセス

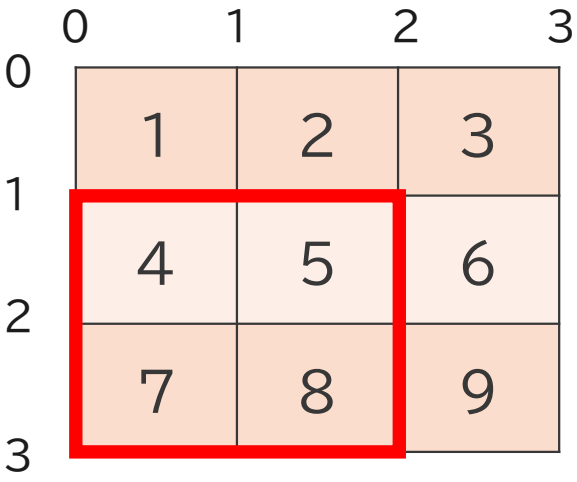
- 部分配列の取得
  - A～B行目C～D列目へのアクセスは a[A:B, C:D]




```
[19]: a = np.array([[1,2,3],[4,5,6],[7,8,9]])

[20]: a[1:3,0:2]

[20]: array([[4, 5],
             [7, 8]])
```

要素へアクセス

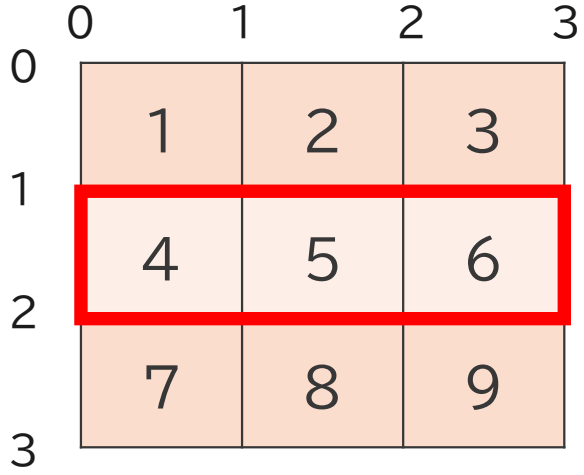
n → n:n+1 と同じ意味



コロンの前の値を省略すると先頭
コロンの後の値を省略すると最後

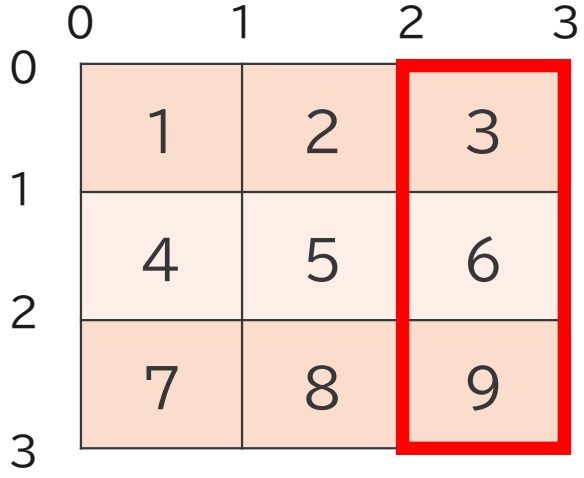
a[1:2, 0:3]
a[1, 0:3]

a[1, :]



a[0:3, 2:3]
a[0:3, 2]

a[:, 2]



要素へアクセス

- 特定の要素を抽出する
  - マスク行列: True(真: T)とFalse(偽: F)からなる行列

m =

F	F	F
T	T	F
T	T	F

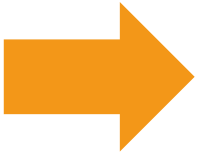
要素へアクセス

a =

1	2	3
4	5	6
7	8	9

m =

F	F	F
T	T	F
T	T	F



a[m] =

4	5
7	8

要素へアクセス

- 3より小さい要素だけTrue, それ以外はFalseに

a =

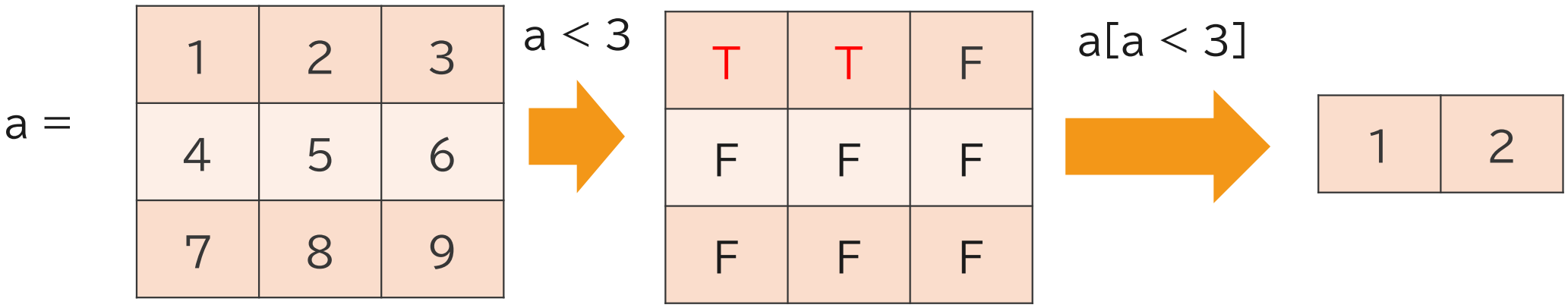
1	2	3
4	5	6
7	8	9

a < 3

T	T	F
F	F	F
F	F	F

要素へアクセス

- 3より小さい要素だけを取り出す



配列のサイズ変更（reshape）

- 1×9の配列を3×3の配列に変更
  - reshape(変更後のサイズ)

```
[24]: a = np.arange(9)

[25]: a

[25]: array([0, 1, 2, 3, 4, 5, 6, 7, 8])

[26]: a.reshape((3,3))

[26]: array([[0, 1, 2],
             [3, 4, 5],
             [6, 7, 8]])
```

配列の演算

- 関数
  - ceil(x) : x以上の最小の整数
  - floor(x) : x以下の最大の整数
  - fabs(x) : 絶対値
  - exp(x) : eのx乗
  - power(x,y) : xのy乗
  - sqrt(x) : $\sqrt{x}$
  - log(x) : $\log x$
  - sin(x), cos(x), tan(x) :
    - xはラジアン
- 定数
  - pi, e, inf(無限大), nan(非数)

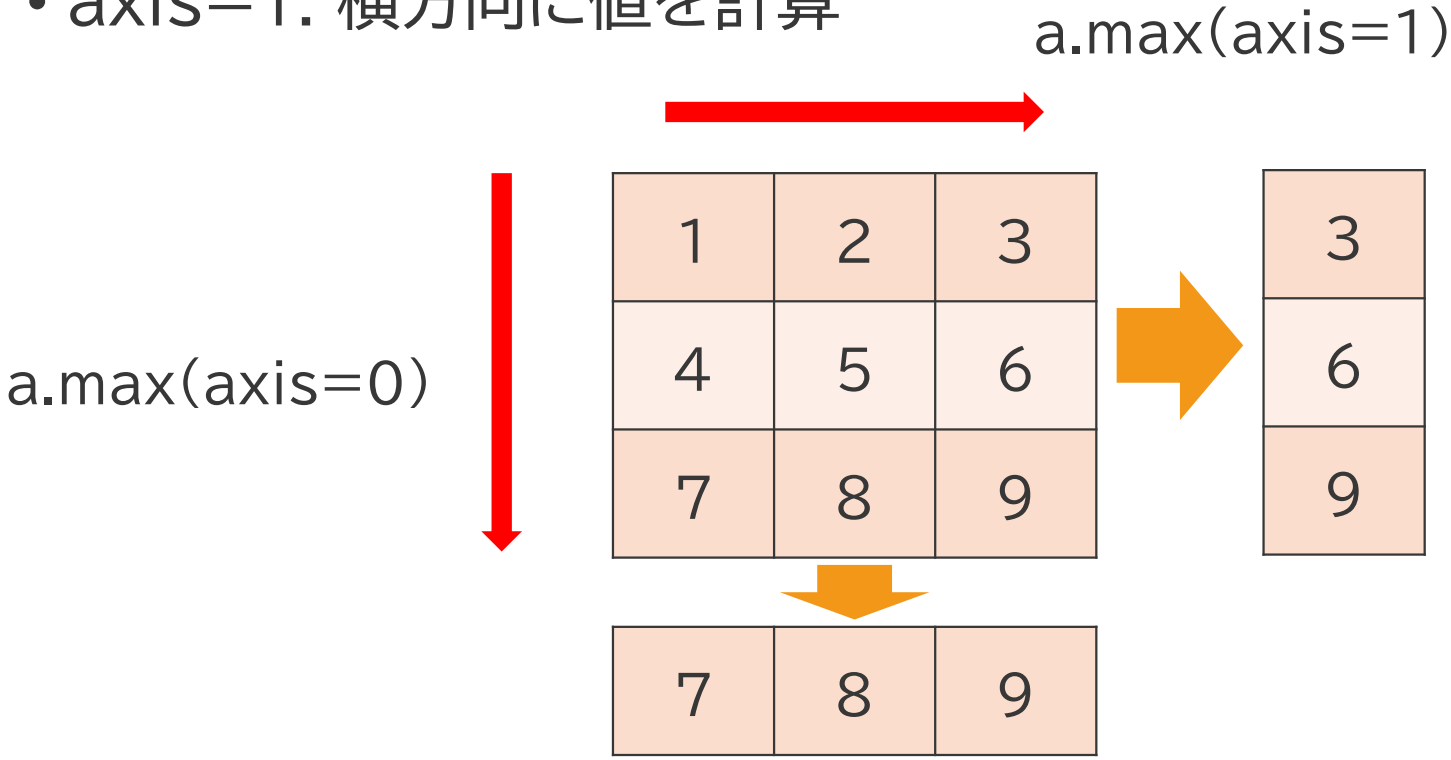
```
[28]: a = np.arange(9).reshape((3,3))

[29]: a

[29]: array([[0, 1, 2],
             [3, 4, 5],
             [6, 7, 8]])

[30]: a+a

[30]: array([[ 0,  2,  4],
             [ 6,  8, 10],
             [12, 14, 16]])

[31]: a*3

[31]: array([[ 0,  3,  6],
             [ 9, 12, 15],
             [18, 21, 24]])
```

配列の演算

- a.sum() : 要素の総和
- a.mean() : 要素の平均
- a.max() : 要素の最大
- a.min() : 要素の最小

配列の演算

- axis=0: 縦方向に値を計算
- axis=1: 横方向に値を計算



演習

1. 3×3の単位行列を作成
2. 1. の配列に×10したものを作成
3. 3×3の行列 a を 2. に掛ける
4. 3. の配列の右上2×2の部分を切り出す
5. 4.の行列を4×1の行列に変更

a =

1	2	3
4	5	6
7	8	9

matplotlib

データの可視化

matplotlibとは

- 2D/3Dのグラフ描画のためのライブラリ
- データの可視化に利用

- インストール

```
%pip install matplotlib
```

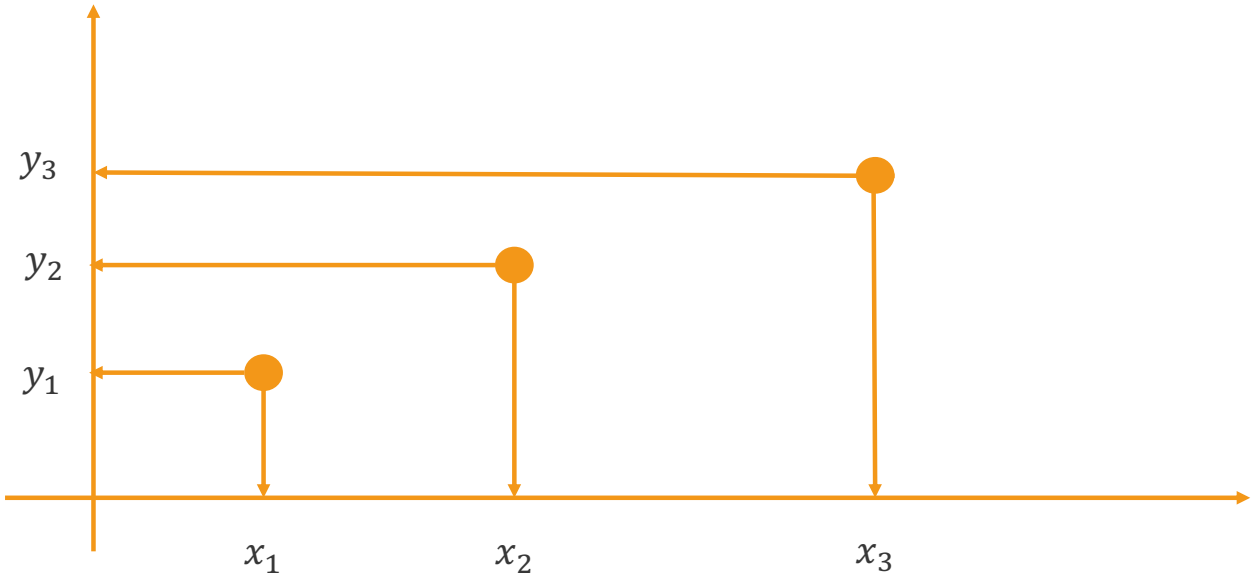
- インポート

```
%matplotlib inline（ページ内にグラフ表示）
import matplotlib.pyplot as plt（別名pltとする）
```

2Dグラフの作成方法

- x軸とy軸それぞれの値を格納したベクトルを準備

$y = [y_1, y_2, y_3]$



$x = [x_1, x_2, x_3]$

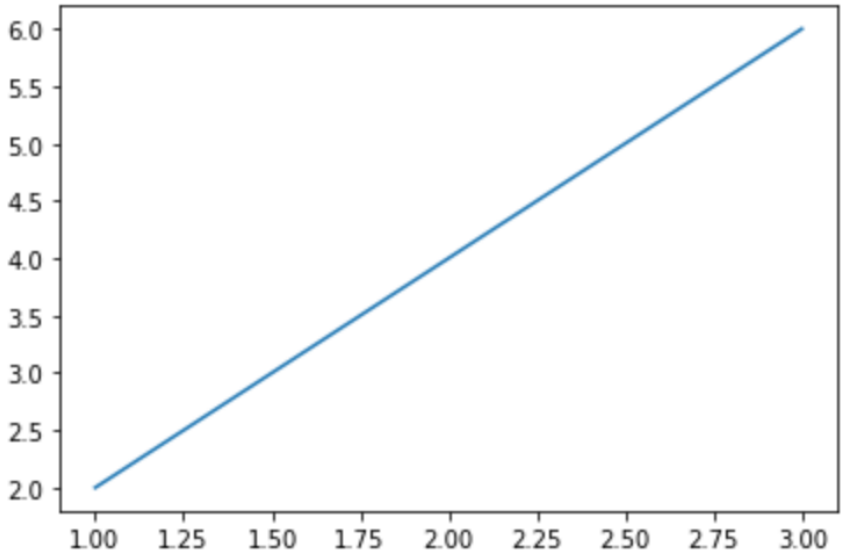
描画

plt.plot(x,y)で描画

[34]:
```
import matplotlib.pyplot as plt
```

[35]:
```
x = [1,2,3]
y = [2,4,6]
plt.plot(x,y)
```

[35]:
```
[<matplotlib.lines.Line2D at 0x7fa75beaba30>]
```



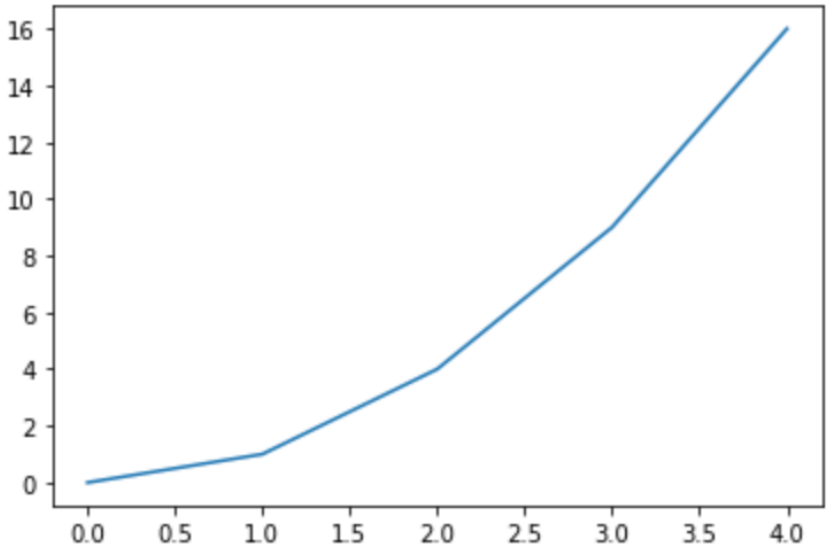
$y = x^2$を図にしてみる

- x軸の値を x= [0,1,2,3,4]
- y軸の値を

y = x*x = [0,1,4,9,16]

```
[36]: x = np.arange(5)
      y = x * x
      plt.plot(x,y)
```

```
[36]: [<matplotlib.lines.Line2D at 0x7fa75bda51f0>]
```

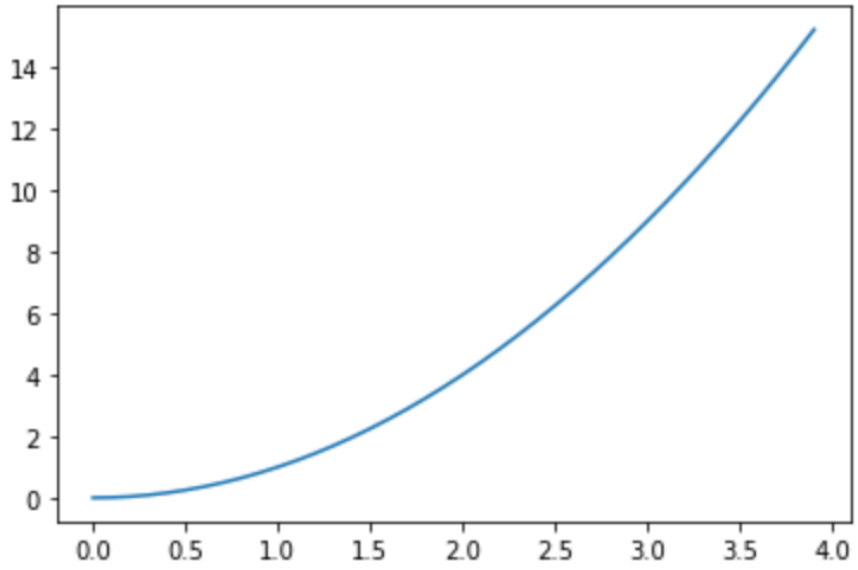


より細かい描画へ変更

- x軸の値を調整
  - np.arange(5)→ [0,1,2,3,4]
  - np.arange(0,5,0.1)
  - →[0,0.1,0.2,…,4.9]

```
[37]: x = np.arange(0,4,0.1)
      y = x * x
      plt.plot(x,y)
```

```
[37]: [<matplotlib.lines.Line2D at 0x7fa75bd13070>]
```



表示範囲の調整

- X軸の表示範囲を-1～1にする→plt.xlim(-1,1)
- Y軸の表示範囲を-1～1にする→plt.ylim(-1,1)
- plt.plot関数と同じコードブロックに書く必要がある

グラフのタイトル

- plt.title(’タイトル’) でタイトル表示
- plt.plot関数と同じコードブロックに書く必要がある

グラフを重ね合わせる

- 同じコードブロック内でplt.plotを2回実行

細かい調整

- 軸にラベルを付ける→plt.xlabel(), plt.ylabel()
- グラフに罫線を付ける→plt.grid()
- グラフを複数並べる→plt.subplot()

課題1-1, 1-2, 1-3

- 1-1: numpyを別名npでインポートしてください
- 1-2: 4×4の単位行列aを生成してください
- 1-3: 次のような行列bを作成してください

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

課題 1-4～1-10

- 1-4: 1-3で作った行列の0行目をcとして生成してください. また, 1-3で作った行列の1列目をdとして生成してください.
- 1-5: 1-4で作ったdは行ベクトルになります. slice(1)ではなくslice(1:2)として, 列ベクトルにしてください.
- 1-6: 1-2で作ったベクトルaと1-3で作ったベクトルbを要素ごとに加算し, 行列fを作ってください.
- 1-7: 行列fの3行2列目の要素を表示してください
- 1-8: 行列fの各要素についてcosの値を計算し, 行列gを作ってください
- 1-9: 行列gを横方向に和をとったベクトルhを作ってください
- 1-10: 行列gを縦方向に和をとったベクトルiを作ってください
- 1-11: ベクトルiのうち0以上の要素だけを持つベクトルjを作ってください
- 1-12: ベクトルiのうち0以上の要素の和を求めて表示してください

課題2

- 2-1: numpyとmatplotlib.pyplotをインポートしてください
- 2-2: -5～5の区間で0.01刻みの配列xを生成してください
- 2-3: 配列xの各要素に対してy=sin2xを計算し, 配列yとしてください
- 2-4: 配列x,yを使ってy=sin2xのグラフを描画してください
- 2-5: 2-4のx軸方向の表示範囲を $-\pi$～ πにしてください
- 2-6: 2-5のグラフにタイトルを付けてください

課題3

次のようなグラフを作成してください

- -5～5の区間で0.01刻みの配列xを生成してください
- 関数$y = e^x \sin 4x$のグラフを配列xを使って作成してください
- x軸は-2π～ 2πの範囲にしてください
- x軸とy軸の両方にそれぞれ「x」「y」をラベルとして表示してください
- グラフに罫線を付けてください

pandas

データの入出力

pandasの機能

- 各種データの入出力(CSV, JSON, Excelなど)
- データに対する各種処理
  - 欠損値処理
  - など
- モジュールのインストール
  - %pip install pandas
- モジュールの別名に pd を使う
  - import pandas as pd

CSVファイルとは？

- Comma Separated Value
  - カンマでデータを区切る方法
  - 拡張子は csv

```
A, B, C
100,200,300
200,300,400
300,400,500
```

CSVファイルの読み込み

- pd.read_csv(ファイルへのパス)
  - 戻り値は DataFrame という表形式データ
  - 先頭行に見出しなければ pd.read_csv(ファイルパス,header=None)

pd.read_csv(https://bit.ly/3MmSz2v)

DataFrameからnumpy配列へ

- DataFrameの変数に対して .values を追加

```
c = pd.read_csv(ファイルのパス)
a = c.values      #aはnumpyの配列
```

CSVで読み込んだ内容を描画

```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
c = pd.read_csv(' https://bit.ly/3MmSz2v' )
a = c.values
x = a[:,0]
y1 = a[:,1]
y2 = a[:,2]
plt.plot(x,y1)
plt.plot(x,y2)
```

データ点だけを表示する

- linewidth, marker, markersizeを設定

plt.plot(x,y1) →
plt.plot(x,y1,linewidth=0,marker='o', markersize=5)

線の太さ　マーカーの形　マーカーの大きさ

不完全なデータ

- 欠損したデータ: https://bit.ly/4aplnmO

- どうするか？
  1. 欠損している部分を削除する
  2. 何か適当な値を入れる
  3. 平均値で補完する
  4. 予測して補完する

欠損しているデータを削除する

1. 欠損している部分を削除する
   .dropna()
2. 何か適当な値を入れる
   .fillna(適当な値)
3. 平均値で補完する
   .fillna(平均値)
4. 予測して補完する
   - scikit-learn を使って予測して補完

scikit-learn

機械学習の基礎

scikit-learn

- 機械学習の様々な方法を網羅したライブラリ
  - 値の推定, 分類, クラスタリングなど
- 様々な手法がこのライブラリに全て凝縮

線形回帰

- 回帰 = ある点を推定すること
  - 例）身長150cmの人の平均体重は50kgです. 身長160cmの人の平均体重は56.3kgです. それでは, 身長170cmの人の平均体重は何kgでしょう？
  - いくつか事例が与えられたとき, その事例にあてはまらない場合にどのような結果が得られるかを予想する問題.

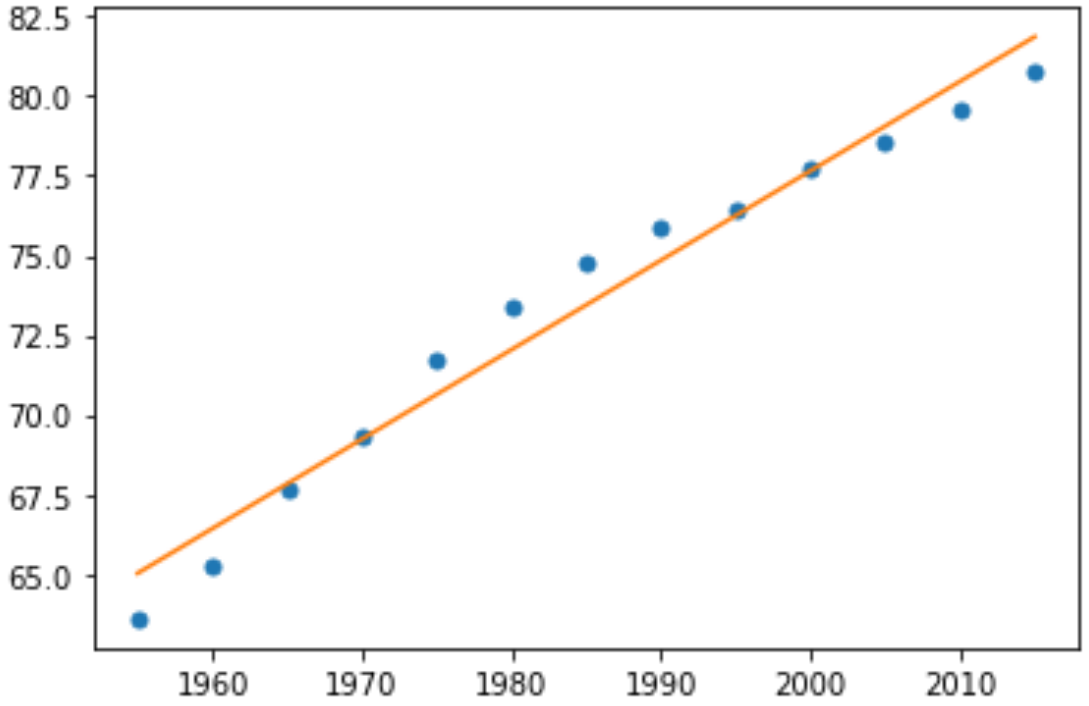
最小二乗法を使って線形回帰

- 入力xと出力yから係数aとbを求める

$$y = ax + b$$

[9]:

	Year	Male	Female
0	1955	63.6	67.8
1	1960	65.3	70.2
2	1965	67.7	72.9
3	1970	69.3	74.7
4	1975	71.7	76.9
5	1980	73.4	78.8
6	1985	74.8	80.5
7	1990	75.9	81.9
8	1995	76.4	82.9
9	2000	77.7	84.6
10	2005	78.6	85.5
11	2010	79.6	86.3
12	2015	80.8	87.0



たくさんのデータを用意

- (x_1, y_1), (x_2, y_2), (x_3, y_3), …
- (1955,63.6), (1960, 65.3), (1965, 67.7),…

- 数式に当てはめてみる
  - $63.6 = a \cdot 1955 + b$
  - $65.3 = a \cdot 1960 + b$
  - $67.7 = a \cdot 1965 + b$

$$\begin{pmatrix} 63.6 \\ 65.3 \\ 67.7 \end{pmatrix} = \begin{pmatrix} 1955 & 1 \\ 1960 & 1 \\ 1965 & 1 \end{pmatrix} \cdot \begin{pmatrix} a \\ b \end{pmatrix}$$

$$Y \qquad X \qquad \beta$$

scikit-learnをインポート

- インストール
  %pip install scikit-learn

- 線形モデルをインポート
  from sklearn import linear_model

- 初期化
  lr = linear_model.LinearRegression()

関数を使う

- fit関数
  - データからモデルを学習
  - XとYからβを計算
- predict関数
  - 入力xからβを使って出力yを予測

$$\begin{pmatrix} 63.6 \\ 65.3 \\ 67.7 \end{pmatrix} = \begin{pmatrix} 1955 & 1 \\ 1960 & 1 \\ 1965 & 1 \end{pmatrix} \cdot \begin{pmatrix} a \\ b \end{pmatrix}$$

Y X β

predict関数で推測する

fit関数でこれを学習で求める

学習

- lr.fit(入力X, 出力Y)

[7]:
```
import numpy as np
x = np.array([[1955], [1960], [1965]])
y = np.array([[63.6], [65.3], [70.2]])
```

[3]:
```
from sklearn import linear_model
lr = linear_model.LinearRegression()
```

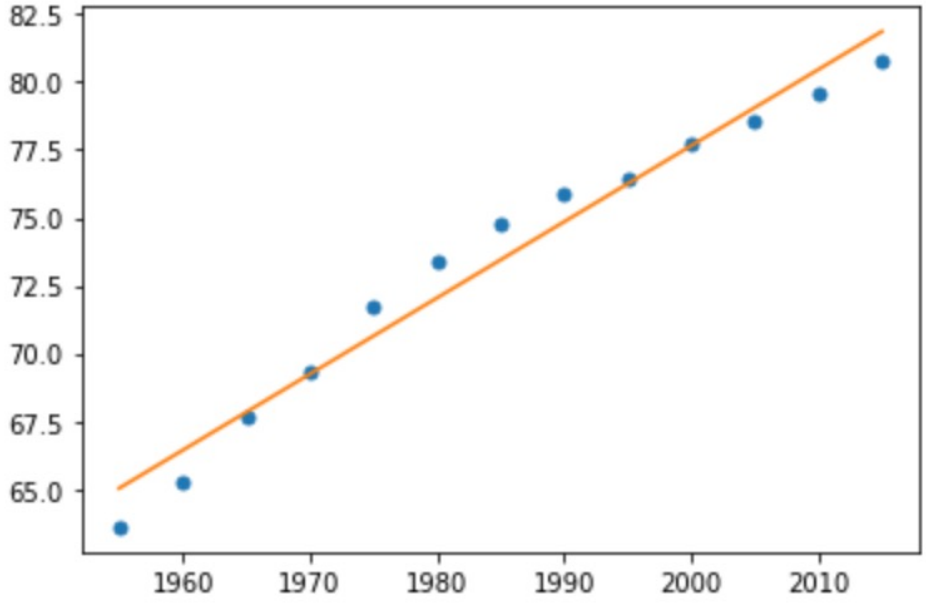
[8]:
```
lr.fit(x,y)
```

[8]:
```
LinearRegression()
```

回帰と描画

[7]:
```
plt.plot(x,y,linewidth=0,marker='o',markersize=5)
yy = lr.predict(x)
plt.plot(x,yy)
```

[7]:
```
[<matplotlib.lines.Line2D at 0x7f24b4d40d90>]
```



いろいろな回帰方法

- Lasso

```
from sklearn import linear_model
lr = linear_model.Lasso()
```

- Ridge

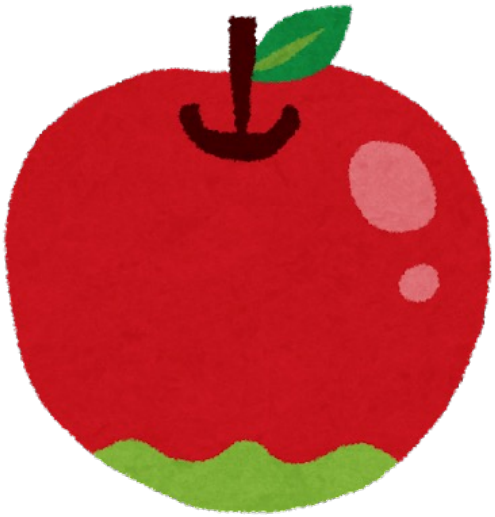
```
from sklearn import linear_model
lr = linear_model.Ridge()
```

機械学習の基礎

分類

分類とは？

- これはりんごかどうかを自動的に認識したい



入力画像

丸い度合い: 0.8
赤成分: 1.0
緑成分: 0.2
青成分: 0.2
...

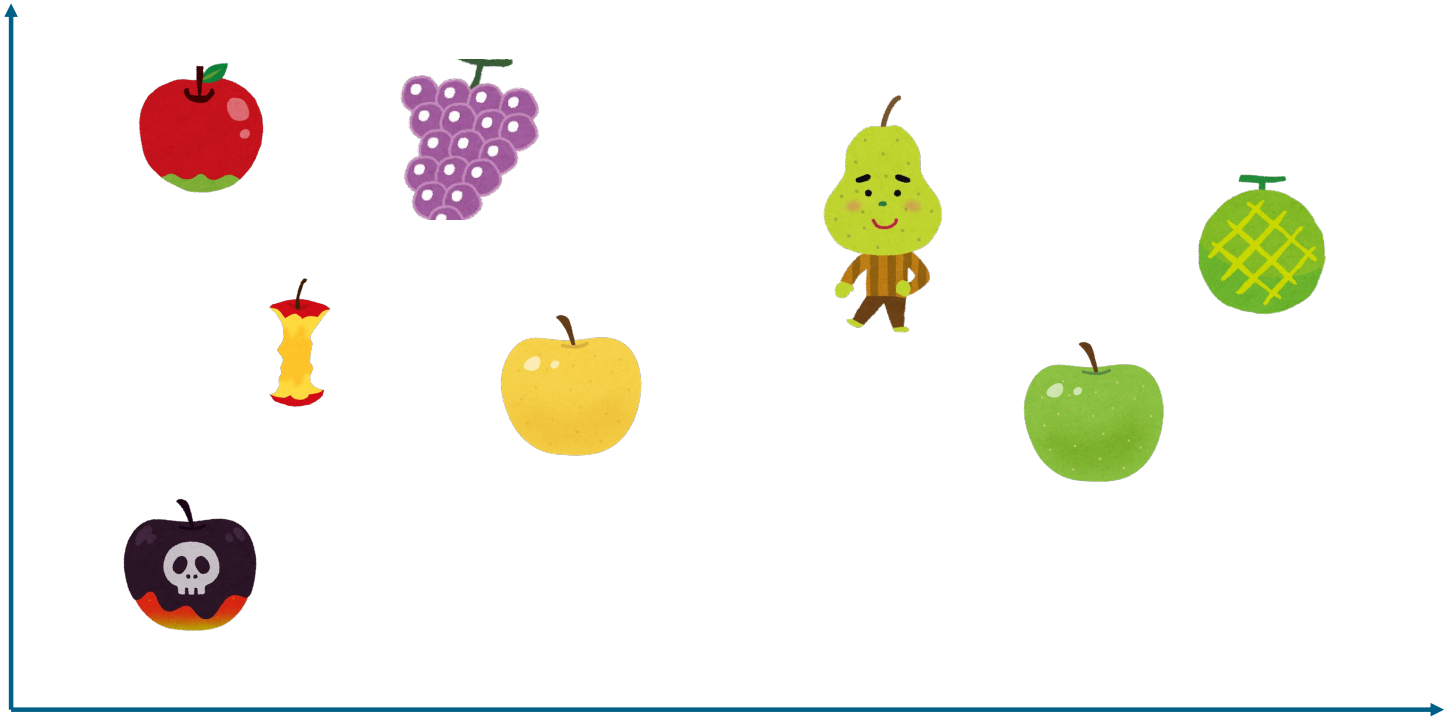
特徴量

$$\begin{pmatrix} 0.8 \\ 1.0 \\ 0.2 \\ 0.2 \end{pmatrix}$$

特徴ベクトル

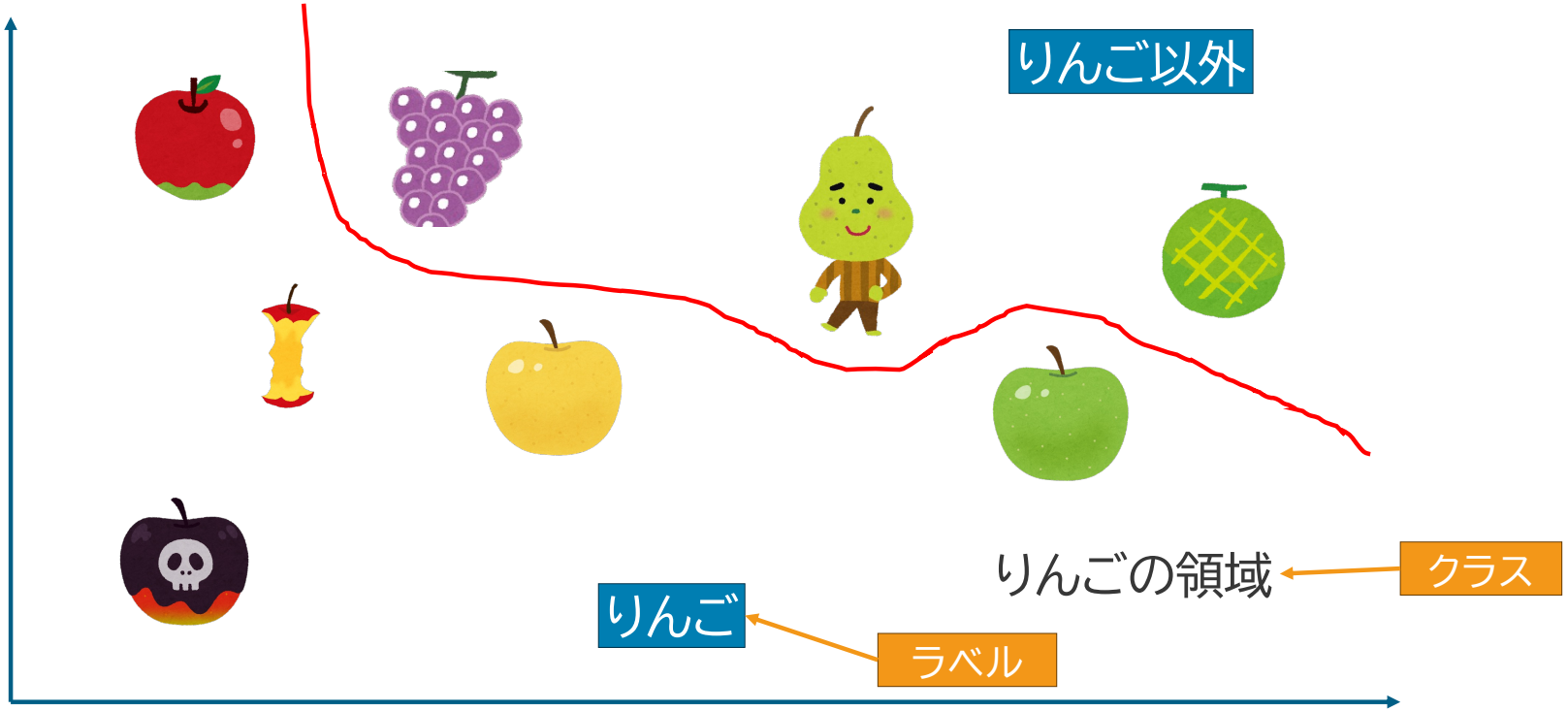
分類の境界を探す

- さきほどの特徴ベクトルを図にしてみる



境界を探す

- りんごかりんごじゃないかを示す境界線を自動的にひく



機械学習とは

- 人間が自然に行っている学習能力と同様の機能を計算機で実現しようとする技術
  - データの集合から何らかのルールや法則を計算機に学習させる

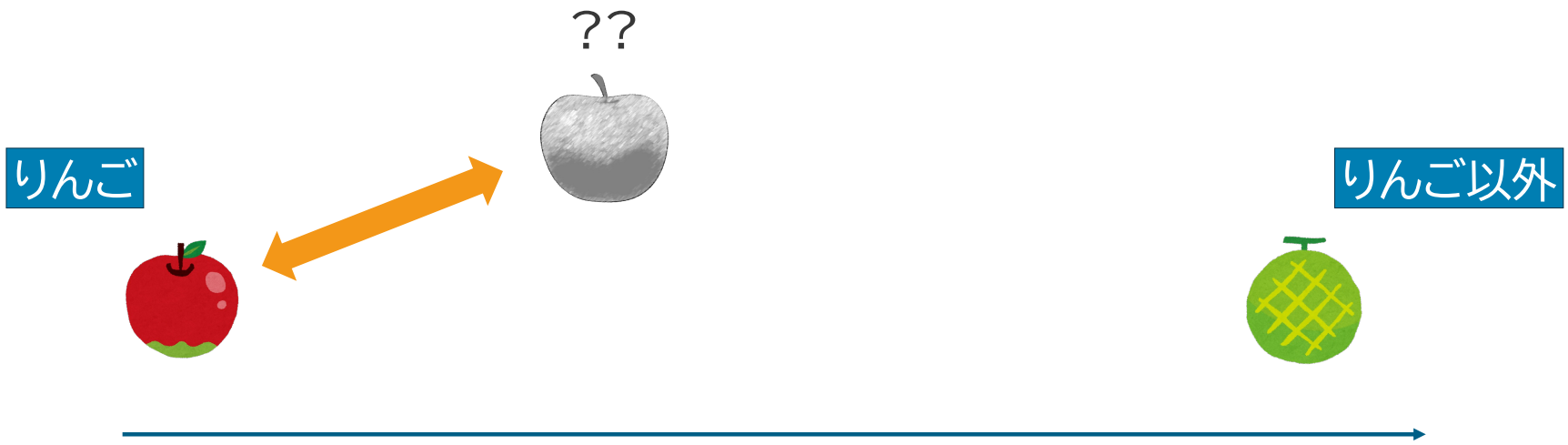
機械学習を試してみよう

https://teachablemachine.withgoogle.com

K最近傍法

機械学習(scikit-learn)応用

最近傍法

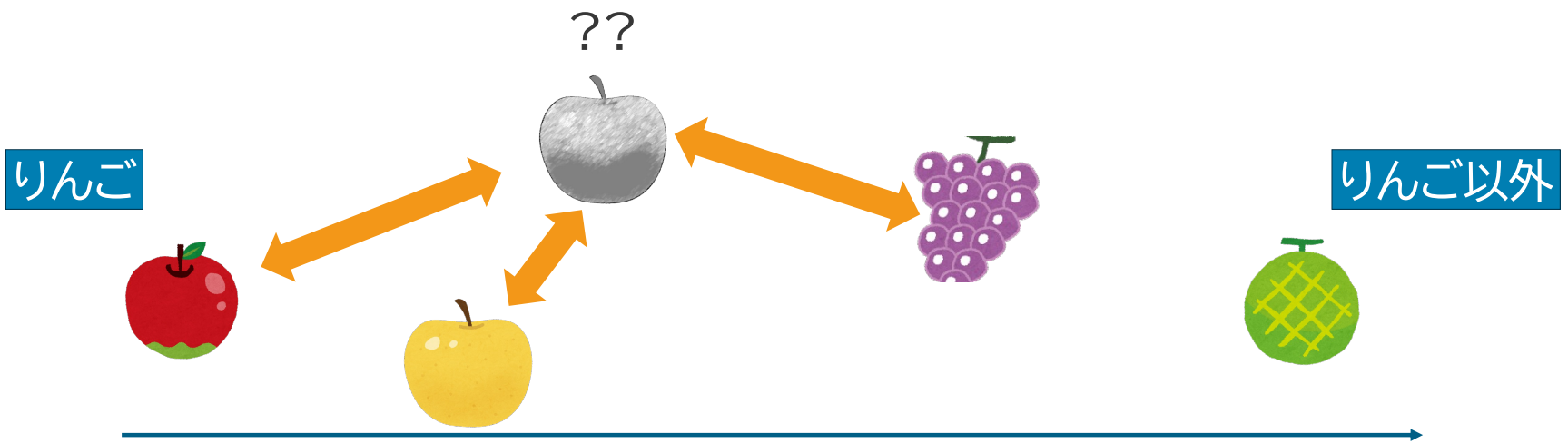


りんごに近い→「りんご」と判定する

入力データに最も近い教師データを出力する

K最近傍法

- 入力データの近くのk個の教師データで，どちらが多いかで決める方法



近いもの3個のうちりんご2個，りんご以外1個→「りんご」と判定する

データセットを読み込む

- IrisDataset
  - 3種類のアヤメを次の特徴で分類できるか？
    - ガクの長さ
    - ガクの幅
    - 花びらの長さ
    - 花びらの幅

- データセットの読み込み

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

- データセットを確認
  - iris.data : 特徴
  - iris.target : アヤメの種類

データを分割する

- テストデータと学習データ
  - 学習用のデータを入力して正解するのは当たり前
  - テスト用と学習用にデータを分けて学習データだけを使って学習
  - 学習に使わなかったデータでテストをしてどれだけ当てられるかで性能測定
- 分割方法

```
from sklearn.model_selection import train_test_split
(x_train, x_test, y_train, y_test)
        = train_test_split(iris.data, iris.target, train_size=0.8)
```

モジュール使用の準備

- インポート
  from sklearn.neighbors import KNeighborsClassifier

- 初期化
  nn = KNeighborsClassigier(n_neighbors=5)
  - 近くにある何個を見るかを指定(ここでは5個)

関数を使う

- fit関数
  - 入力と正解ラベルのペアを与えてモデルを学習

- predict関数
  - 入力から対応するラベルを予測

fit関数とpredict関数

- fit関数
  - nn.fit(入力, 正解ラベル)

```
nn.fit(x_train, y_train)
```

- predict関数
  - nn.predict(入力)

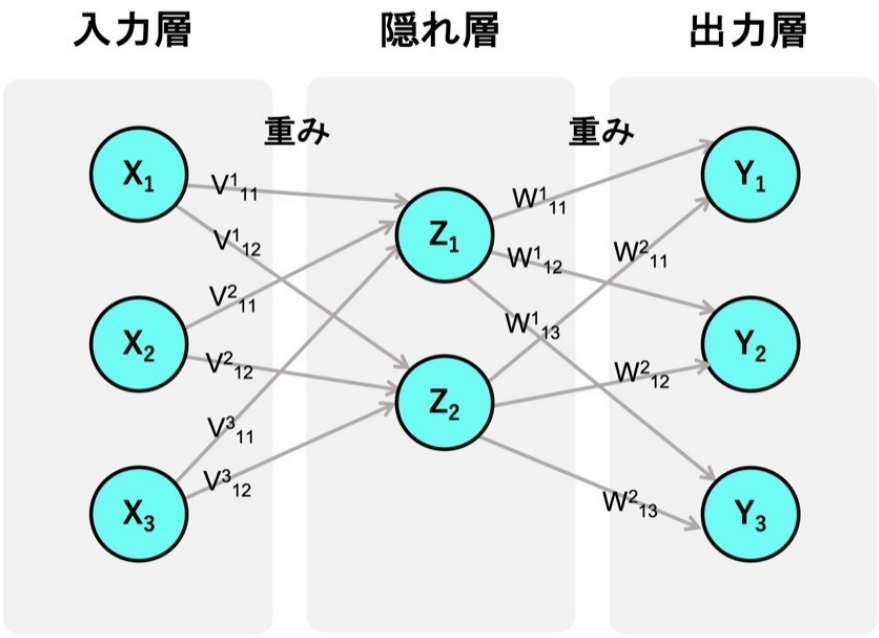
```
nn.predict(x_test)
```

ニューラルネットワーク

scikit-learn

ニューラルネットワークとは？

- 脳の神経回路網を模した機械学習技術
  - 深層学習, ディープラーニング



モジュール使用の準備

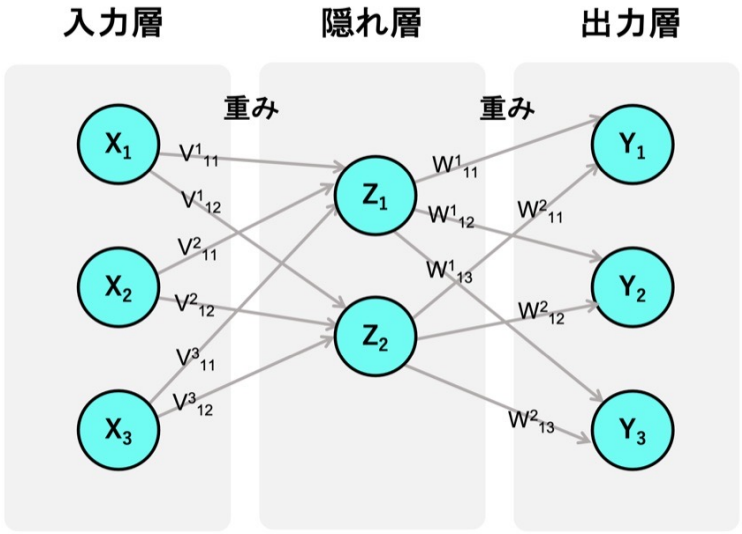
- インポート
  from sklearn.neural_network import MLPClassifier

- 初期化
  mlp = MLPClassifier(パラメータ)

  mlp = MLPClassifier(hidden_layer_sizes=(100,100,100))

パラメータ

- hidden_layer_sizes
  - 隠れ層のユニット数
- Activation
  - 活性化間数
- batch_size
  - 一回の反復で使う学習データ数
- learning_rate_init
  - 学習率の初期値
- max_iter
  - 学習の最大反復回数



fit関数とpredict関数

- fit関数
  - mlp.fit(入力, 正解ラベル)

```
mlp.fit(x_train, y_train)
```

- predict関数
  - mlp.predict(入力)

```
mlp.predict(x_test)
```

評価

どれくらい正しい予測ができたのか

Accuracy(正解率)

- 予測結果がどれくらい正しいのかを測定する

$$Accuracy = \frac{\text{正しく予測できたデータ数}}{\text{評価データ数}}$$

- インポート
  from sklearn.metrics import accuracy_score
- 使用
  accuracy_score(正しい値, 予測した値)

混同行列（Confusion Matrix）

- 予測結果をまとめた表

- インポート

```
from sklearn.metrics import confuxion_matrix,
ConfusionMatrixDisplay
```

- 使用

```
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(cm, display_labels=mlp.classes_)
disp.plot
```

課題

- タイタニック号のデータセットを分析しよう
  https://bit.ly/3Txqj38
  CSVファイルがあります
- 内容
  - 生存したかどうか(当てたいもの)
  - 名前
  - 性別(男性0女性1)
  - 年齢
  - 搭乗券のクラス
  - 同乗した兄弟・配偶者の数
  - 同乗した親・子供の数
  - 運賃

1-1～1-7

- 1-1: CSVファイルを読み込み, 中身を表示してください
- 1-2: .valuesを使ってDataFrameからndarrayへ変更してください
- 1-3: 正解ラベルを変数 y に入れてください(データの0列目)
- 1-4: 特徴量をint型に変更してください
  - 正解ラベルを入れるところの最後に .astype(dtype=int)を追加してください
- 1-5: 特徴量を変数Xに入れてください(データの2～7列目)
- 1-6: 学習データ (80%)とテストデータ (20%)に分けてください
- 1-7: .shapeを使って学習データ, テストデータの要素数を表示してください
- 1-8: 学習データの0番目の行を表示してください(特徴量とラベルをどちらも示してください)

2-1～2-4, 3-1～3-4

- K最近傍法を使って分類を行います
- 2-1から2-4ではn_neighborsを1に設定してください
- 3-1から3-4ではn_neighborsを3に設定してください

- 2-1, 3-1: KNeiborsClassifierを初期化してください
- 2-2, 3-2: 訓練データからモデルを学習してください
- 2-3, 3-3: テストデータのラベルを予測し, y_predに入れてください
- 2-4, 3-4: テストデータを使って正解率を求めてください

4-1～4-4

- ニューラルネットワークを使って分類を行います.
  - hidden_layer_size: (100,100)
  - activation: relu
  - batch_size: 255
  - max_iter: 300
- 4-1: MLPClassifierを初期化してください
- 4-2: 訓練データからモデルを学習してください
- 4-3: テストデータのラベルを予測し, y_predに入れてください
- 4-4: テストデータを使って正解率を求めてください

5

以下は, 最近傍法で混同行列を表示する例です.

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

# 混合行列を算出
nn_cm = confusion_matrix(y_test, nn.predict(X_test))

# 混合行列を表示
nn_cm_disp = ConfusionMatrixDisplay(nn_cm, display_labels=nn.classes_)
nn_cm_disp.plot()
```

- 5-1: K近傍法で混同行列を計算し, 描画してください
- 5-2: ニューラルネットワークで混同行列を計算し, 描画してください

自由課題

- 軸受診断データがあります. これを分析してください

https://bit.ly/3vcBBPW