

医療とAI・ビッグデータ応用

①MNISTの読み込みと加工

本スライドは、自由にお使いください。
使用した場合は、このQRコードからアンケート
に回答をお願いします。



統合教育機構
須藤毅顕

医療とAI・ビッグデータ入門

- pythonの基本
- 機械学習とは（糖尿病データ、乳がんデータ）
- 深層学習とは（肺のレントゲン画像）

体験してもらおう（ipynbファイルの実行メイン）

医療とAI・ビッグデータ応用

深層学習

- MLP（多層パーセプトロン）
- CNN（畳み込みニューラルネットワーク）

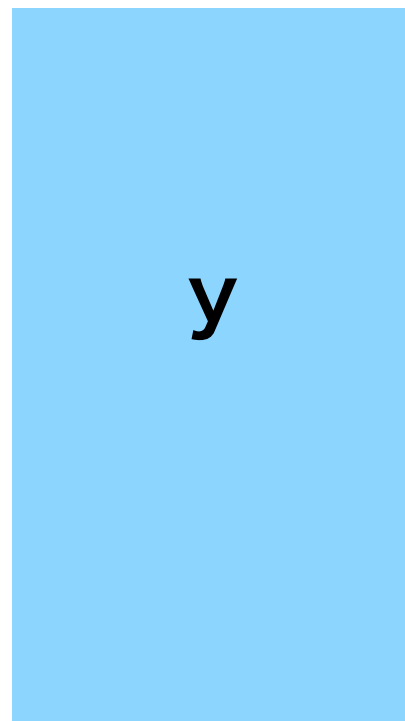
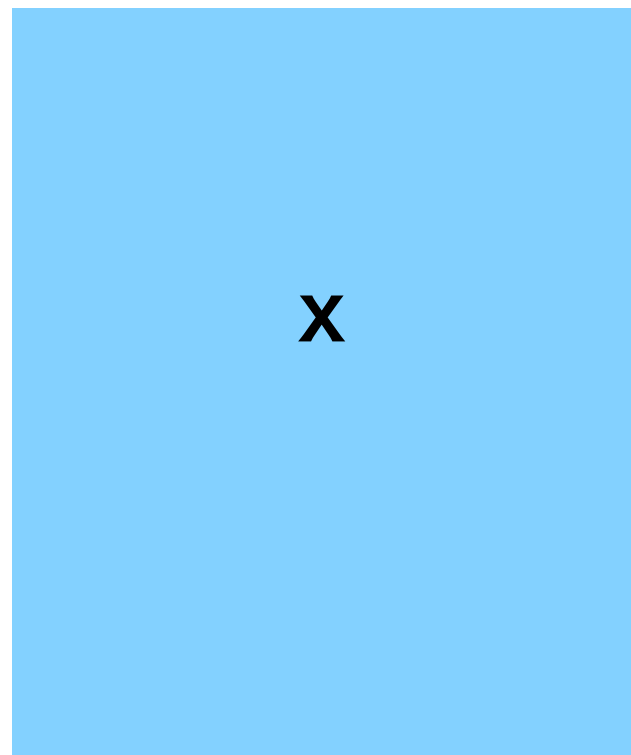
理解してもらおう（自分でタイピング、グループ演習）

深層学習(教師あり機械学習)の復習

データを用意する

x(特徴量データ)

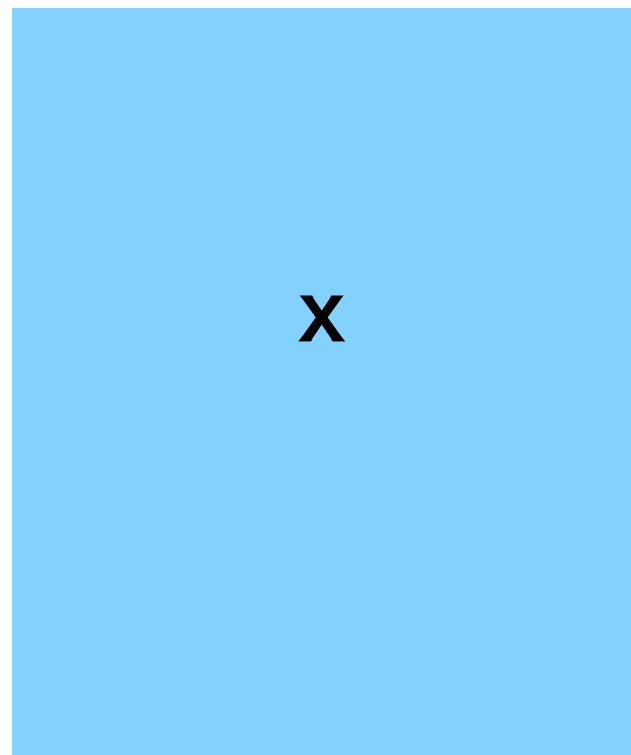
y(正解データ)



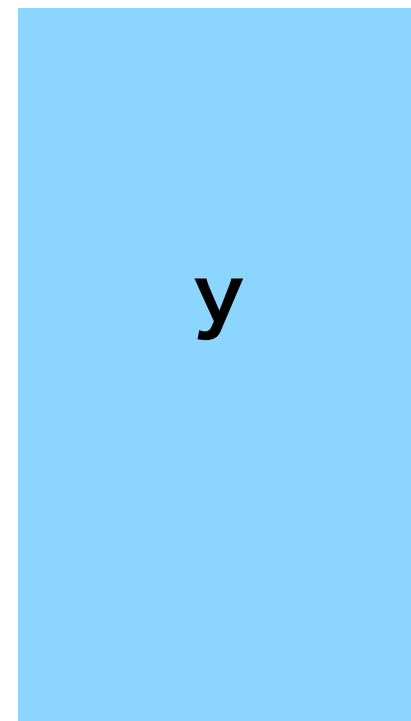
深層学習(教師あり機械学習)の復習

データを用意する

x(特徴量データ)

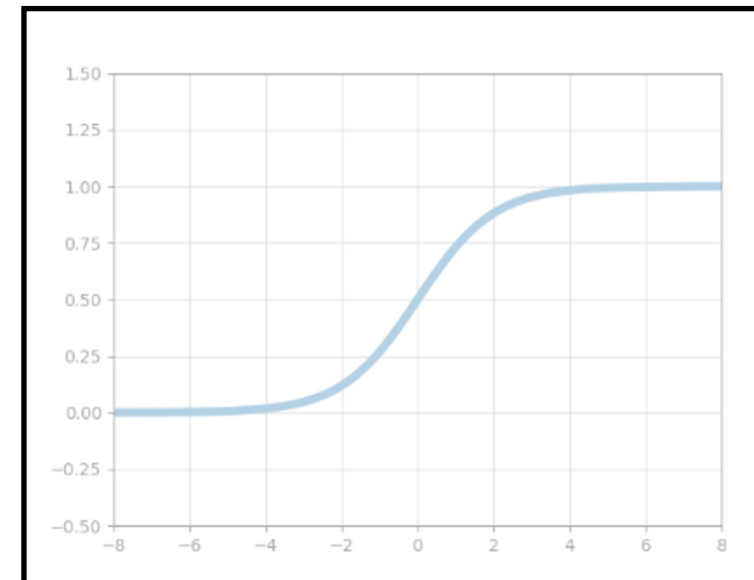


y(正解データ)

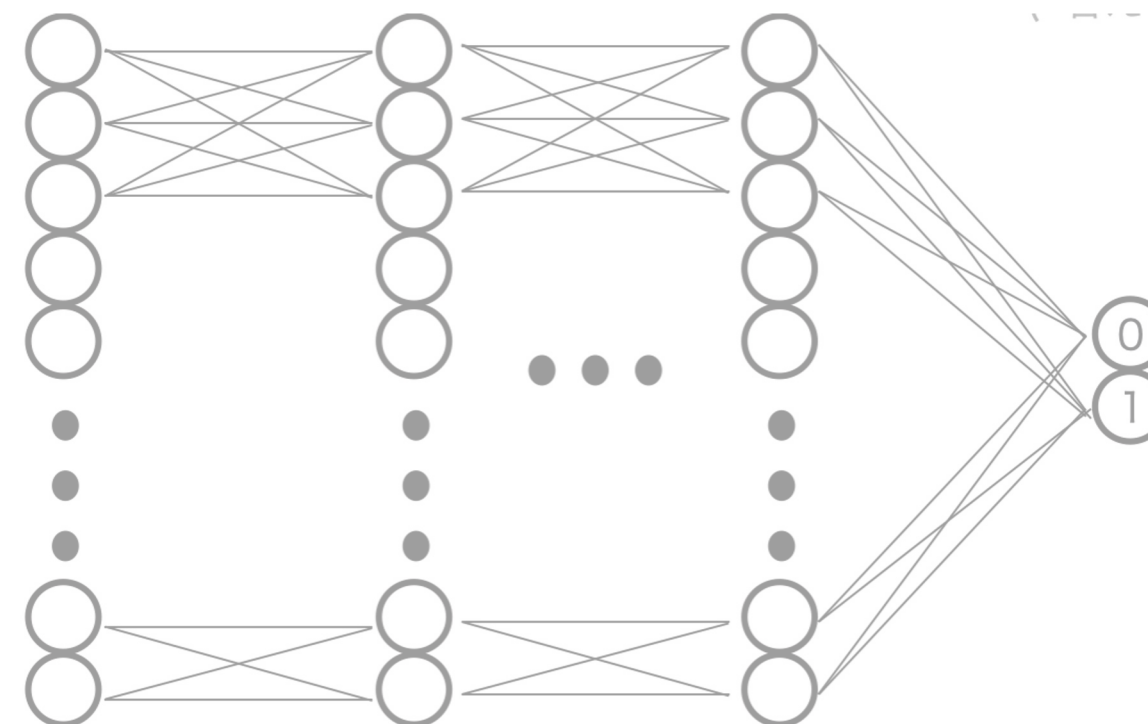


学習させる

ロジスティック回帰分析



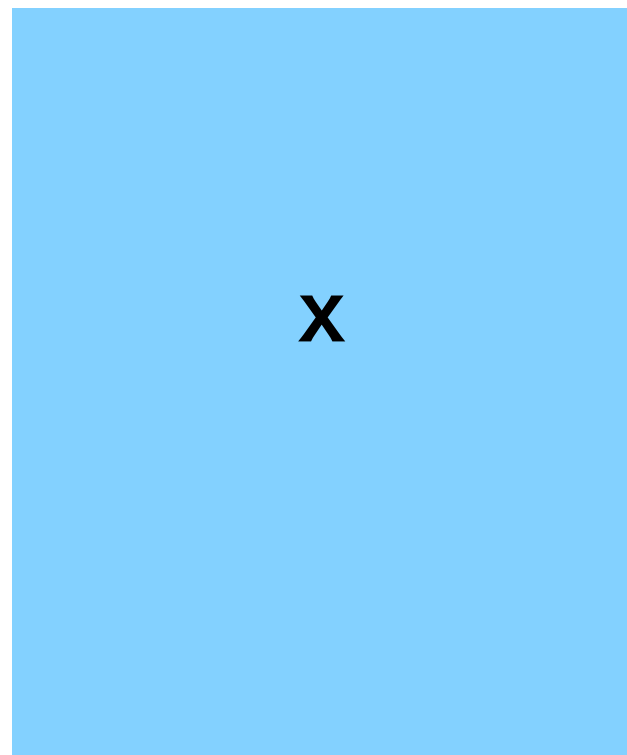
ニューラルネットワーク



深層学習(教師あり機械学習)の復習

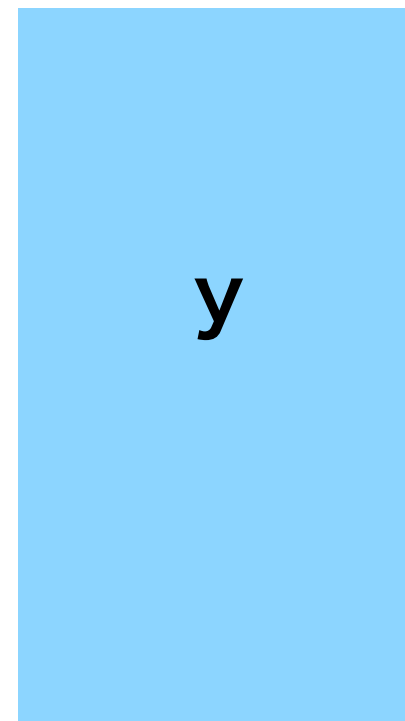
データを用意する

x(特徴量データ)



x

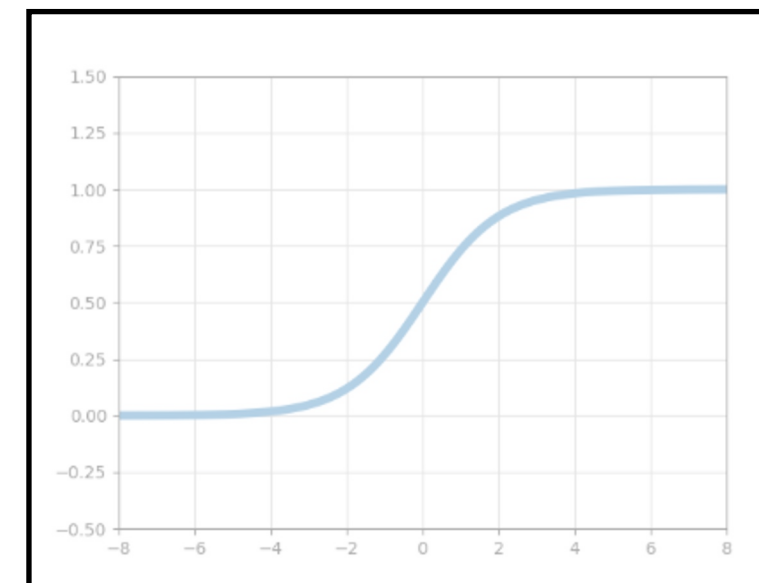
y(正解データ)



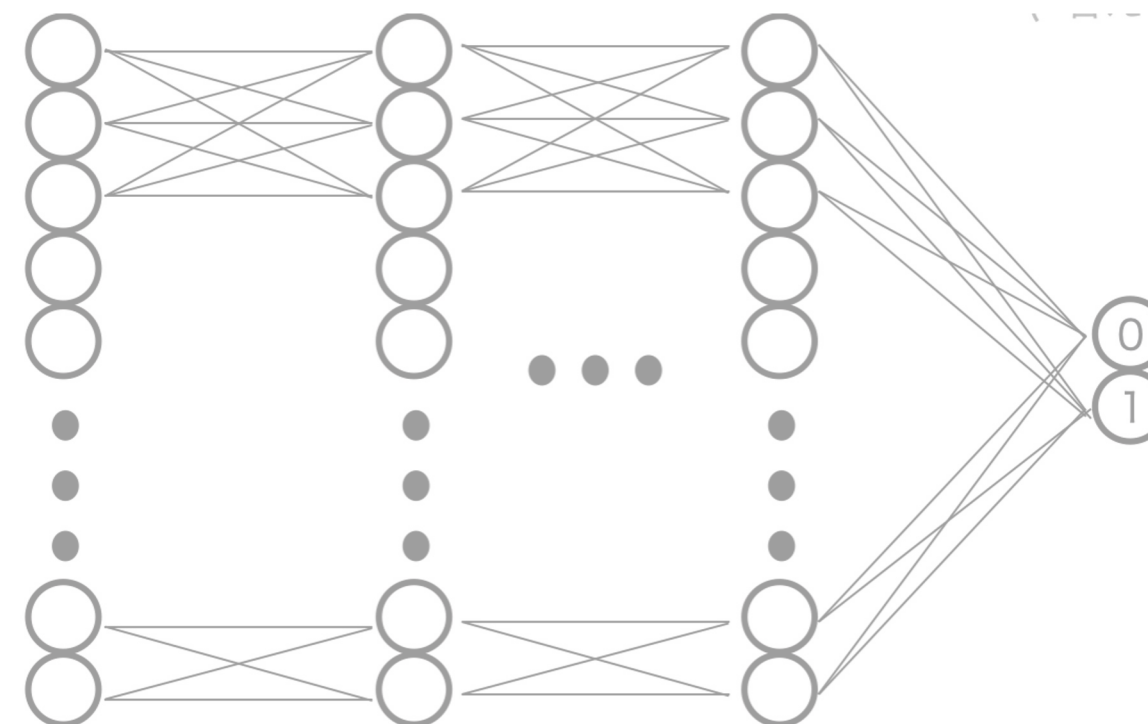
y

学習させる

ロジスティック回帰分析



ニューラルネットワーク



評価する
(分類、予測など)

病気か否か

犬か猫か

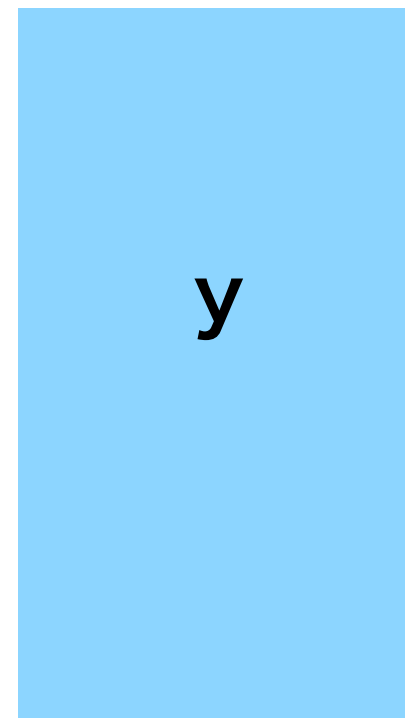
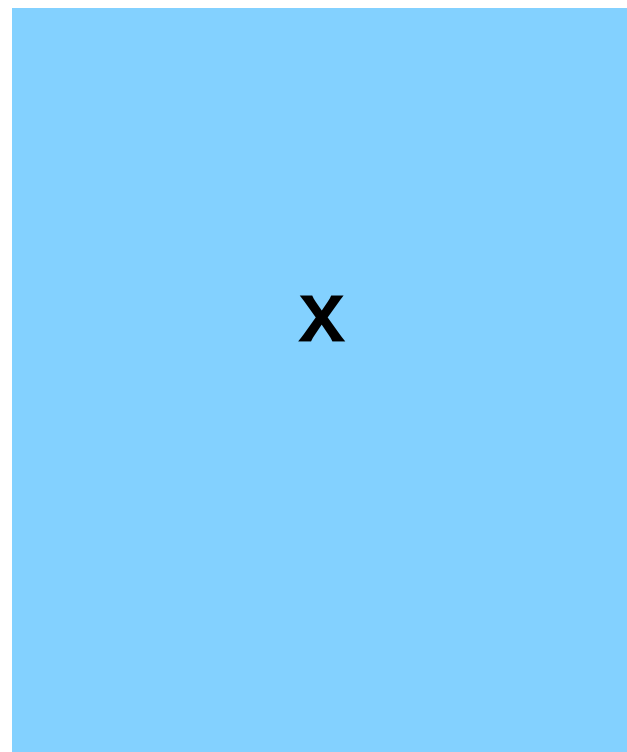
深層学習(教師あり機械学習)の復習

データを用意する

=特徴量 x と正解 y を
配列の形で整える

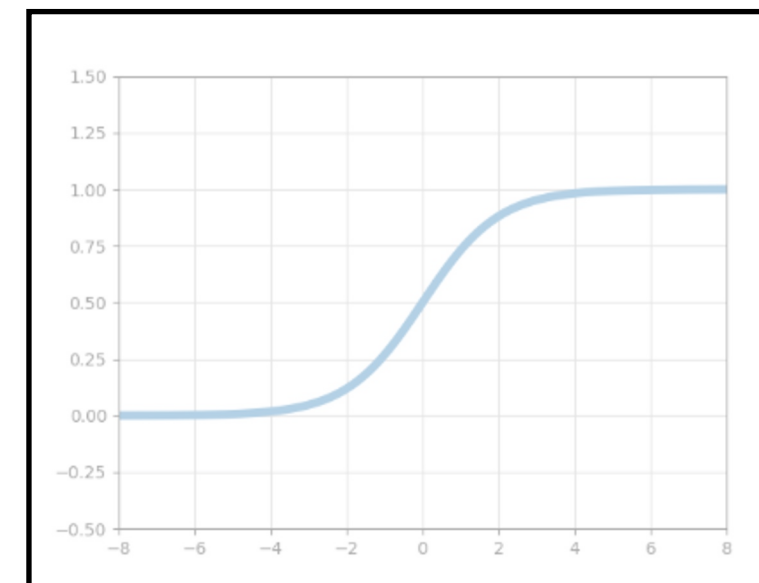
x (特徴量データ)

y (正解データ)

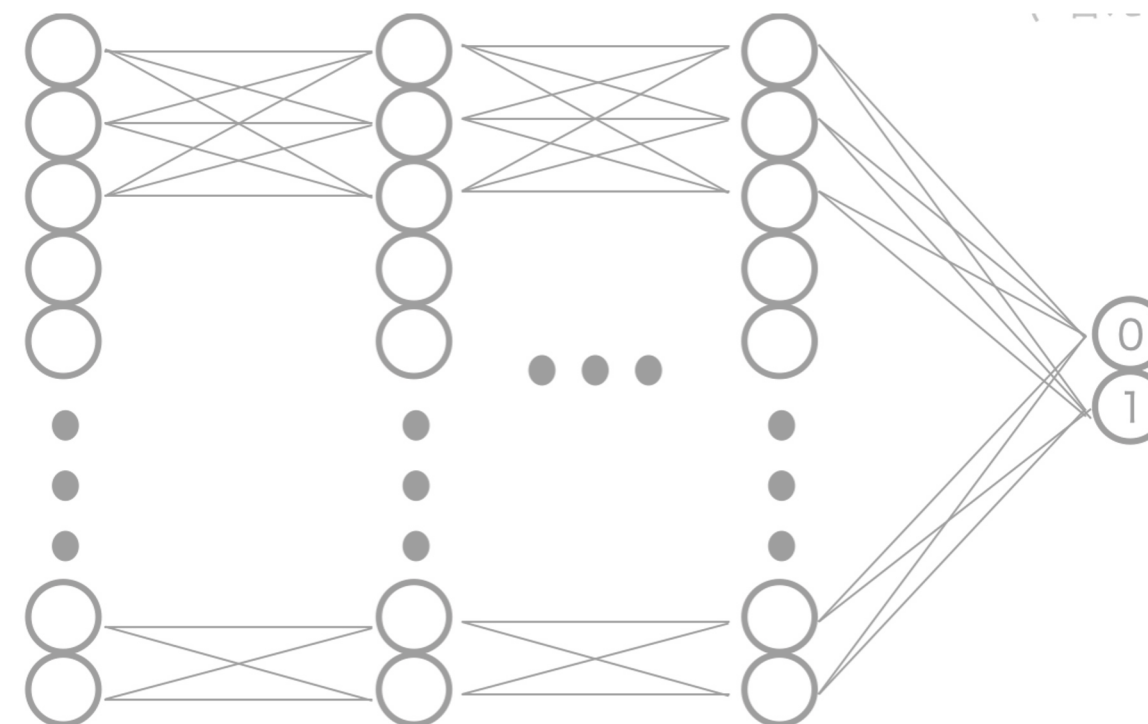


学習させる

ロジスティック回帰分析



ニューラルネットワーク



評価する
(分類、予測など)

病気か否か
犬か猫か

入門で扱った表形式のデータ

糖尿病データ(scikit-learn)

特徴量(患者の臨床情報)

	age	sex	bmi	bp	s1	s2	s3	s4	s5	s6
0	59.0	2.0	32.1	101.00	157.0	93.2	38.0	4.00	4.8598	87.0
1	48.0	1.0	21.6	87.00	183.0	103.2	70.0	3.00	3.8918	69.0
2	72.0	2.0	30.5	93.00	156.0	93.6	41.0	4.00	4.6728	85.0
3	24.0	1.0	25.3	84.00	198.0	131.4	40.0	5.00	4.8903	89.0
4	50.0	1.0	23.0	101.00	192.0	125.4	52.0	4.00	4.2905	80.0
...

正解(重症度)

0	151.0
1	75.0
2	141.0
3	206.0
4	135.0
	...

乳がんデータ(scikit-learn)

特徴量(がんの情報)

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800
...

正解(良性/悪性)

0	0
1	0
2	0
3	0
4	0
	..



特徴量(1部or全部)と正解をセットで学習させる
(モデル名).fit(特徴量, 正解)



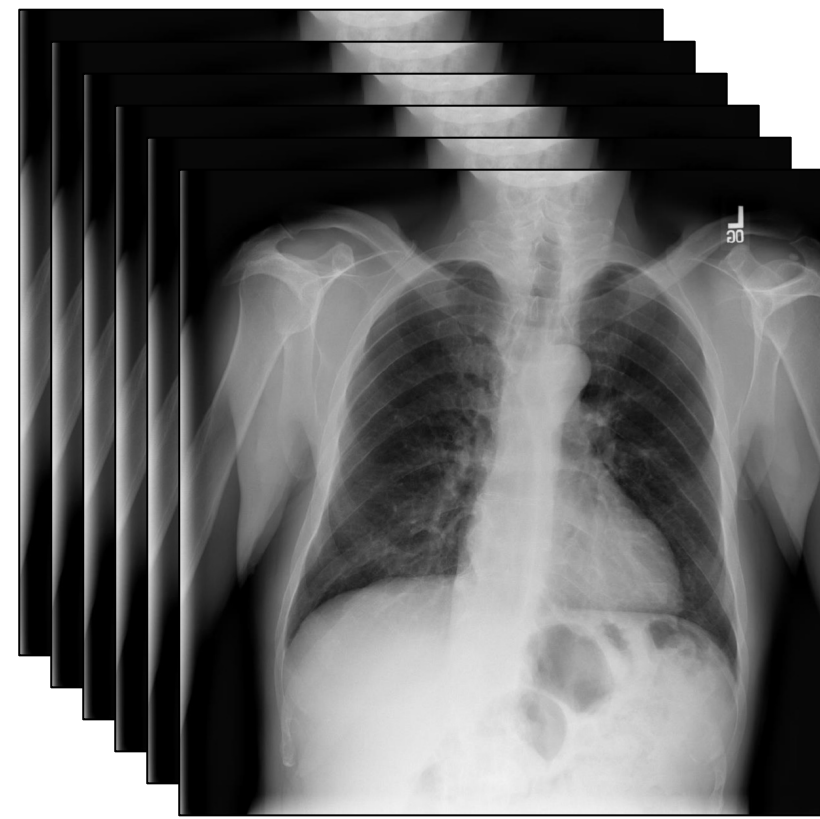
学習したモデルの評価、未知のデータでの予測/分類

scikit-learnのこれらのデータはあらかじめデータは配列になっていた

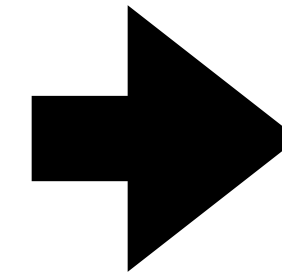
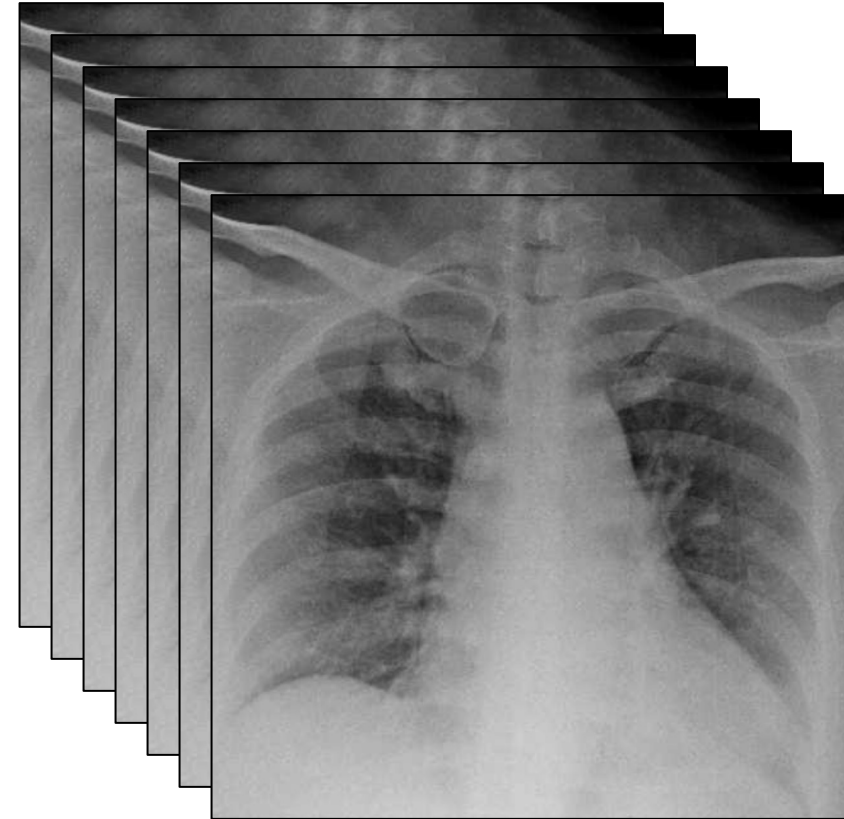
入門で扱った画像データ

肺のX線画像

健康(1)



肺炎(0)



配列データ

特徴量

x_train

```
[150, 70, 60],  
[120, 40, 35],  
[144, 45, 40],  
[162, 56, 50],  
[98, 40, 32],  
[128, 59, 35],  
[155, 77, 45]]
```

正解

y_train

```
[1],  
[0],  
[1],  
[1],  
[0],  
[0],  
[1]]
```



特徴量と正解をセットで学習させる (モデル名).fit(特徴量, 正解)



学習したモデルの評価、未知のデータでの分類

入門では画像を読み込んで配列に置き換えていた

今回はkerasというライブラリに用意されている画像セットを使います

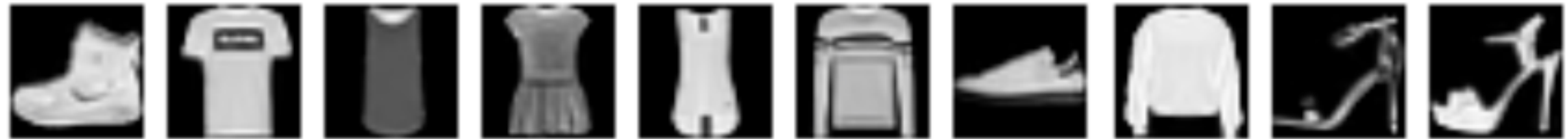
(あらかじめ配列になっています)

MNIST：0~9の文字画像のデータ



FASHION-MNIST：白黒の洋服の画像データ

0 : T-shirt/top、1 : Trouser、2 : Pullover、3 : Dress、4 : Coat、5 : Sandal
6 : Shirt、7 : Sneaker、8 : Bag、9 : Ankle boot



CIFAR10

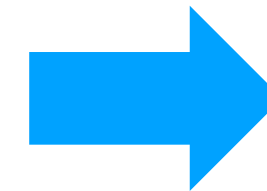
0 : airplane、1 : automobile、2 : bird、3 : cat、4 : deer、5 : dog
6 : frog、7 : horse、8 : ship、9 : truck



Keras: 深層学習用のライブラリ(入門でも使用)

colaboratoryを準備しよう(入門の復習)

「Google Colaboratory」で検索



「Colaboratoryへようこそ」をクリック



colaboratoryを準備しよう(入門の復習)

“ノートブックを新規作成”で開始

ノートブック：colabotoryで作られるpythonの実行ファイル(ipynbファイル)

ノートブックを開く

例 >

最近 >












Google ドライブ >

GitHub >

アップロード >

ノートブックを検索

🗑️

タイトル	最終閲覧 ↓	最初に開いた日時 ↑↓	
 Colaboratory へようこそ	18:27	2020年11月2日	
 Untitled32.ipynb	18:24	18:24	 
 応用2024_1回目草案_20240412.ipynb	18:18	4月12日	 
 Untitled31.ipynb	14:10	14:10	 

+ ノートブックを新規作成

キャンセル

”例”
サンプル集(時間あるときに)

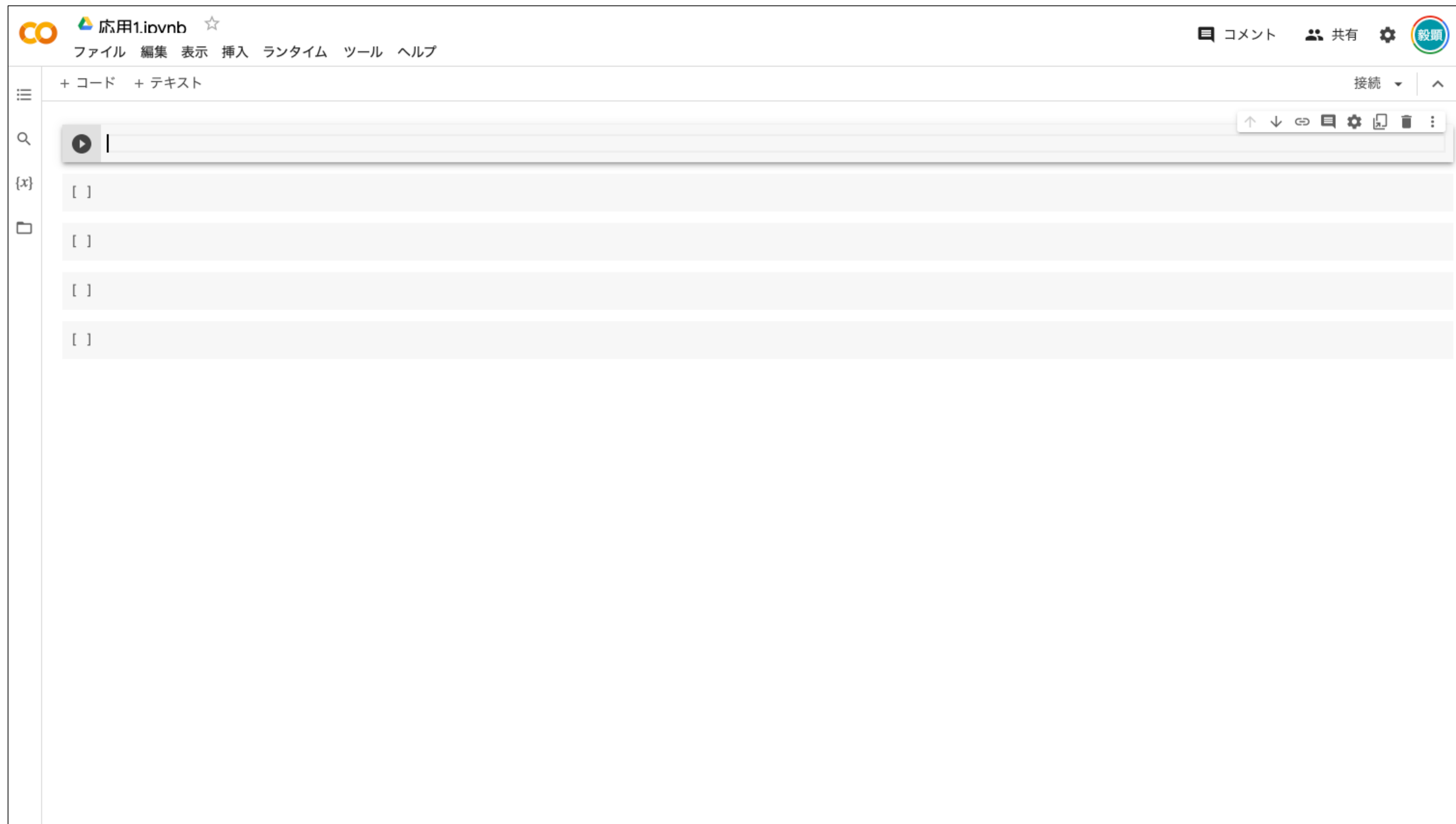
”最近”
最近実行したノートブック

”Google ドライブ”
過去に作成したノートブック

”GitHub(バージョン管理アプリ)”
GitHub上のノートブック

”アップロード”
Google ドライブ外のノートブック

colaboratoryを準備しよう



Pythonの復習

Aという変数に5を代入してAを出力してください

Pythonの復習

```
A = 5
```

Aに5を代入

```
print(A)
```

print(A)→()の中身を出力

```
5
```

```
A
```

Aだけで実行→中身を出力

```
5
```

```
A = 5
```

```
A
```

セル内でEnterで改行

```
5
```

```
A = 6
```

```
A
```

Aに6を上書き

```
6
```


Pythonの復習

Bに1から5の連続した要素からなるリストを代入してください

Bの中から3の値を出力してください

Pythonの復習

Bに1から5の連続した要素からなるリストを代入してください

Bの中から3の値を出力してください

```
B = [1, 2, 3, 4, 5]
```

```
B
```

```
[1, 2, 3, 4, 5]
```

```
B[2]
```

```
3
```

リストは[値, 値, ...]の形式

取り出すときは変数名[番号]

この番号をインデックスという
インデックスは0から始まる

MNISTデータを扱ってみよう



```
from keras.datasets import mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

この2行をセルに書き込んで実行してみよう

MNISTデータを扱ってみよう



The screenshot shows a Jupyter Notebook interface with a top bar containing the logo, the name '応用1.ipynb', and a star icon. Below the bar is a menu with 'ファイル', '編集', '表示', '挿入', 'ランタイム', 'ツール', 'ヘルプ', and a status message 'すべての変更を保存しました'. The left sidebar has icons for a menu, search, variables, and a file explorer. The main area displays a code cell with a play button icon, a green checkmark, and a '6 秒' (6 seconds) execution time. The code in the cell is:

```
from keras.datasets import mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

Below the code, the output of the cell is shown, which is a message indicating the data is being downloaded from a specific URL. This output is circled in red in the image:

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [=====] - 0s 0us/step
```

Below the output, there are four empty list boxes, each containing '[]'.

実行するとmnistのデータがダウンロードされて
変数に代入されます

ライブラリの色んな読み込み方

```
import keras
```

kerasを読み込む

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

pandasを読み込んでpdと省略して使う
numpyを読み込んでnpと省略して使う

```
import matplotlib.pyplot as plt
```

matplotlibのpyplotを読み込んでpltと省略して使う

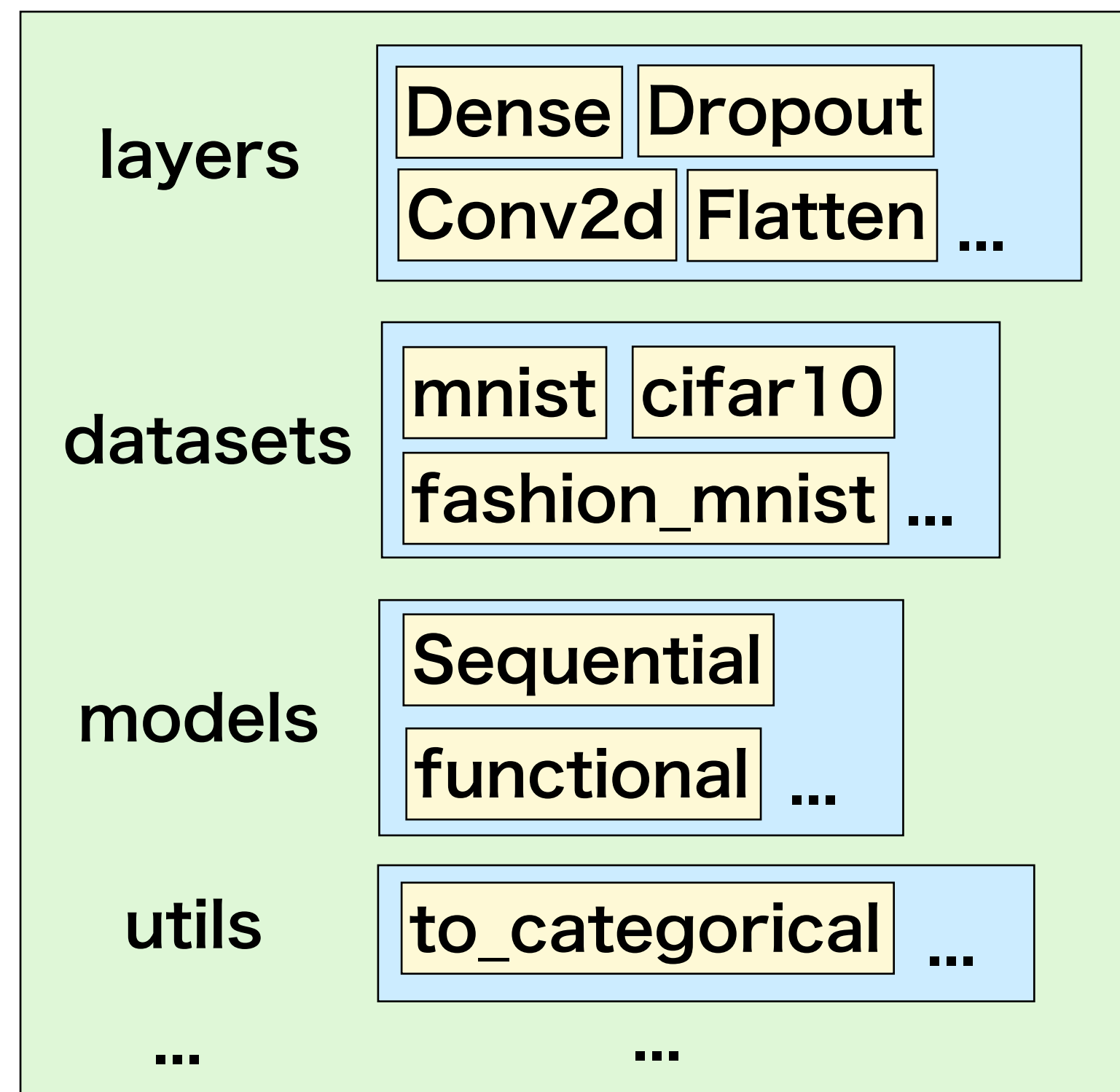
```
from keras.datasets import mnist
```

kerasのdatasetsの中のmnistを読み込む

```
from keras.datasets import mnist
```

kerasの中のdatasetsの中のmnistという関数を読み込む

keras




```
from keras.datasets import mnist
```

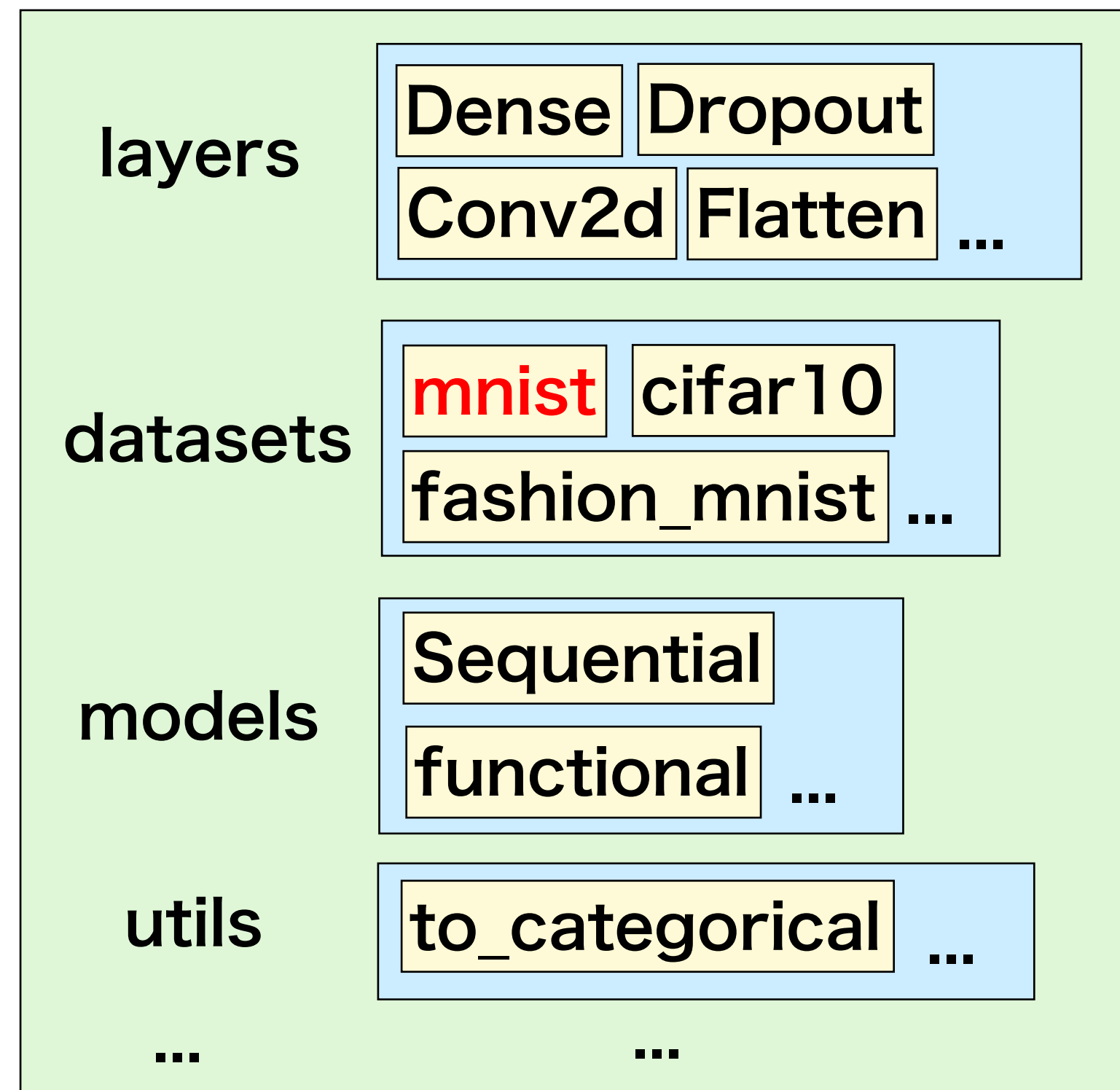
kerasの中のdatasetsの中のmnistという関数を読み込む

```
from keras.datasets import mnist
```

使う際には

```
mnist.~~( ここではmnist.load_data() )
```

keras



```
from keras.datasets import mnist
```

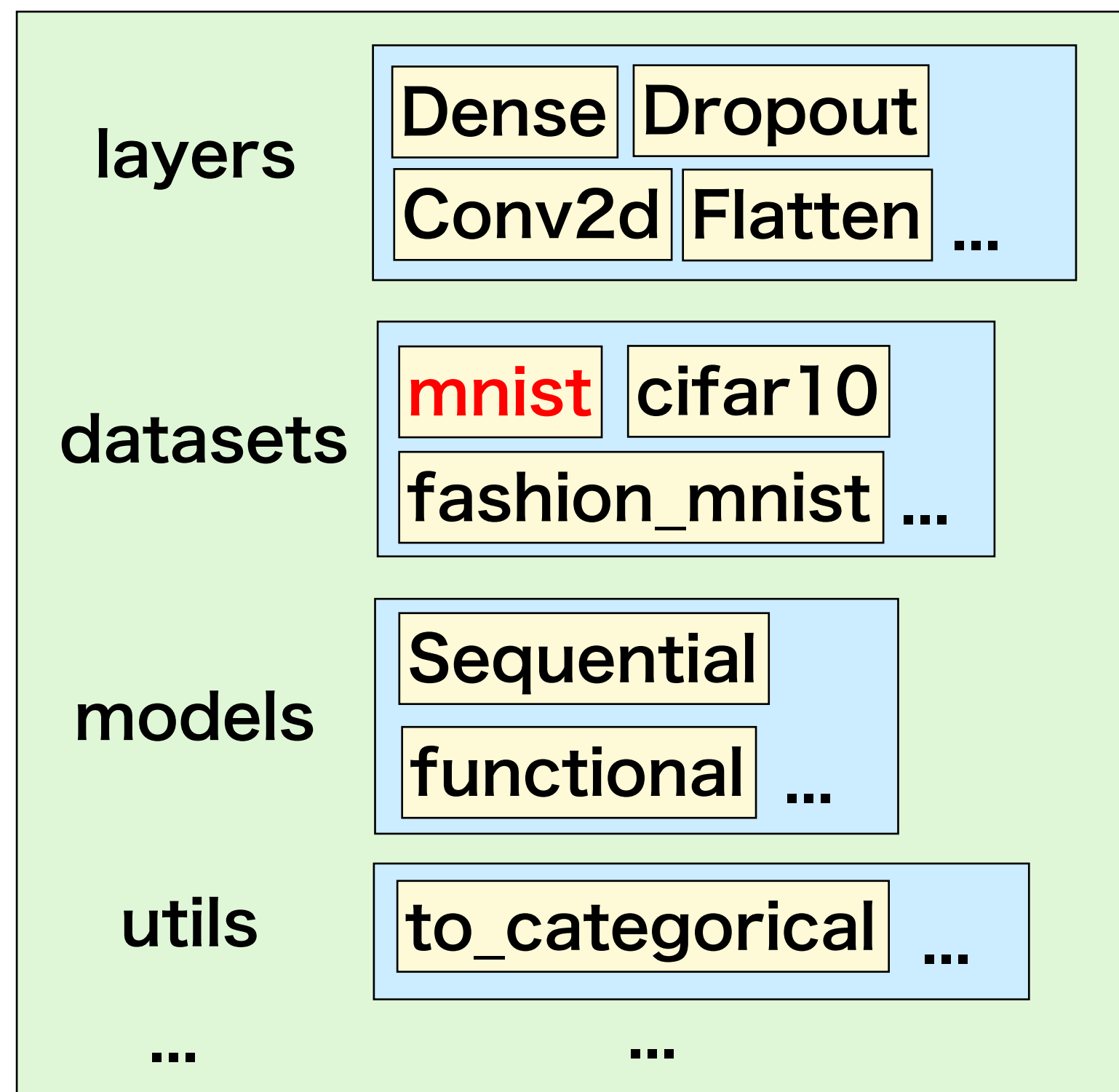
kerasの中のdatasetsの中のmnistという関数を読み込む

```
from keras.datasets import mnist
```

使う際には

mnist.~~(ここではmnist.load_data())

keras



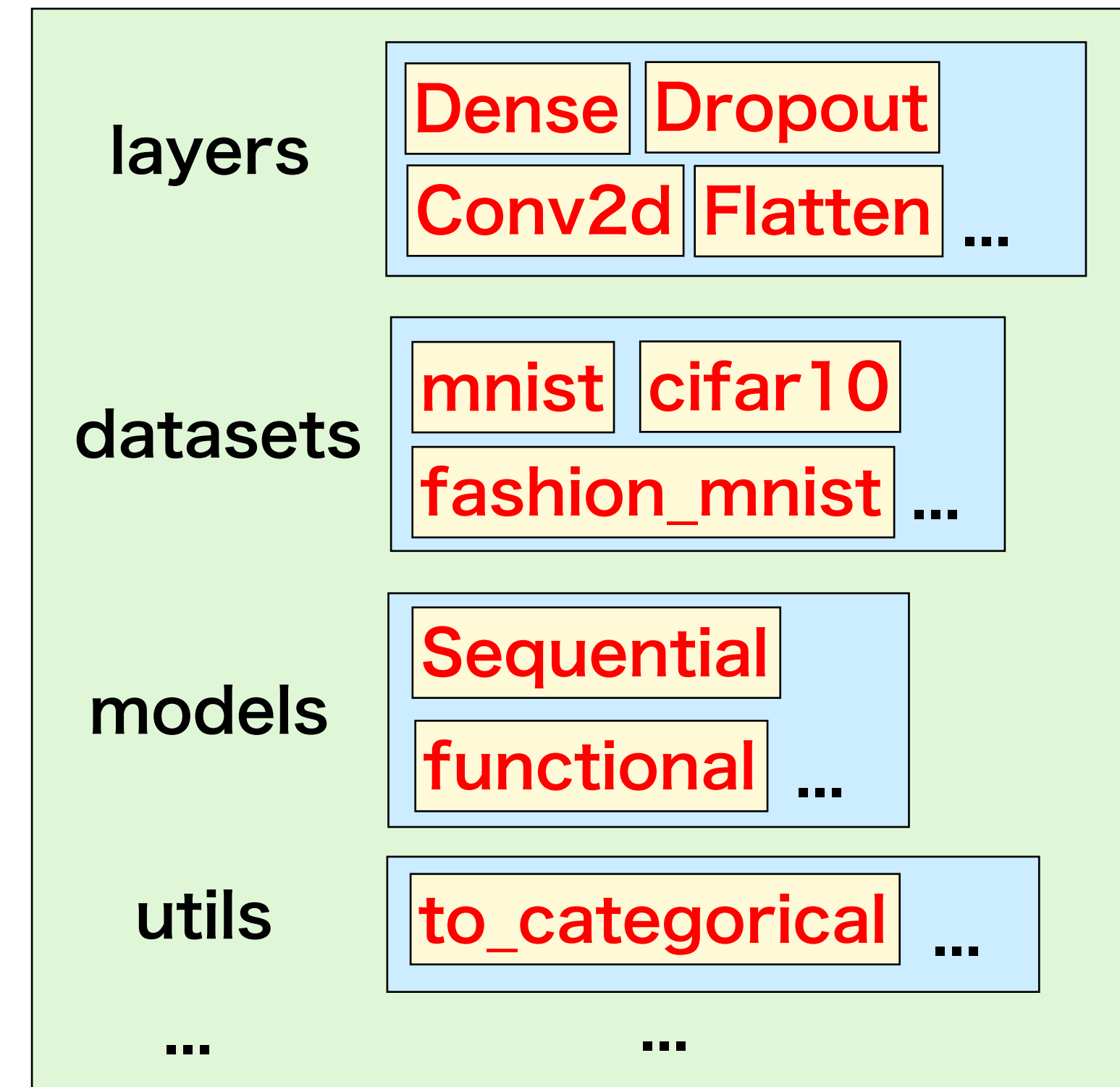
mnistしか使えないがシンプルに書ける

```
import keras
```

使う際には**keras.datasets.mnist**.~~

(ここでは**keras.datasets.mnist.load_data()**)

keras

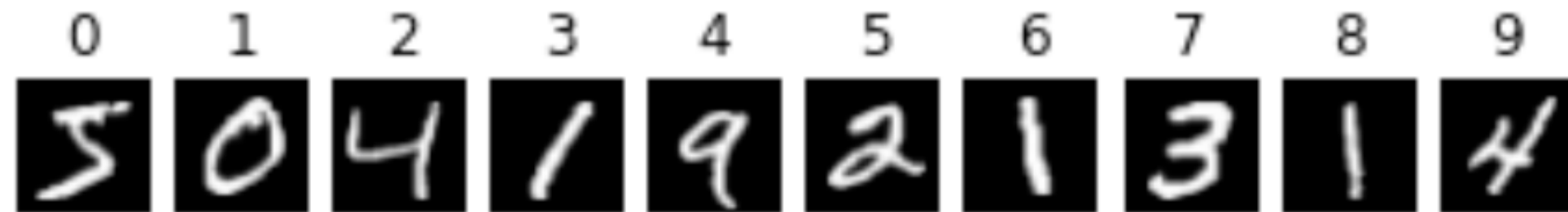


全て使えるが書くのが面倒

mnist.load_data()

mnistのデータを読み込む

MNIST : 0~9の文字画像のデータ



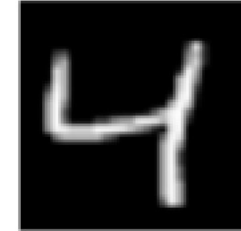
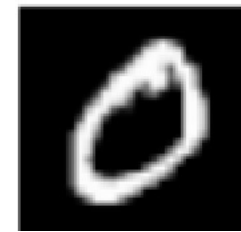
実行すると、
学習用の画像データ
学習用の正解データ
テスト用の画像データ
テスト用の正解データ
を配列(数値)の形式で返してくれる

実行すると左の変数に値が代入されること
“返す”と言います

学習用

画像

60000個



⋮



正解

60000個

5

0

4

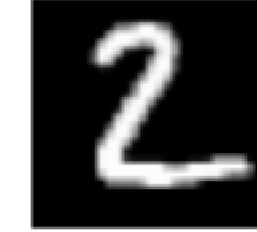
6

8

テスト用

画像

10000個



⋮



正解











10000個

7

2

6

`(x_train, y_train), (x_test, y_test) = mnist.load_data()`

60000個	60000個	10000個	10000個
	5		7
	0		2
	4	⋮	
	1		5
⋮			6
	6		
	8		

実行すると、
学習用の画像データ
学習用の正解データ
テスト用の画像データ
テスト用の正解データ
を配列(数値)の形式で返してくれる

`x_train` : 60000枚の画像の配列データ
`y_train` : 60000枚の正解の数字の配列データ
`x_test` : 10000枚の画像の配列データ
`y_test` : 10000枚の正解の数字の配列データ

変数の中身を見てみる

```
▶ x_train  
⇒ array([[0, 0, 0, ..., 0, 0, 0],  
         [0, 0, 0, ..., 0, 0, 0],  
         [0, 0, 0, ..., 0, 0, 0],  
         ...,  
         [0, 0, 0, ..., 0, 0, 0],  
         [0, 0, 0, ..., 0, 0, 0],  
         [0, 0, 0, ..., 0, 0, 0]])
```

```
[6] y_train  
    array([5, 0, 4, ..., 5, 6, 8], dtype=uint8)
```

array([])はnumpy配列という配列
(リスト[]と似ているので注意)

x_trainは何やら[]が三重になっている([[[]])

y_trainは何やら[]が一重になっている([])

numpy配列を理解する


```
import numpy as np
x = np.array([1,2,3,4])
```

`import numpy as np`

numpyというライブラリをnpと省略して読み込む

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

`[1,2,3,4]`というnumpy配列を返す

xの中身を確認

```
x
array([1, 2, 3, 4])
```

`array([])`なのでnumpy配列

numpy

abs, add, all, allclose,
amax, amin, any, arange,
argmax, argmin, argpartition,
argsort,
argwhere, around, array,
asarray, astype, clip, concat,
cos, e, floor, log, matmul, max,
mean, min,
ndarray, ones, pi, pow, power,
ravel, reshape, resize, shape,
sin, sqrt, square, squeeze,
stack, std, sum, tan, tanh,
transpose, vstack,
where, zeros, ...

numpy(=np)はほぼ全ての機能を
np.~~~で使えるように設計されている


```
x.shape
```

```
(4,)
```

numpy配列.shape

配列の形状を確認する

配列の形状

(n,) は n個の要素からなる1次元の配列

(n, m) は n×m個の要素からなる2次元の配列

(n, m, l) は n×m×l個の要素からなる3次元の配列

**xはnumpy配列なのでx.shapeでxの形状を確認
→(4,)なので4個の要素からなる1次元の配列**

numpy配列の各要素はインデックスで取り出せる

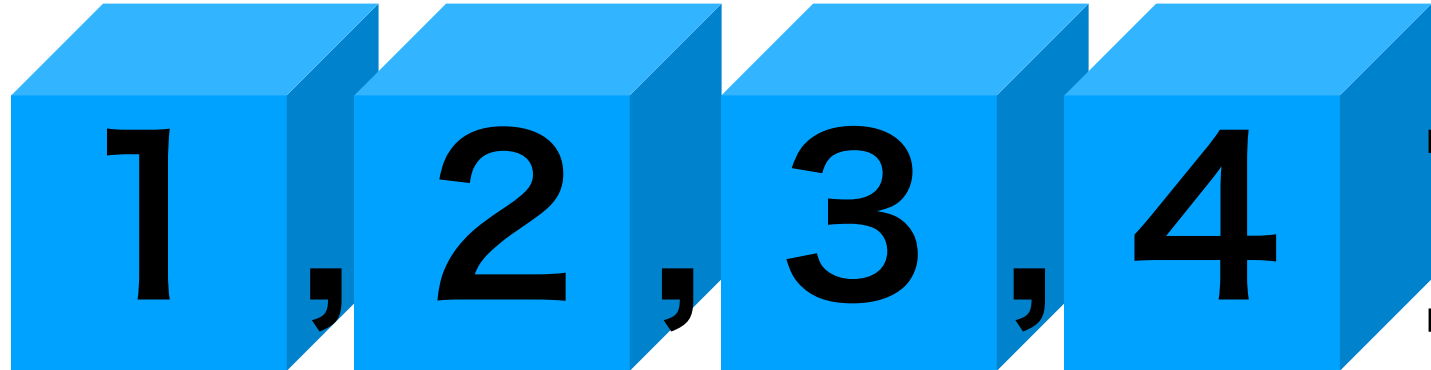
$x \rightarrow [1 \ 2 \ 3 \ 4]$

4つの要素からなる1次元配列

$x[0] \rightarrow 1$

x の1つ目 (変数名 $[n]$ の $[]$ がインデックス)

箱のイメージ

[]

$[]$ が1つなので1次元配列

$x[0] \ x[1] \ x[2] \ x[3]$

2次元配列

```
x2 = np.array([[1,2,3],[4,5,6],[7,8,9]])
```

形状は？ `x2.shape` → (3,3) **3 × 3 の2次元配列**

```
[[ 1 , 2 , 3 ],[ 4 , 5 , 6 ],[ 7 , 8 , 9 ]]
```

[]の中に[]があるので2次元配列

2次元配列

```
x2 = np.array([[1,2,3],[4,5,6],[7,8,9]])
```

形状は？ `x2.shape` → (3,3) **3 × 3 の2次元配列**

[4,5,6]を取り出すには？

```
[[ 1 , 2 , 3 ],[ 4 , 5 , 6 ],[ 7 , 8 , 9 ]]
```

2次元配列

```
x2 = np.array([[1,2,3],[4,5,6],[7,8,9]])
```

形状は？ `x2.shape` → (3,3) **3 × 3 の2次元配列**

[4,5,6]を取り出すには？

`x2[1]` → [4,5,6] **x2の2つ目の配列**

[[1 , 2 , 3],[4 , 5 , 6],[7 , 8 , 9]]

2次元配列

```
x2 = np.array([[1,2,3],[4,5,6],[7,8,9]])
```

形状は？ `x2.shape` → (3,3) **3 × 3 の2次元配列**

[4,5,6]を取り出すには？

`x2[1]` → [4,5,6] **x2の2つ目の配列**

[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
x2[0] x2[1] x2[2]

1 番外側の[]だけに注目すると、コンマ(,)区切りで3つの要素が存在

**xが[1,2,3,4]のshapeは(4,)で1 番外側の[]には4つの要素が存在
→shapeの(n,) や(n,m)の1 番左の数字nは1 番外側の要素の数**

2次元配列

```
x2 = np.array([[1,2,3],[4,5,6],[7,8,9]])
```

形状は？ `x2.shape` → (3,3) **3 × 3 の2次元配列**

[4,5,6]を取り出すには？

`x2[1]` → [4,5,6] **x2の2つ目の配列**

[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
x2[0] x2[1] x2[2]

1番外側の[]だけに注目すると、コンマ(,)区切りで3つの要素が存在

**xが[1,2,3,4]のshapeは(4,)で1番外側の[]には4つの要素が存在
→shapeの(n,) や(n,m)の1番左の数字nは1番外側の要素の数**

2次元配列

```
x2 = np.array([[1,2,3],[4,5,6],[7,8,9]])
```

形状は？ `x2.shape` → (3,3) **3 × 3 の2次元配列**

[4,5,6]の5を取り出すには？

`[[1 , 2 , 3],[4 , 5 , 6],[7 , 8 , 9]]`

2次元配列

```
x2 = np.array([[1,2,3],[4,5,6],[7,8,9]])
```

形状は？ `x2.shape` → (3,3) **3 × 3 の2次元配列**

[4,5,6]の5を取り出すには？

`x2[1][1]` → 5 **`x2[1]`の2つ目**

`[[1 , 2 , 3],[4 , 5 , 6],[7 , 8 , 9]]`

2次元配列

```
x2 = np.array([[1,2,3],[4,5,6],[7,8,9]])
```

形状は？ `x2.shape` → (3,3) **3 × 3 の2次元配列**

[4,5,6]の5を取り出すには？

`x2[1][1]` → 5 **`x2[1]`の2つ目**

`[[1 , 2 , 3], [4 , 5 , 6], [7 , 8 , 9]]`

`x2[1]`

`x2[2]`の中身である[7,8,9]はコンマ区切りで要素が3つ

2次元配列

```
x2 = np.array([[1,2,3],[4,5,6],[7,8,9]])
```

形状は？ `x2.shape` → (3,3) **3 × 3 の2次元配列**

[4,5,6]の5を取り出すには？

`x2[1][1]` → 5 **`x2[1]`の2つ目**

`[[1 , 2 , 3], [4 , 5 , 6], [7 , 8 , 9]]`

`x2[1]`

`x2[1][0]` `x2[1][1]` `x2[1][2]`

`x2[1]`の中身である[4,5,6]はコンマ区切りで要素が3つ

```
x2 = np.array([[1,2,3],[4,5,6],[7,8,9]])  
print(x2.shape) → (3,3)
```

3 × 3 の2次元配列

```
print(x2)
```

```
[[ 1  2  3]  
 [ 4  5  6]  
 [ 7  8  9]]
```


[]の中に[]があるので2次元配列

```
x2 = np.array([[1,2,3],[4,5,6],[7,8,9]])  
print(x2.shape) → (3,3)
```


3 × 3 の2次元配列

```
print(x2)
```

3列



3行



```
[[ 1  2  3]  
 [ 4  5  6]  
 [ 7  8  9]]
```

[]の中に[]があるので2次元配列

形状が(2,3,2)の場合は？

```
x3 = np.arange(12)
print(x3.shape)
x3
```

```
(12,)
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11])
```

```
x3 = x3.reshape(2,3,2)
print(x3.shape)
x3
```

```
(2, 3, 2)
array([[[ 0,  1],
        [ 2,  3],
        [ 4,  5]],

       [[ 6,  7],
        [ 8,  9],
        [10, 11]]])
```

np.arange(num)

0からnum未満の1次元配列を作成(連番)

np配列.reshape(指定する形状)

np配列を指定する形状に変換


```
print(x3.shape)  
x3
```

```
(2, 3, 2)  
array([[[ 0,  1],  
        [ 2,  3],  
        [ 4,  5]],  
       [[ 6,  7],  
        [ 8,  9],  
        [10, 11]]])
```

()の1つ目が2なので一番外側の[]の要素は2つ

[ , ]

```
print(x3.shape)  
x3
```

(2, 3, 2)

```
array([[ 0,  1],  
       [ 2,  3],  
       [ 4,  5]],  
      [[ 6,  7],  
       [ 8,  9],  
       [10, 11]])
```

()の1つ目が2なので一番外側の[]の要素は2つ

[, ]

```
x3[0]
```

最初の[]の要素の1つ目
3×2の配列の1つ目

```
array([ 0,  1],  
      [ 2,  3],  
      [ 4,  5])
```

```
x3[1]
```

最初の[]の要素の2つ目
3×2の配列の2つ目

```
array([ 6,  7],  
      [ 8,  9],  
      [10, 11])
```

```
print(x3.shape)  
x3
```

```
(2, 3, 2)  
array([[ 0,  1],  
       [ 2,  3],  
       [ 4,  5]],  
      [[ 6,  7],  
       [ 8,  9],  
       [10, 11]])
```

()の2つ目が3なので2つ目の[]の要素は3つ

[[, , ], [, , ]]

```
x3[0][0]
```

```
array([0, 1])
```

最初の[]の要素の1つ目
2つ目の[]の要素の1つ目

```
x3[1][2]
```

```
array([10, 11])
```

最初の[]の要素の2つ目
2つ目の[]の要素の3つ目

```
print(x3.shape)  
x3
```

```
(2, 3, 2)  
array([[ [ 0,  1],  
        [ 2,  3],  
        [ 4,  5]],  
       [[ [ 6,  7],  
        [ 8,  9],  
        [10, 11]]])
```

()の3つ目が2なので3つ目の[]の要素は2つ

[[[■, ■], [■, ■], [■, ■]], [[■, ■], [■, ■], [■, ■]]]

```
x3[0][0][0]
```

0

最初の[]の要素の1つ目
2つ目の[]の要素の1つ目
3つ目の[]の要素の1つ目

```
x3[1][2][1]
```

11

最初の[]の要素の2つ目
2つ目の[]の要素の3つ目
3つ目の[]の要素の2つ目

```
print(x_train.shape)  
60000, 28, 28
```

```
print(x_train[0])は？
```

1つ目を取り出してみる

```
print(x_train[0])
```

```
print(x_train[0].shape)
(28, 28)
```

```
print(x_train[0])
```

```
[
  [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
    0  0  0  0  0  0  0  0  0  0]
  [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
    0  0  0  0  0  0  0  0  0  0]
  [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
    0  0  0  0  0  0  0  0  0  0]
  [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
    0  0  0  0  0  0  0  0  0  0]
  [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
    0  0  0  0  0  0  0  0  0  0]
  [ 0  0  0  0  0  0  0  0  0  0  0  0  0  3  18  18  18 126 136
    175 26 166 255 247 127 0 0 0 0]
  [ 0  0  0  0  0  0  0  0  30 36 94 154 170 253 253 253 253 253
    225 172 253 242 195 64 0 0 0 0]
  [ 0  0  0  0  0  0  0  49 238 253 253 253 253 253 253 253 253 251
    93 82 82 56 39 0 0 0 0 0]
  [ 0  0  0  0  0  0  0  18 219 253 253 253 253 253 198 182 247 241
    0  0  0  0  0  0  0  0  0 0]
  [ 0  0  0  0  0  0  0  80 156 107 253 253 205 11 0 43 154
    0  0  0  0  0  0  0  0  0 0]
  [ 0  0  0  0  0  0  0  0 14 1 154 253 90 0 0 0 0
    0  0  0  0  0  0  0  0  0 0]
  [ 0  0  0  0  0  0  0  0 0 0 139 253 190 2 0 0 0
    0  0  0  0  0  0  0  0  0 0]
  [ 0  0  0  0  0  0  0  0 0 0 0 11 190 253 70 0 0
    0  0  0  0  0  0  0  0  0 0]
  [ 0  0  0  0  0  0  0  0 0 0 0 0 35 241 225 160 108 1
    0  0  0  0  0  0  0  0  0 0]
  [ 0  0  0  0  0  0  0  0 0 0 0 0 0 81 240 253 253 119
    25 0 0 0 0 0 0 0 0 0 0 0 0 0]
  [ 0  0  0  0  0  0  0  0 0 0 0 0 0 0 0 45 186 253 253
    150 27 0 0 0 0 0 0 0 0 0 0 0]
  [ 0  0  0  0  0  0  0  0 0 0 0 0 0 0 0 0 16 93 252
    253 187 0 0 0 0 0 0 0 0 0 0]
  [ 0  0  0  0  0  0  0  0 0 0 0 0 0 0 0 0 0 0 249
    253 249 64 0 0 0 0 0 0 0 0]
  [ 0  0  0  0  0  0  0  0 0 0 0 0 0 0 46 130 183 253
    253 207 2 0 0 0 0 0 0 0 0]
  [ 0  0  0  0  0  0  0  0 0 0 0 0 0 39 148 229 253 253 253
    250 182 0 0 0 0 0 0 0 0 0]
  [ 0  0  0  0  0  0  0  0 0 0 24 114 221 253 253 253 253 201
    78 0 0 0 0 0 0 0 0 0 0 0]
  [ 0  0  0  0  0  0  0  23 66 213 253 253 253 253 198 81 2
    0  0  0  0  0  0  0  0 0]
  [ 0  0  0  0  0  18 171 219 253 253 253 195 80 9 0 0
    0  0  0  0  0  0  0  0 0]
  [ 0  0  0  55 172 226 253 253 253 253 244 133 11 0 0 0
    0  0  0  0  0  0  0  0 0]
  [ 0  0  0  0 136 253 253 253 212 135 132 16 0 0 0 0
    0  0  0  0  0  0  0  0 0]
  [ 0  0  0  0  0  0  0  0 0 0 0 0 0 0 0 0 0
    0  0  0  0  0  0  0  0 0]
  [ 0  0  0  0  0  0  0  0 0 0 0 0 0 0 0 0 0
    0  0  0  0  0  0  0  0 0]
  [ 0  0  0  0  0  0  0  0 0 0 0 0 0 0 0 0 0
    0  0  0  0  0  0  0  0 0]
```


テキストエディタで開いたところ

1	[0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0]	↓		
2	[0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0]	↓		
3	[0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0]	↓		
4	[0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0]	↓		
5	[0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0]	↓		
6	[0	0	0	0	0	0	0	0	0	0	0	0	3	18	18	18	126	136	175	26	166	255	247	127	0	0	0	0]	↓
7	[0	0	0	0	0	0	0	0	30	36	94	154	170	253	253	253	253	253	225	172	253	242	195	64	0	0	0	0]	↓
8	[0	0	0	0	0	0	0	49	238	253	253	253	253	253	253	253	253	251	93	82	82	56	39	0	0	0	0]	↓	
9	[0	0	0	0	0	0	0	18	219	253	253	253	253	253	198	182	247	241	0	0	0	0	0	0	0	0	0]	↓	
10	[0	0	0	0	0	0	0	0	80	156	107	253	253	205	11	0	43	154	0	0	0	0	0	0	0	0	0]	↓	
11	[0	0	0	0	0	0	0	0	0	14	1	154	253	90	0	0	0	0	0	0	0	0	0	0	0	0	0]	↓	
12	[0	0	0	0	0	0	0	0	0	0	0	139	253	190	2	0	0	0	0	0	0	0	0	0	0	0	0]	↓	
13	[0	0	0	0	0	0	0	0	0	0	0	11	190	253	70	0	0	0	0	0	0	0	0	0	0	0	0]	↓	
14	[0	0	0	0	0	0	0	0	0	0	0	0	35	241	225	160	108	1	0	0	0	0	0	0	0	0	0]	↓	
15	[0	0	0	0	0	0	0	0	0	0	0	0	0	81	240	253	253	119	25	0	0	0	0	0	0	0	0]	↓	
16	[0	0	0	0	0	0	0	0	0	0	0	0	0	0	45	186	253	253	150	27	0	0	0	0	0	0	0]	↓	
17	[0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	16	93	252	253	187	0	0	0	0	0	0	0]	↓	
18	[0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	249	253	249	64	0	0	0	0	0	0]	↓	
19	[0	0	0	0	0	0	0	0	0	0	0	0	0	0	46	130	183	253	253	207	2	0	0	0	0	0	0]	↓	
20	[0	0	0	0	0	0	0	0	0	0	0	0	39	148	229	253	253	253	250	182	0	0	0	0	0	0	0]	↓	
21	[0	0	0	0	0	0	0	0	0	0	24	114	221	253	253	253	253	201	78	0	0	0	0	0	0	0	0]	↓	
22	[0	0	0	0	0	0	0	0	23	66	213	253	253	253	253	198	81	2	0	0	0	0	0	0	0	0	0]	↓	
23	[0	0	0	0	0	0	18	171	219	253	253	253	253	195	80	9	0	0	0	0	0	0	0	0	0	0	0]	↓	
24	[0	0	0	0	55	172	226	253	253	253	253	244	133	11	0	0	0	0	0	0	0	0	0	0	0	0	0]	↓	
25	[0	0	0	0	136	253	253	253	212	135	132	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0]	↓	
26	[0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0]	↓	
27	[0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0]	↓	
28	[0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0]]	↓	

テキストエディタで開いたところ

[illegible]



```
print(x_train[0][7])
```

```
[ 0  0  0  0  0  0  0 49 238 253 253 253 253 253 253 253 251
 93 82 82 56 39  0  0  0  0]
```

1	[0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0]	↓	
2	[0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0]	↓	
3	[0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0]	↓	
4	[0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0]	↓	
5	[0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0]	↓	
6	[0	0	0	0	0	0	0	0	0	0	0	0	3	18	18	18	126	136	175	26	166	255	247	127	0	0	0]	↓
7	[0	0	0	0	0	0	0	0	30	36	94	154	170	253	253	253	253	253	225	172	253	242	195	64	0	0	0]	↓
8	[0	0	0	0	0	0	0	49	238	253	253	253	253	253	253	253	253	251	93	82	82	56	39	0	0	0]	↓	
9	[0	0	0	0	0	0	0	18	219	253	253	253	253	253	198	182	247	241	0	0	0	0	0	0	0	0]	↓	
10	[0	0	0	0	0	0	0	0	80	156	107	253	253	205	11	0	43	154	0	0	0	0	0	0	0	0]	↓	
11	[0	0	0	0	0	0	0	0	0	14	1	154	253	90	0	0	0	0	0	0	0	0	0	0	0	0]	↓	
12	[0	0	0	0	0	0	0	0	0	0	0	139	253	190	2	0	0	0	0	0	0	0	0	0	0	0]	↓	
13	[0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0]	↓	
14	[0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0]	↓	
15	[0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0]	↓	
16	[0	0	0	0	0	0	0	0	0	0	0	0	0	0	45	186	253	253	150	27	0	0	0	0	0	0]	↓	
17	[0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	16	93	252	253	187	0	0	0	0	0	0]	↓	
18	[0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	249	253	249	64	0	0	0	0	0]	↓	
19	[0	0	0	0	0	0	0	0	0	0	0	0	0	0	46	130	183	253	253	207	2	0	0	0	0	0]	↓	
20	[0	0	0	0	0	0	0	0	0	0	0	0	39	148	229	253	253	253	250	182	0	0	0	0	0	0]	↓	
21	[0	0	0	0	0	0	0	0	0	0	24	114	221	253	253	253	253	201	78	0	0	0	0	0	0	0]	↓	
22	[0	0	0	0	0	0	0	0	23	66	213	253	253	253	253	198	81	2	0	0	0	0	0	0	0	0]	↓	
23	[0	0	0	0	0	0	18	171	219	253	253	253	253	195	80	9	0	0	0	0	0	0	0	0	0	0]	↓	
24	[0	0	0	0	55	172	226	253	253	253	253	244	133	11	0	0	0	0	0	0	0	0	0	0	0	0]	↓	
25	[0	0	0	0	136	253	253	253	212	135	132	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0]	↓	
26	[0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0]	↓	
27	[0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0]	↓	
28	[0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0]	↓	

0始まりなので[0][7]は1枚目の8行目

0始まりなので[0][7]は1枚目の8行目



```
print(x_train[0][7][7])
```

49

1	[0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0]	↓			
2	[0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0]	↓			
3	[0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0]	↓			
4	[0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0]	↓			
5	[0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0]	↓			
6	[0	0	0	0	0	0	0	0	0	0	0	0	3	18	18	18	126	136	175	26	166	255	247	127	0	0	0]	↓		
7	[0	0	0	0	0	0	0	0	30	36	94	154	170	253	253	253	253	253	225	172	253	242	195	64	0	0	0]	↓		
8	[0	0	0	0	0	0	0	49	238	253	253	253	253	253	253	253	253	251	93	82	82	56	39	0	0	0]	↓			
9	[0	0	0	0	0	0	0	18	219	253	253	253	253	253	198	182	247	241	0	0	0	0	0	0	0	0]	↓			
10	[0	0	0	0	0	0	0	0	80	156	107	253	253	205	11	0	43	154	0	0	0	0	0	0	0	0]	↓			
11	[0	0	0	0	0	0	0	0	0	14	1	154	253	90	0	0	0	0	0	0	0	0	0	0	0	0]	↓			
12	[0	0	0	0	0	0	0	0	0	0	0	139	253	190	2	0	0	0	0	0	0	0	0	0	0	0]	↓			
13	0始まりなので[0][7][7]は1枚目の8行目の8列目																												0	0]	↓
14	[0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0]	↓			
15	[0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0]	↓			
16	[0	0	0	0	0	0	0	0	0	0	0	0	0	0	45	186	253	253	150	27	0	0	0	0	0	0]	↓			
17	[0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	16	93	252	253	187	0	0	0	0	0	0]	↓			
18	[0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	249	253	249	64	0	0	0	0	0]	↓			
19	[0	0	0	0	0	0	0	0	0	0	0	0	0	0	46	130	183	253	253	207	2	0	0	0	0	0]	↓			
20	[0	0	0	0	0	0	0	0	0	0	0	0	39	148	229	253	253	253	250	182	0	0	0	0	0	0]	↓			
21	[0	0	0	0	0	0	0	0	0	0	24	114	221	253	253	253	253	201	78	0	0	0	0	0	0	0]	↓			
22	[0	0	0	0	0	0	0	0	23	66	213	253	253	253	253	198	81	2	0	0	0	0	0	0	0	0]	↓			
23	[0	0	0	0	0	0	18	171	219	253	253	253	253	195	80	9	0	0	0	0	0	0	0	0	0	0]	↓			
24	[0	0	0	0	55	172	226	253	253	253	253	244	133	11	0	0	0	0	0	0	0	0	0	0	0	0]	↓			
25	[0	0	0	0	136	253	253	253	212	135	132	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0]	↓			
26	[0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0]	↓			
27	[0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0]	↓			
28	[0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0]	↓			

0始まりなので[0][7][7]は1枚目の8行目の8列目


```
[6] print(y_train)
```

```
[5 0 4 ... 5 6 8]
```

```
▶ print(y_train[0])
```

```
5
```

```
[6] print(y_train)
```

```
[5 0 4 ... 5 6 8]
```

```
▶ print(y_train[0])
```

```
5
```

```
[6] print(y_train)
```

```
[5 0 4 ... 5 6 8]
```

```
▶ print(y_train[0])
```

```
5
```

```
[6] print(y_train)
```

```
[5 0 4 ... 5 6 8]
```

```
▶ print(y_train[0])
```

```
5
```

y_trainには60000枚の数字(=正解)
y_train[0]が1枚目の数字(=5)

[illegible]


```
[6] print(y_train)
```

```
[5 0 4 ... 5 6 8]
```

```
print(y_train[0])
```

```
5
```

y_trainには60000枚の数字(=正解)
y_train[0]が1枚目の数字(=5)

x_train[0]は1枚目の画像の配列(28*28)、
y_train[0]には1枚目の正解の数字

x_train[i]はi+1枚目の画像の配列(28*28)、
y_train[i]にはi+1枚目の正解の数字

課題

- WebClassにある課題1をやしましょう

締め切りは1週間後の5/2の23:59です。
締め切りを過ぎた課題は受け取らないので注意して下さい。
(1週間後に正解をアップします)