プログラミング言語

Python 入門

2020-02-20 – 2020-02-21

京都府農林水産技術センター

農研機構・農業情報研究センター
孫 建強

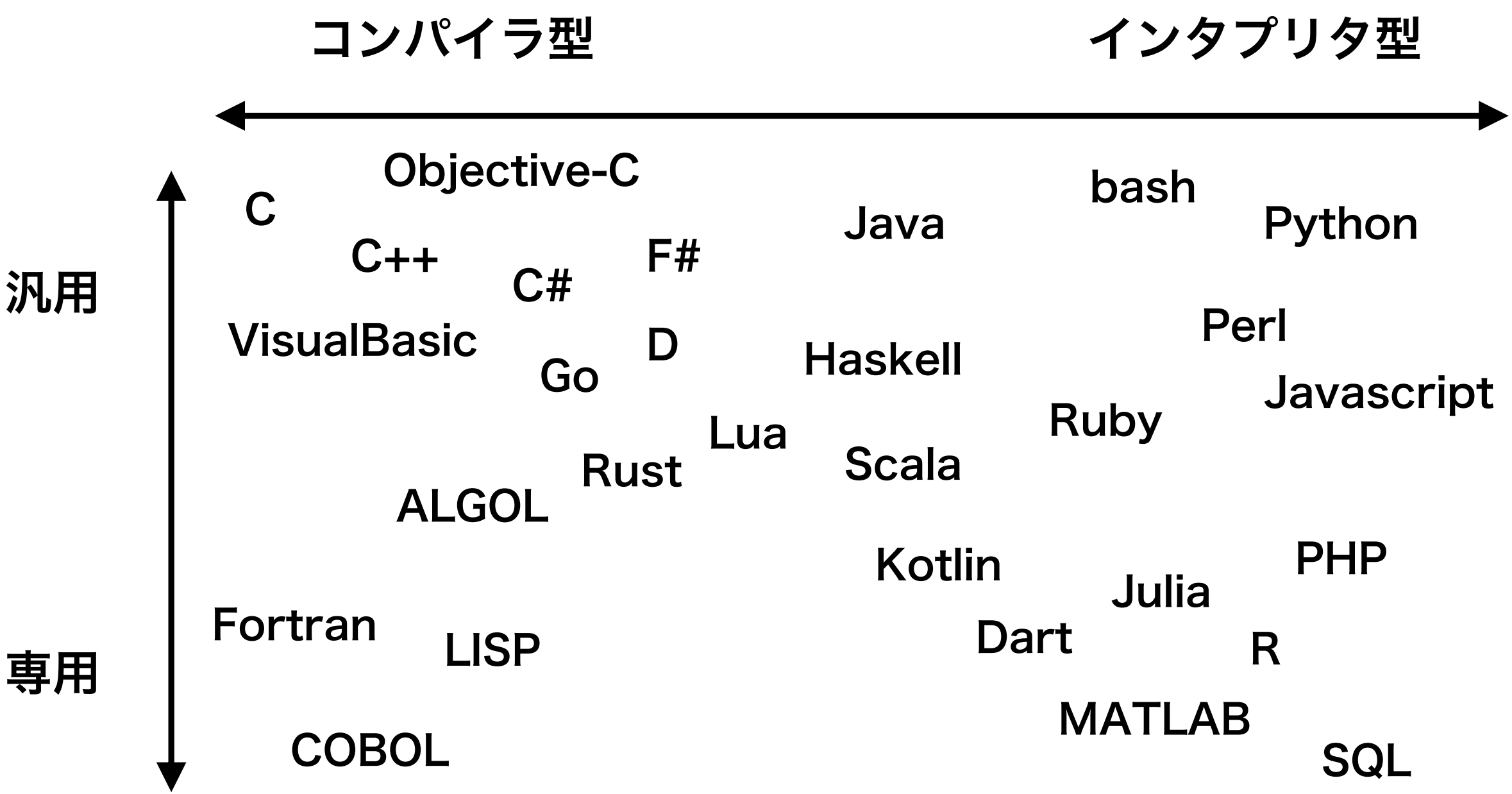
- Python 基礎
- NumPy
- Pandas
- matplotlib
- 機械学習概略

Python 基礎

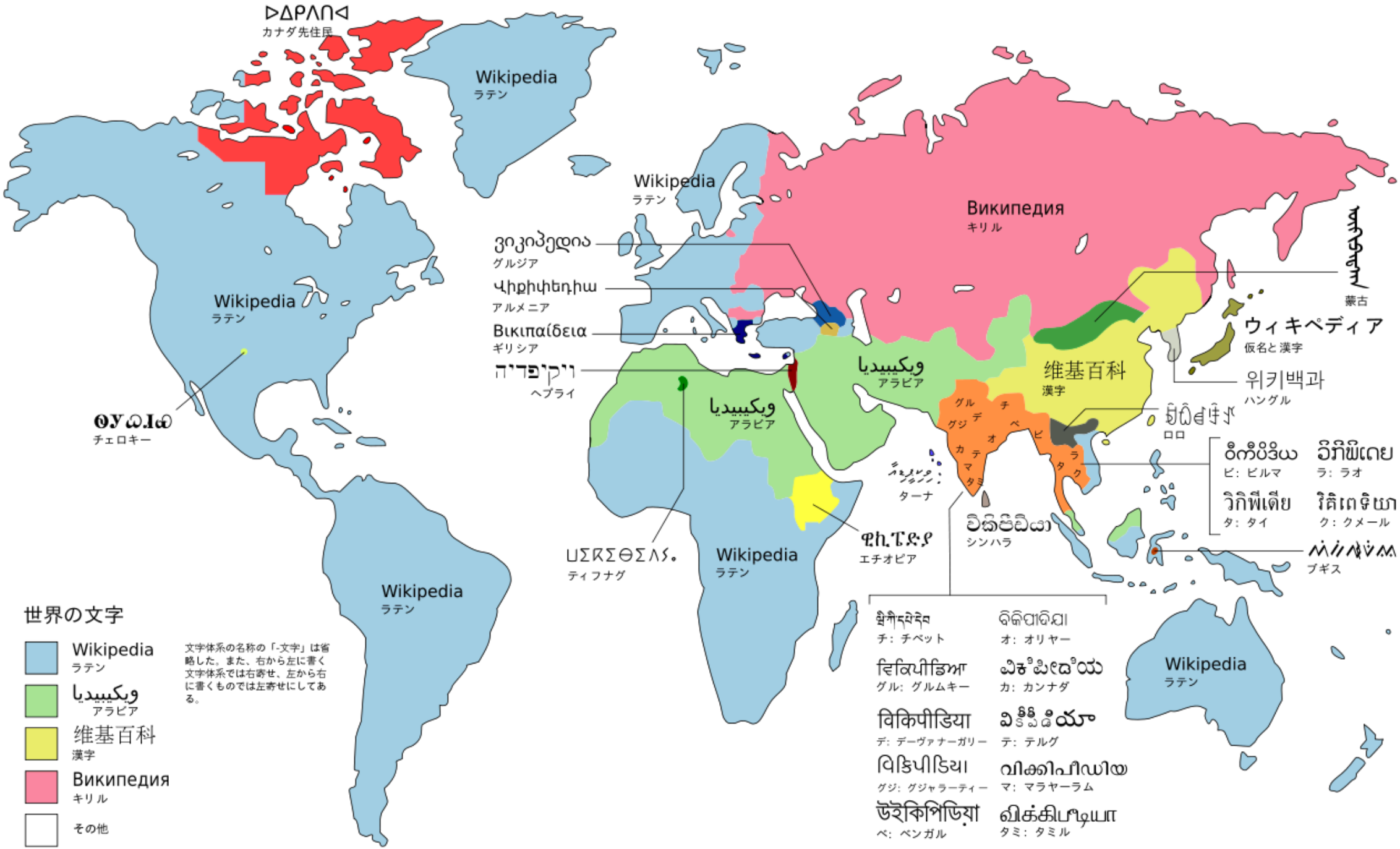
- プログラミング言語
- 基本オブジェクト
- 基本文法

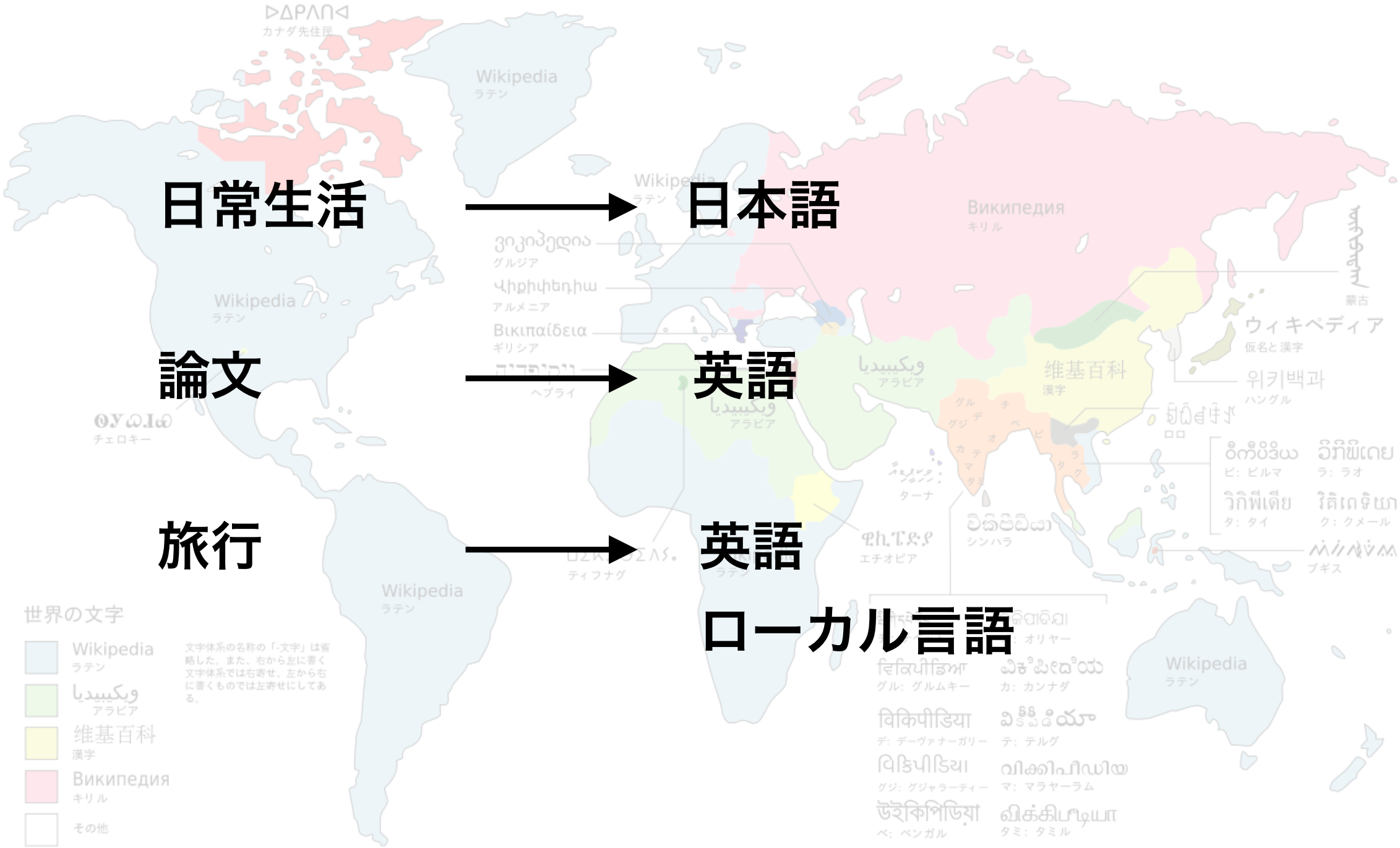
Python 基礎

- プログラミング言語
- 基本オブジェクト
- 基本文法


コンパイラ型
インタプリタ型
汎用
専用
C
Objective-C
C++
C#
F#
Java
bash
Python
VisualBasic
D
Go
Haskell
Perl
Javascript
Ruby
Lua
Rust
Scala
ALGOL
Kotlin
PHP
Julia
Fortran
LISP
Dart
R
MATLAB
COBOL
SQL


コンパイラ型
インタプリタ型
手続き型プログラミング言語
オブジェクト指向型プログラミング言語
関数型プログラミング言語
動的型付きプログラミング言語
理論型プログラミング言語
静的型付きプログラミング言語
汎用
専用
C
Objective-C
C++
C#
F#
Java
Python
VisualBasic
Haskell
Perl
Ruby
Scala
Javascript
ALGOL
Kotlin
Julia
Fortran
LISP
R
MATLAB
COBOL
SQL


ᐅᑕᕐᒪᐃᓐᓇᖅ
カナダ先住民
Wikipedia
ラテン
Wikipedia
ラテン
ᏫᎩᏪᎥᏫ
チェロキー
Wikipedia
ラテン
Wikipedia
ラテン
ვიკიპედია
グルジア
Վիքիփեդիա
アルメニア
Βικιπαίδεια
ギリシア
ויקיפדיה
ヘブライ
Википедия
キリル
ويكيبيديا
アラビア
ويكيبيديا
アラビア
维基百科
漢字
ᠤᠯᠠᠭᠠᠨᠪᠠᠭᠠᠲᠠᠷ
蒙古
ウィキペディア
仮名と漢字
위키백과
ハングル
ᥖᥭᥰᥛᥣᥴ
ロロ
ဝီကီပီဒီယ
ビ：ビルマ
ວິກິພີເດຍ
ラ：ラオ
วิกิพีเดีย
タ：タイ
វិគីពេទ្យ
ク：クメール
ᨅᨔᨗᨀᨔᨗᨊᨚ
ブギス
ⵜⴰⵎⴰⵣⵉⵖⵜ
ティフナグ
Wikipedia
ラテン
ᐋᑎᐧᑭᐱᑌᐤ
エチオピア
Wikipedia
ラテン
ཝི་ཀི་པི་ཌི་ཡ
チ：チベット
ವಿಕಿಪೀಡಿಯ
カ：カンナダ
വിക്കിപീഡിയ
マ：マラヤーラム
විකිපීඩියා
シンハラ
ವಿಕಿಪೀಡಿಯಾ
タミ：タミル
ᠸᠢᠺᠢᠫᠧᠳᠢᠶᠠ
モンゴル
Wikipedia
ラテン
世界の文字
Wikipedia
ラテン
ويكيبيديا
アラビア
维基百科
漢字
Википедия
キリル
その他
文字体系の名称の「-文字」は省略した。また、右から左に書く文字体系では右寄せ、左から右に書くものでは左寄せにしてある。


日常生活
日本語
論文
英語
旅行
英語
ローカル言語
世界の文字
Wikipedia
ラテン
Википедия
キリル
その他

アプリ開発 ⟶ C/C++, Java, Python, ...

ウェブ開発 ⟶ Python, Ruby, JavaScript, ...

科学計算 ⟶ Python, R, FORTRAN, ...

Guido van Rossum が C 言語のように機能が多くて、shell のように簡単に描けるプログラミング言語を開発した。プログラミング言語の名前はBBC コメディ番組『空飛ぶモンティ・パイソン』にちなんで Python と名付けられた。2004 年に ウェブ開発フレームワーク Django のリリースにより、Dropbox、YouTube、CIA などのウェブに Python が使われるようになる。また、2006 年以降、機械学習、人工知能などパッケージが開発され、Python ユーザー数がさらに増える。現在では 3.x が主流となり、2.x は 2020 年を持ってサポート終了。これから Python を始めるとき 3.7 以降を使うことをお薦めする。

1990 2000 2010

Python 0.9

Python 1.0

Python 2.0

Python 2.7

Python 3.0

Python 3.7

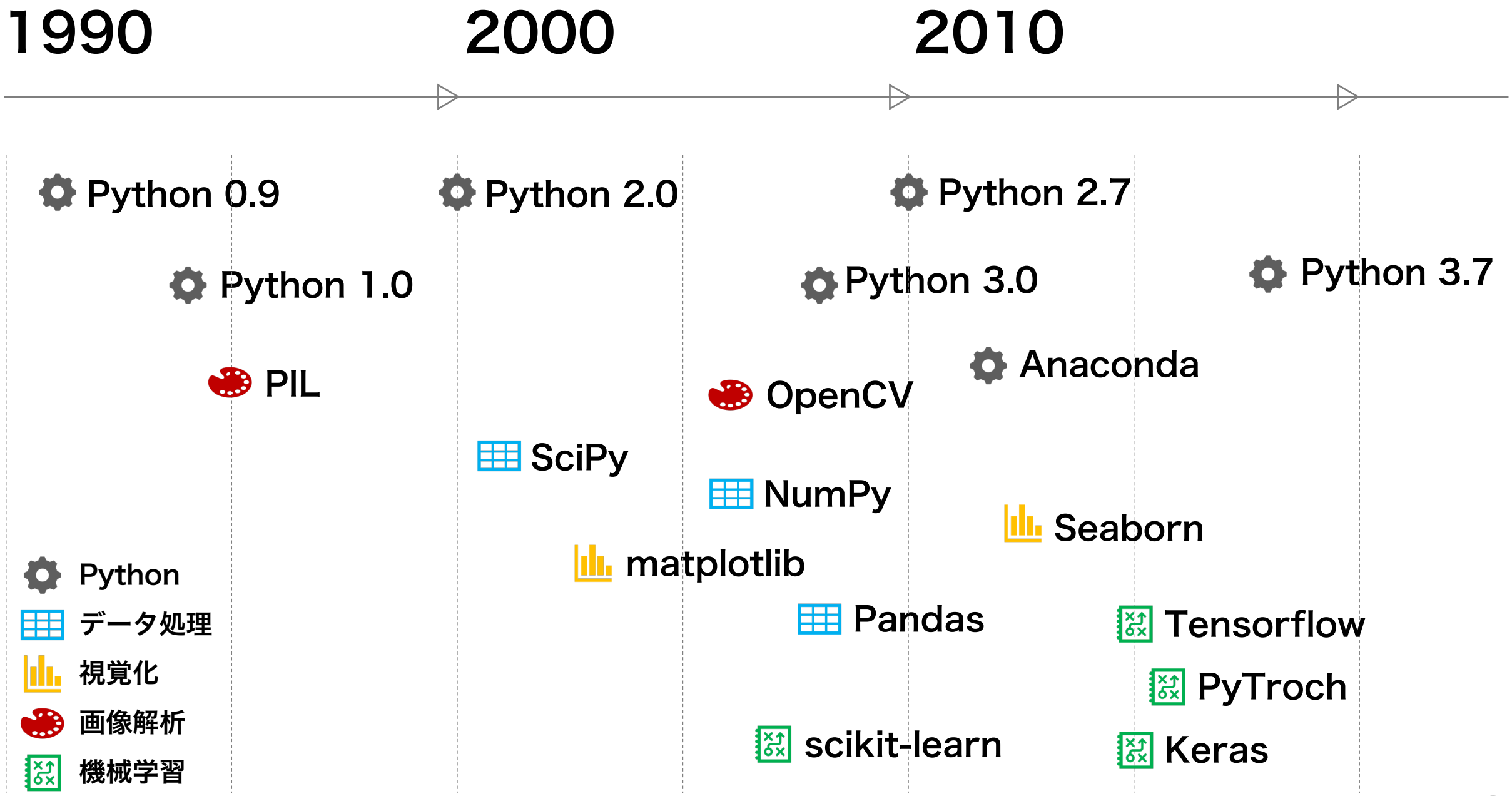
Guido van Rossum が C 言語のように機能が多くて、shell のように簡単に描けるプログラミング言語を開発した。プログラミング言語の名前はBBC コメディ番組『空飛ぶモンティ・パイソン』にちなんで Python と名付けられた。Python は性能重視に作られた言語ではないため、性能を高めたい場合は C 言語でライブラリー .so を作成して Python に読み込むことで対応する。

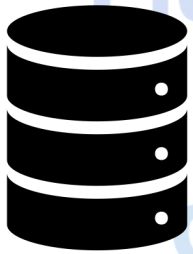
Python の基本的な特徴が決定される。
- オブジェクト指向型
- 動的型付け言語
- 他言語との互換性
- パッケージ

また、2004 年に ウェブ開発フレームワーク Django のリリースにより、Dropbox 、 YouTube 、 CIA などのウェブに Python が使われるようになる。

Python 2.x が盛んに使われる中、Python 3.0 がリリースされる。2008 年以降に、2.x と 3.0 系が並行して使われる時代に入る。機械学習、人工知能などパッケージが開発され、Python ユーザー数がさらに増える。現在では 3.x が主流となり、2.x は 2020 年を持ってサポート終了。これから Python を始めるとき 3.7 以降を使うことをお薦めする。

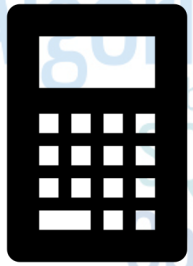
Python 4.0 は 2023 年にリリース予定？


1990
2000
2010
Python 0.9
Python 1.0
PIL
Python 2.0
SciPy
NumPy
matplotlib
OpenCV
Python 2.7
Python 3.0
Anaconda
Pandas
scikit-learn
Seaborn
Tensorflow
PyTroch
Keras
Python 3.7
Python
データ処理
視覚化
画像解析
機械学習



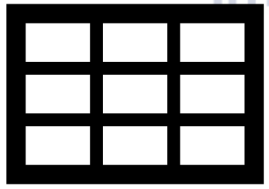
アプリ開発

統計処理



数値演算

画像解析

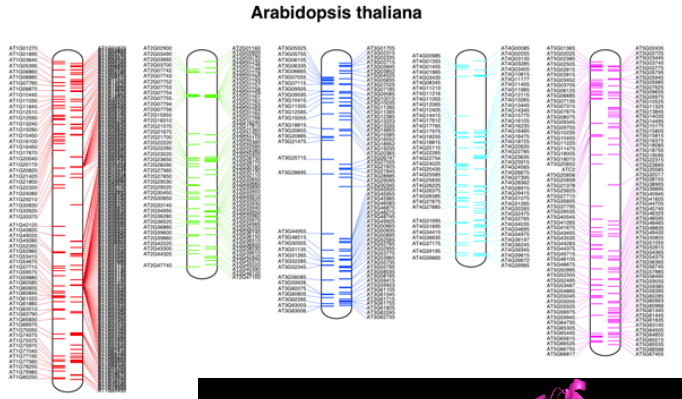


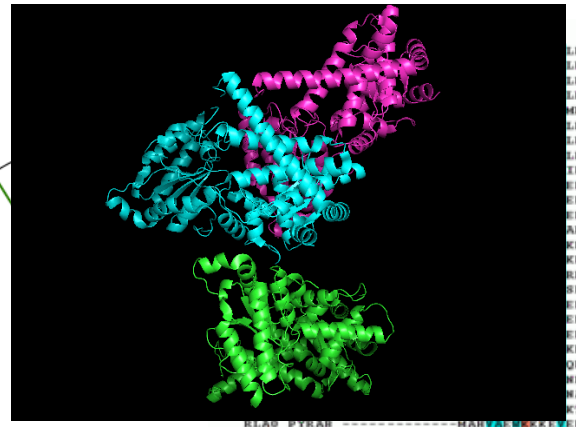
データ処理

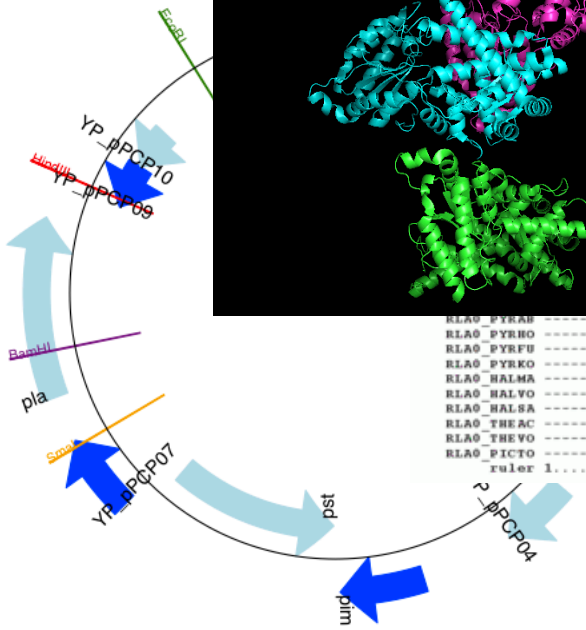
機械学習

塩基配列・アミノ酸配列解析

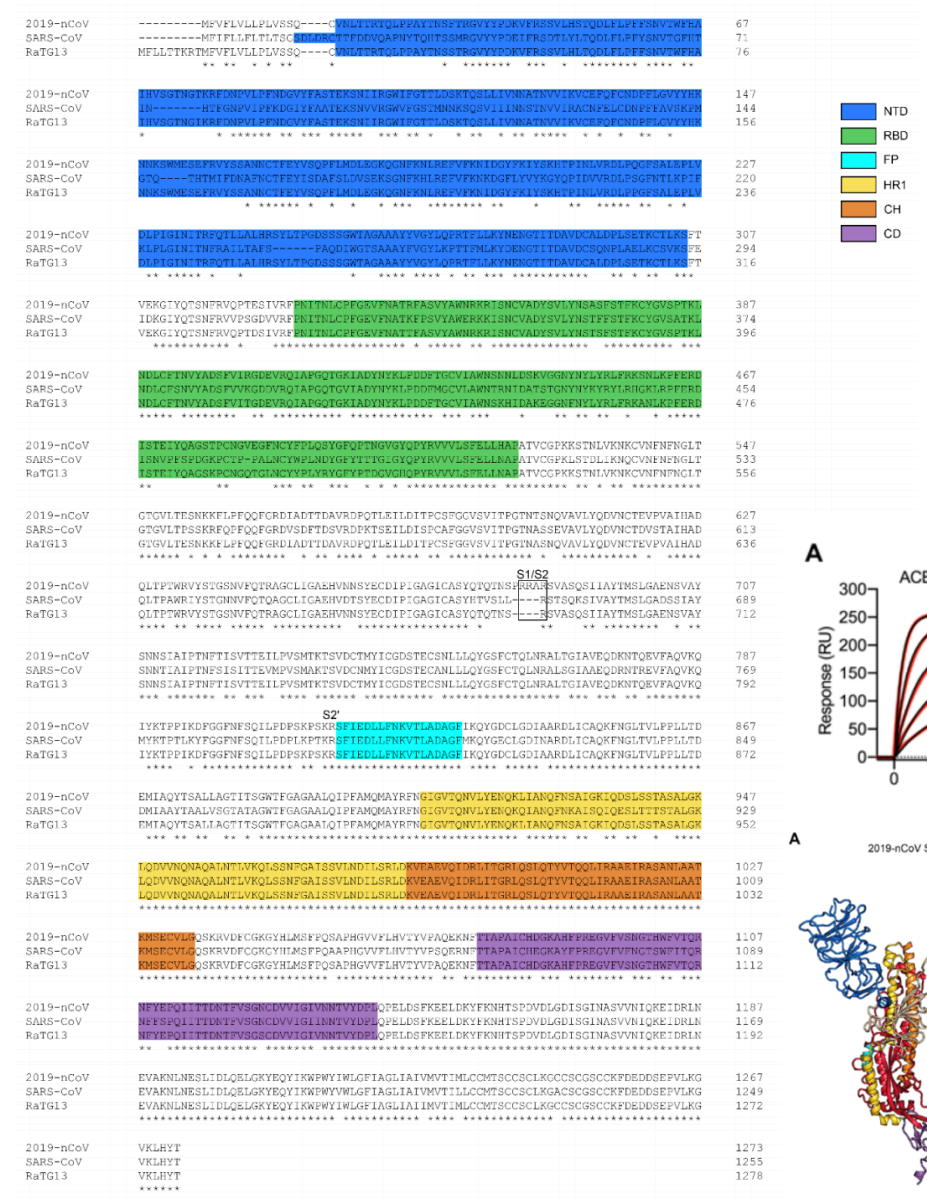
- biopython
  - DNA, RNA, アミノ酸配列処理
  - FASTA, PDB ファイルの処理
  - BLAST 結果パース
- scikit-bio
  - アラインメントアルゴリズム
- cutadapt / HTSeq
  - 高速シーケンサーデータ処理
- PyMOL
  - タンパク質立体構造視覚化

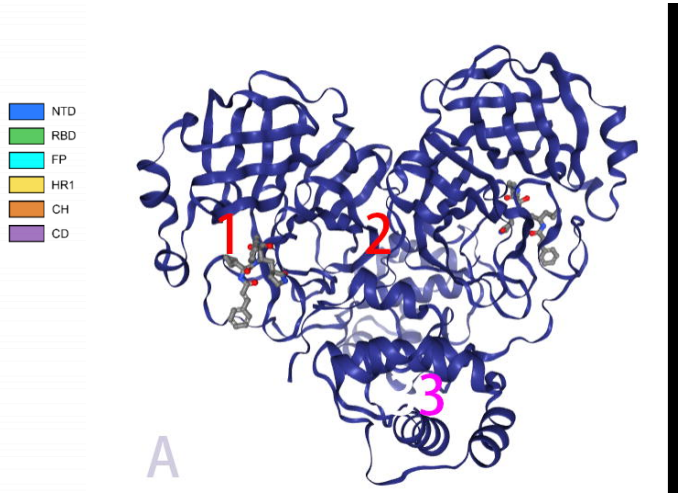


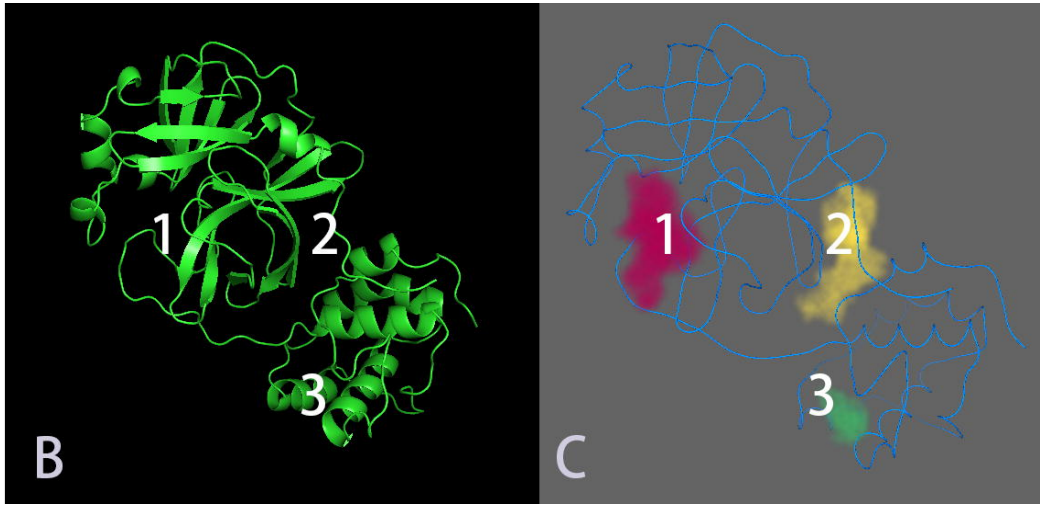


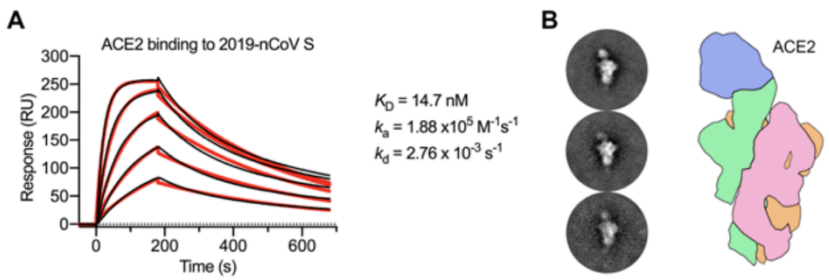


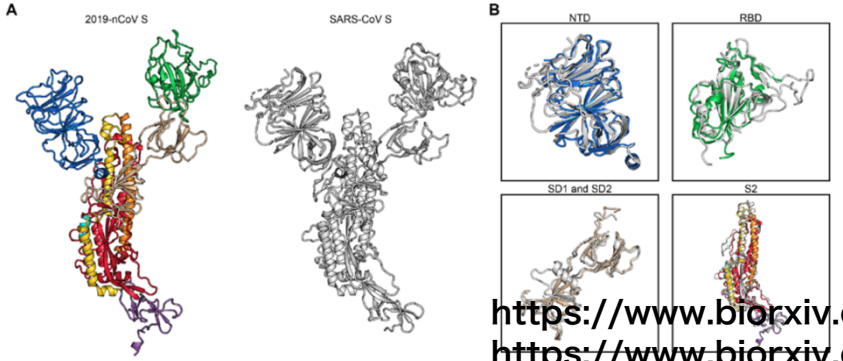
塩基配列・アミノ酸配列解析

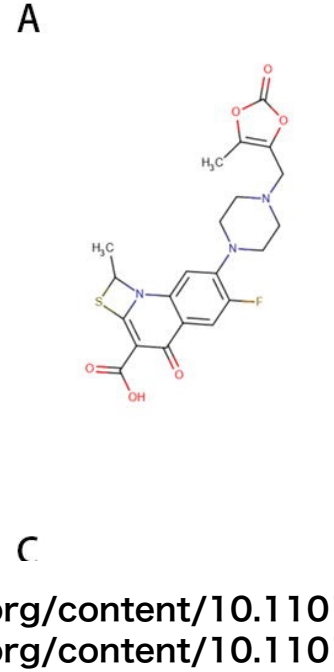










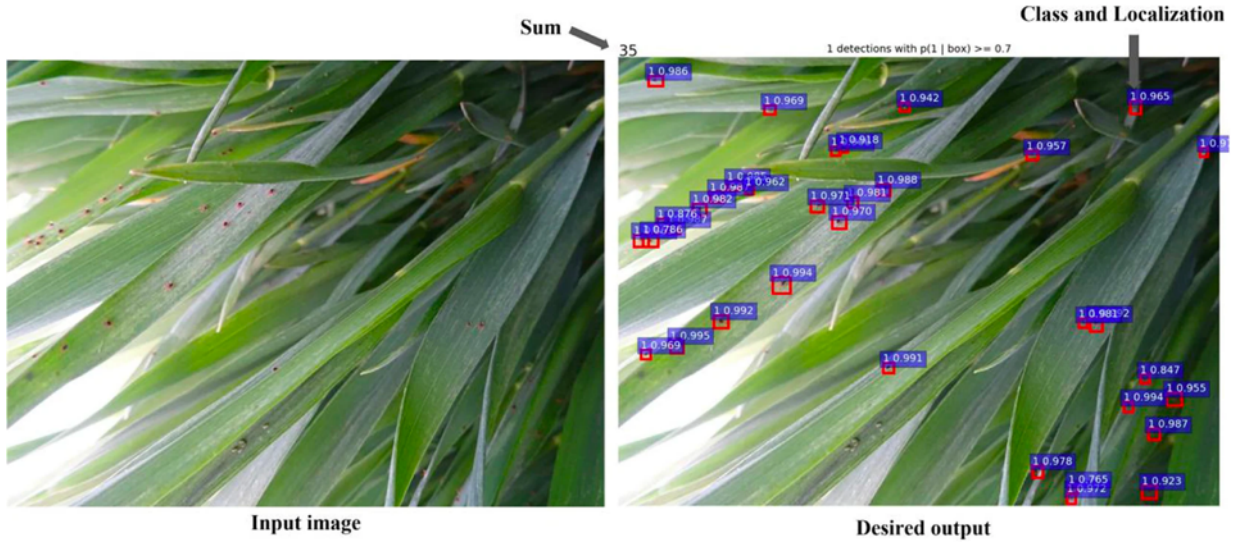


https://www.biorxiv.org/content/10.1101/2020.01.28.922922v2.full.pdf
https://www.biorxiv.org/content/10.1101/2020.02.11.944462v1.full.pdf

病害痕部位検出

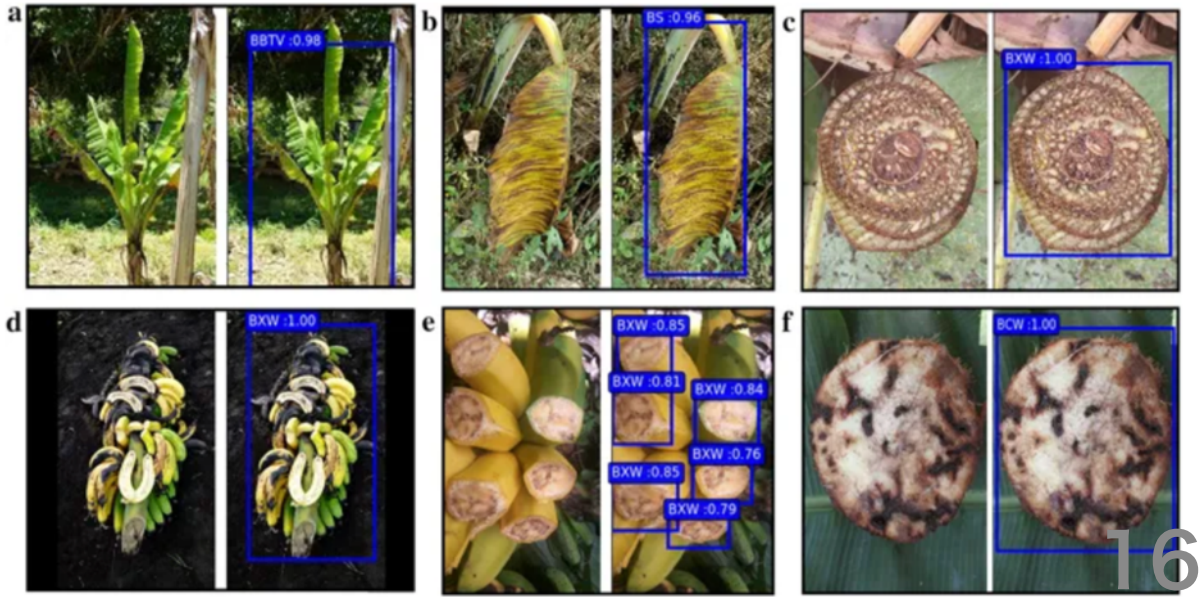
Li et al., Automatic Localization and Count of Agricultural Crop Pests Based on an Improved Deep Learning Pipeline.

https://doi.org/10.1038/s41598-019-43171-0



Selvaraj et al., AI-powered banana diseases and pest detection.

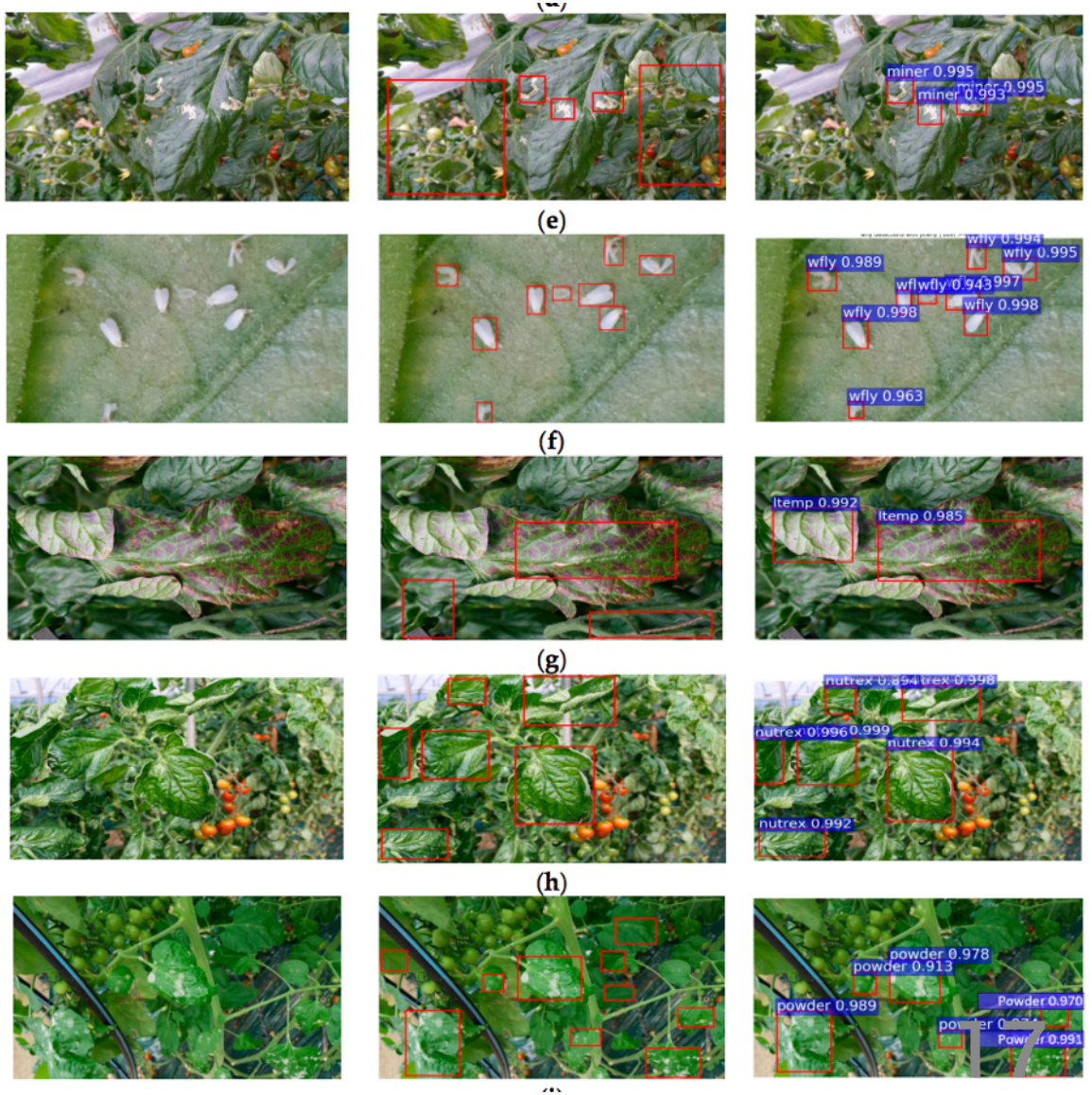
https://doi.org/10.1186/s13007-019-0475-z



病害痕部位検出

Fuentes et al., A Robust Deep-Learning-Based Detector for Real-Time Tomato Plant Diseases and Pests Recognition.

https://doi.org/10.3390/s17092022



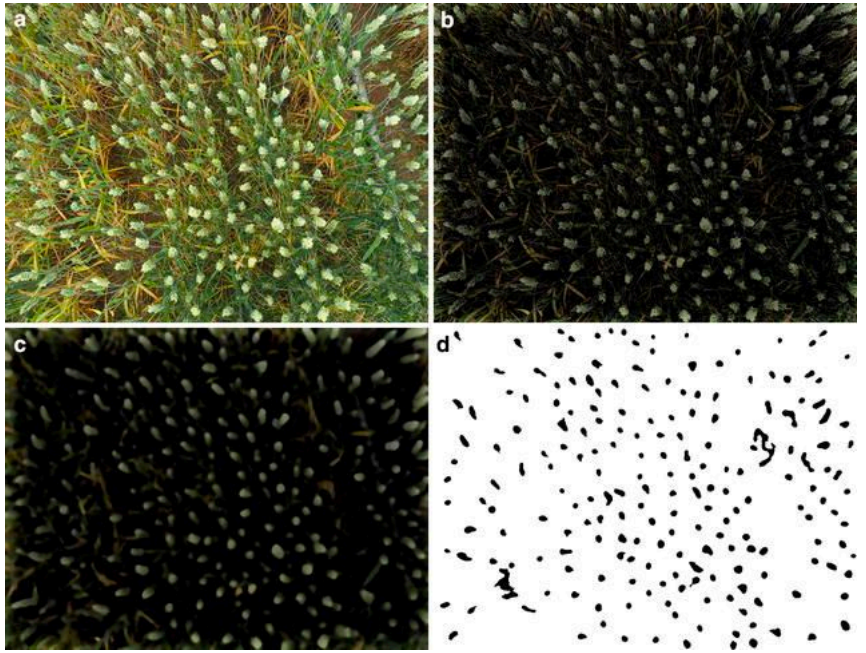
穂検出

Hasan et al., Detection and analysis of wheat spikes using Convolutional Neural Networks.

https://doi.org/10.1186/s13007-018-0366-8

Fernandez-Gallego et al., Wheat ear counting in-field conditions: high throughput and low-cost approach using RGB images.

https://doi.org/10.1186/s13007-018-0289-4



よくある質問

MATLAB と Python/R のどれを使った方がいいですか？

- シンプルな GUI
  - one-click で様々な処理が可能
  - サポートあり
- 高機能な関数が揃っている
- 学術機関で安く使える

- ほぼ CUI
  - ユーザー自身が各処理機能を書く
  - サポートなし
- ライブラリーに頼るか自分で関数を作る
- 無料

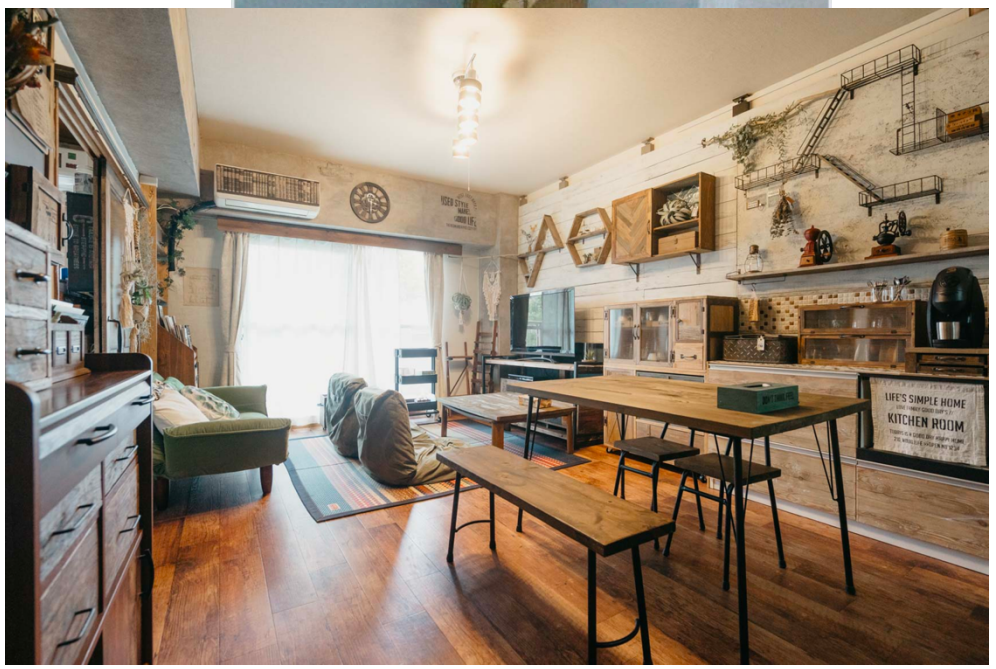
よくある質問

MATLAB と Python/R のどれを使った方がいいですか？

時間と財布に相談してください。

PRO

DIY



よくある質問

Python と R、どっちをよく使っていますか？

目的に応じて使い分けています。

- 統計検定
- 統計モデリング
- 高速シーケンサーデータ解析
- グラフ作成

- 左記以外

よくある質問

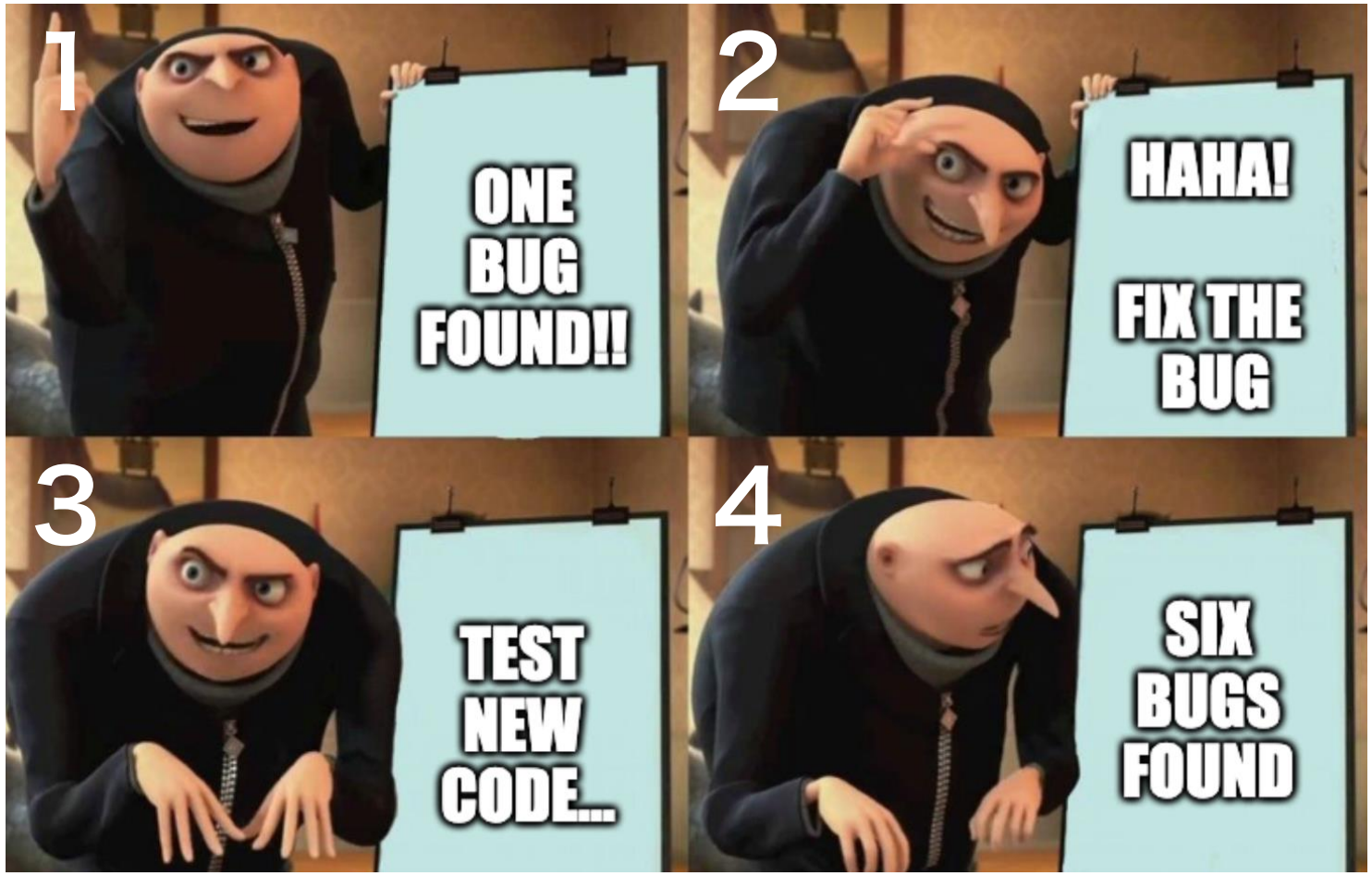
Python の関数をすべて覚えていますか？

覚えていません。常にインターネットで調べながら使っています。

よくある質問

慣れるとプログラムをすらすらと書けるようになりますか？

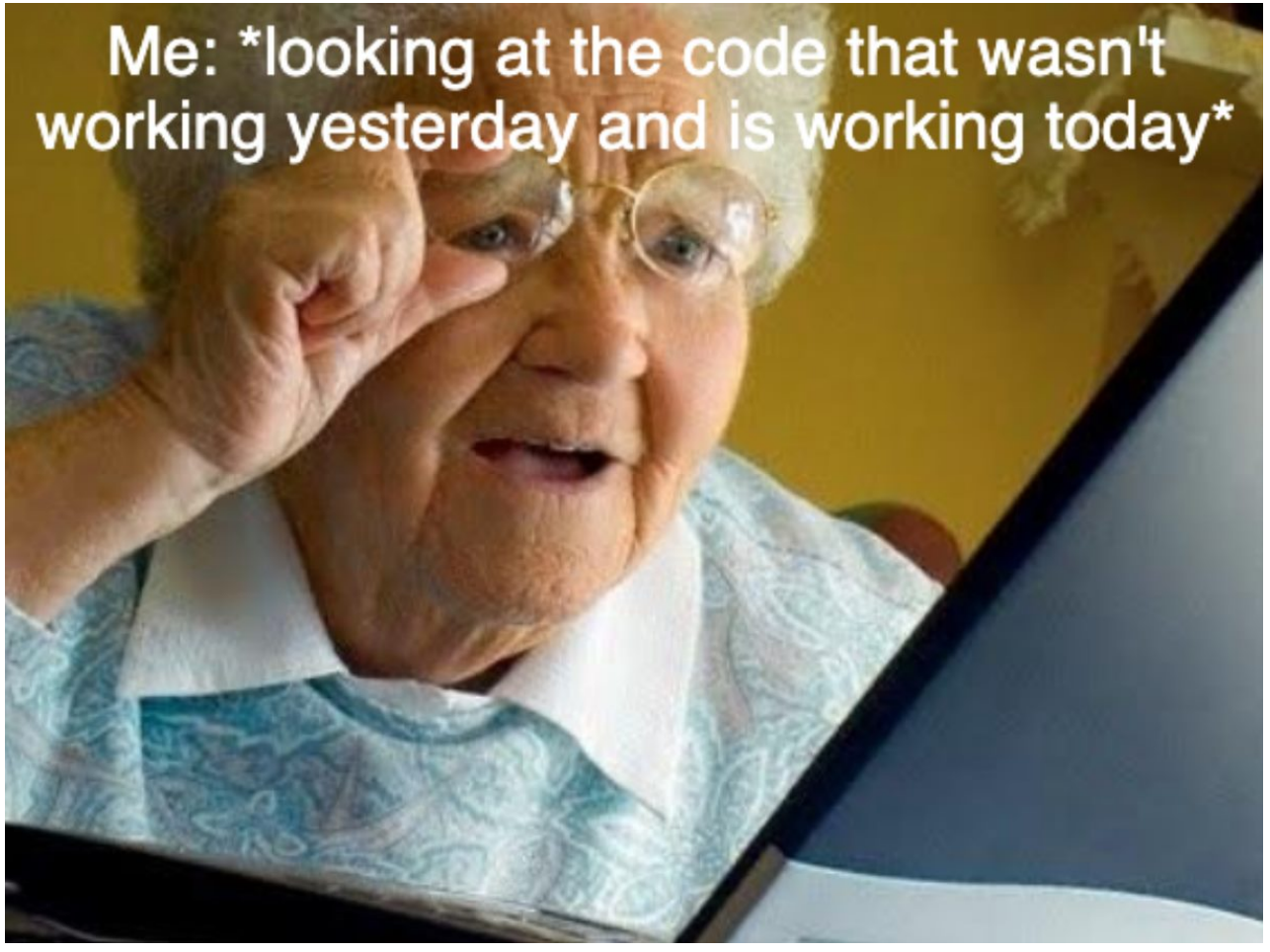
なりません。常にエラーとの戦いです。

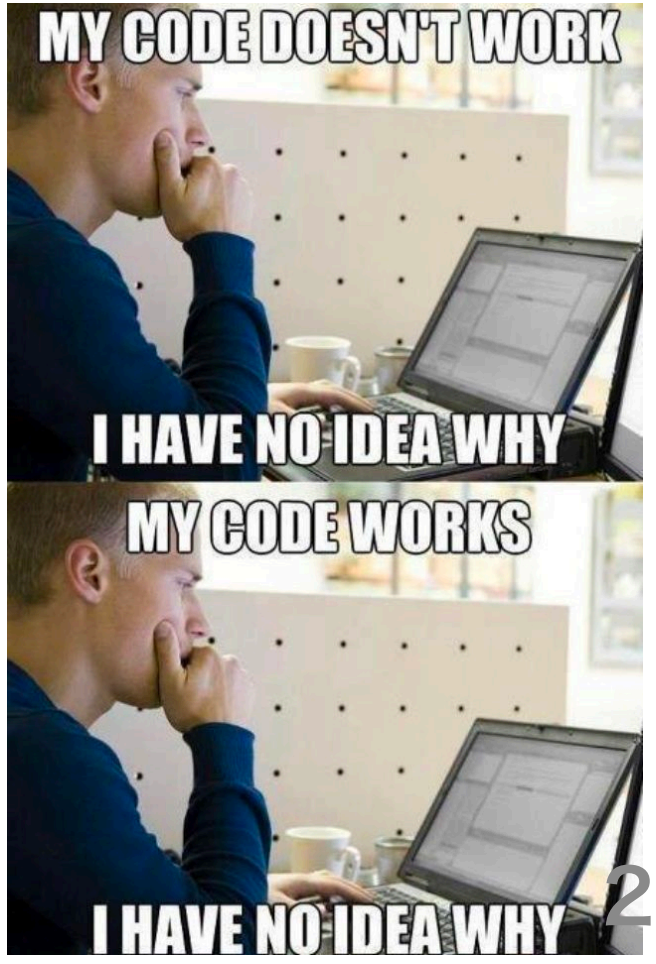


よくある質問

同じコードなのに動いたり動かなかったりしますか？

ありません。

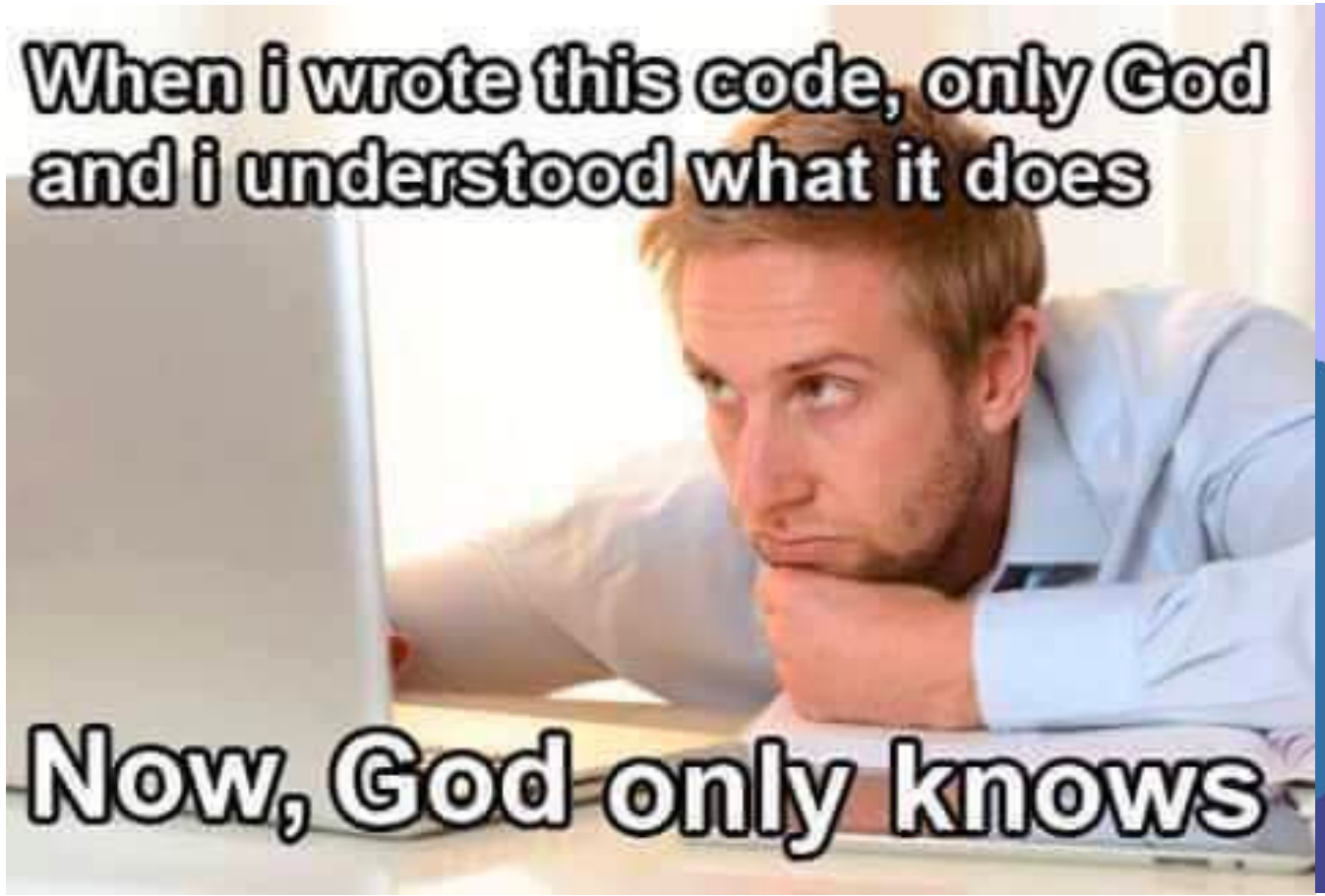


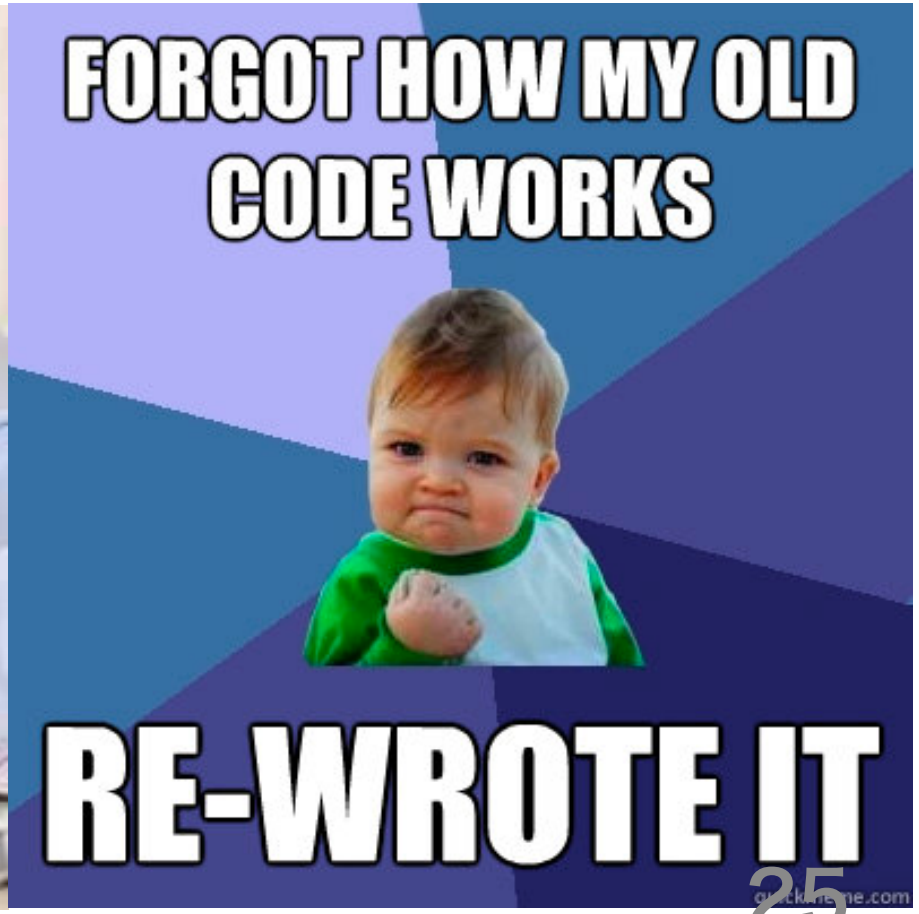


よくある質問

自分が書いたコードを再利用していますか？

- していません。数ヶ月前のコードはもう忘れています。





よくある質問

どうすればプログラミング技術を早く上達できますか？

研究・業務に取り入れて、毎日書くことで早く上達できます。

高校	C	挫折
学部	R	挫折
	Perl	研究に必要だったので、上達できた。今は忘れている。
大学院	R	研究に必要だったので、上達できた。
ポスドク	Python	共同研究者が Python を使ってたので、上達できた。

Python 基礎

- プログラミング言語
- 基本オブジェクト
- 基本文法

オブジェクト

コンピュータでデータを処理するためには、データや処理手順をコンピュータに保持する必要がある。Python では、オブジェクトと呼ばれている概念を用いてデータや処理手順を記憶させている。オブジェクトには様々な種類があります。

- スカラー
- リスト
- ディクショナリ
- セット
- タプル
- 関数
- クラス

基本オブジェクト

- スカラー
- リスト
- ディクショナリ

スカラー

オブジェクトには様々な種類があり、そのうち、値を 1 つだけしか保持できないオブジェクトをスカラーとよぶ。スカラーに値を保持させるためには、そのスカラーに値を付与する必要がある。この付与作業を代入という。

Python では、オブジェクトに値を代入するとき、代入演算子 "=" を使う。Python において、"=" は代入演算子のことであり、右辺の値またはオブジェクトを、左辺のオブジェクトに代入する機能を持つ。"=" は数学上の等しいという意味はない。

次のコードは、1 という値を、a という名のオブジェクトに代入することを表している。このコードを実行することによって、プログラムが終了するまで、a は 1 を保持している状態になる。

```
a = 1
```

```
a = 1
b = 1


c = a + b
# 2


d = c - 2
# ?


e = a + 2
# ?


f = d + e
# ?
```

スカラー

オブジェクトには様々な種類があり、そのうち、値を 1 つだけしか保持できないオブジェクトをスカラーとよぶ。スカラーに値を保持させるためには、そのスカラーに値を付与する必要がある。この付与作業を代入という。

Python では、オブジェクトに値を代入するとき、代入演算子 "=" を使う。Python において、"=" は代入演算子のことであり、右辺の値またはオブジェクトを、左辺のオブジェクトに代入する機能を持つ。"=" は数学上の等しいという意味はない。

次のコードは、1 という値を、a という名のオブジェクトに代入することを表している。このコードを実行することによって、プログラムが終了するまで、a は 1 を保持している状態になる。

```
a = 1
```

```
a = 1
b = 1


c = a + b
# 2


d = c - 2
# 0


e = a + 2
# 3


f = d + e
# 3
```

標準出力

オブジェクトに代入された値は、コンピュータのメモリ上に保存される。ディスプレイ（標準出力）上に出力されない。ディスプレイに出力させたいときは、print 関数を使用する。

```
a = 1
b = 1


c = a + b
print(c)


d = c - 2
print(d)


e = a + 2
print(e)


f = d + e
print(f)
```

算術演算子

Python で四則演算を行うのに次のような演算子を使用する。割り算に関して、余と商を分けて求める演算子もある。

演算子	意味	例
+	加	7 + 4 = 11
-	減	7 - 4 = 3
*	乗	7 * 4 = 28
/	除	7 / 4 = 1.75
%	余	7 % 4 = 3
//	商	7 // 4 = 1
**	累乗	2 ** 10 = 1024

※小数に対しても余と商を求めることができる。a//b は、a を b で割った後に整数部分を返す。また、a%b は、a - a//b*b で計算された結果が返される。

```
a = 11
b = 3


a + b


a - b


a * b


a / b


a % b


a // b
```

算術演算子

Python で四則演算を行うのに次のような演算子を使用する。割り算に関して、余と商を分けて求める演算子もある。

演算子	意味	例
+	加	7 + 4 = 11
-	減	7 - 4 = 3
*	乗	7 * 4 = 28
/	除	7 / 4 = 1.75
%	余	7 % 4 = 3
//	商	7 // 4 = 1
**	累乗	2 ** 10 = 1024

※小数に対しても余と商を求めることができる。a//b は、a を b で割った後に整数部分を返す。また、a%b は、a - a//b*b で計算された結果が返される。

```
a = 11
b = 3


a + b
# 14
a - b
# 8
a * b
# 33
a / b
# 3.6666666666665
a % b
# 2
a // b
# 3
```

問題S1-1

赤色で書かれたオブジェクト（スカラー）が保持している値を答えよ。

Google Colab を使用すると、Python を手軽に始められる！

```
a = 1
b = 1

c = a + b

a = 2

d = a + b

e = c + d
```

```
a = 9
b = 2

k = a + a // b

l = c % 2 * 5

m = a // 2 - b * 2

n = k + 1 + m
```

基本オブジェクト

- スカラー
- リスト
- ディクショナリ

リスト

リストは、同じ属性を持った値のまとまりである。例えば、大豆 5 個体の乾燥重量データを解析したいと仮定する。5 個体の乾燥重量をすべてスカラーオブジェクトに保存しようとすると、5 つのオブジェクトを用意する必要がある。これに対して、リストオブジェクトを使えば、5 つの値を 1 つのオブジェクトに保存できるようになる。リストを使うことで次のようなメリットがある。

- 入力回数が減り、入力漏れや入力ミスなどが少なくなる。また、プログラム全体のコード量が少なく、全体を見渡しやすくなる。
- 全データを対象とした演算処理（合計・平均など）が容易に行えるようになる。
- すべての要素に対して、同じ処理を繰り返したいときに、処理を 1 回だけ書けばいい。

```
weight_1 = 12
weight_2 = 10
weight_3 = 11


weights = [12, 10, 11]
```

リスト

例えば、同じ属性を複数の要素に対して平均値を求めたいとき、スカラーの場合は、すべてのオブジェクトの合計値を計算し、その合計値をオブジェクトの個数で割る計算を行う。

```
weight_1 = 12
weight_2 = 10
weight_3 = 11


n = 5
s = weight_1 + weight_2 + weight_3


mean = s / n
```

リストの全要素に対して平均値を求めたいとき、全要素を束ねて一括で計算することができる。

```
weights = [12, 10, 11]

n = len(weights)
s = sum(weights)


mean = s / n
```

あるオブジェクトを代入すると、他のオブジェクトが返ってくるオブジェクトを関数という。

関数	機能
len(x)	リスト x の要素数を調べる
sum(x)	リスト x の全要素の合計を求める

リスト

リストは、複数の要素をカンマ区切りで並べて、その外側を角括弧で囲むことで作られる。リストに含まれる要素は、束として扱われる。そのため、リストに対する演算や操作などは、リスト中の全要素に対して行われる。リストの特徴を調べるとき、右にあるような機能を使うと便利である。

リストは基本的に全要素を束ねて使うことが多いが、一部の要素だけを取り出したり、変更したりすることができる。その際、リストの先頭から数えて何番目の要素を取り出したいのかを、整数で指定する必要がある。ただし、Python では 0 番から数え始めることに注意。

```
w = [12, 10, 11, 13, 11]

w[0]

w[1]

w[2]



w[2] = 9

w
```

リスト

リストは、複数の要素をカンマ区切りで並べて、その外側を角括弧で囲むことで作られる。リストに含まれる要素は、束として扱われる。そのため、リストに対する演算や操作などは、リスト中の全要素に対して行われる。リストの特徴を調べるとき、右にあるような機能を使うと便利である。

リストは基本的に全要素を束ねて使うことが多いが、一部の要素だけを取り出したり、変更したりすることができる。その際、リストの先頭から数えて何番目の要素を取り出したいのかを、整数で指定する必要がある。ただし、Python では 0 番から数え始めることに注意。

```
w = [12, 10, 11, 13, 11]


w[0]
# 12
w[1]
# 10
w[2]
# 11


w[2] = 9


w
# [12, 10, 9, 13, 11]
```

※w の２番目の位置に9を代入する。

問題L1-1

赤色で書かれたオブジェクトが保持している値を答えよ。

```
w = [1, 3, 5, 7, 9]

w[2] = 4
w[0] = w[1]
w[4] = w[1] + 2
w[3] = w[3] * 2

w
```

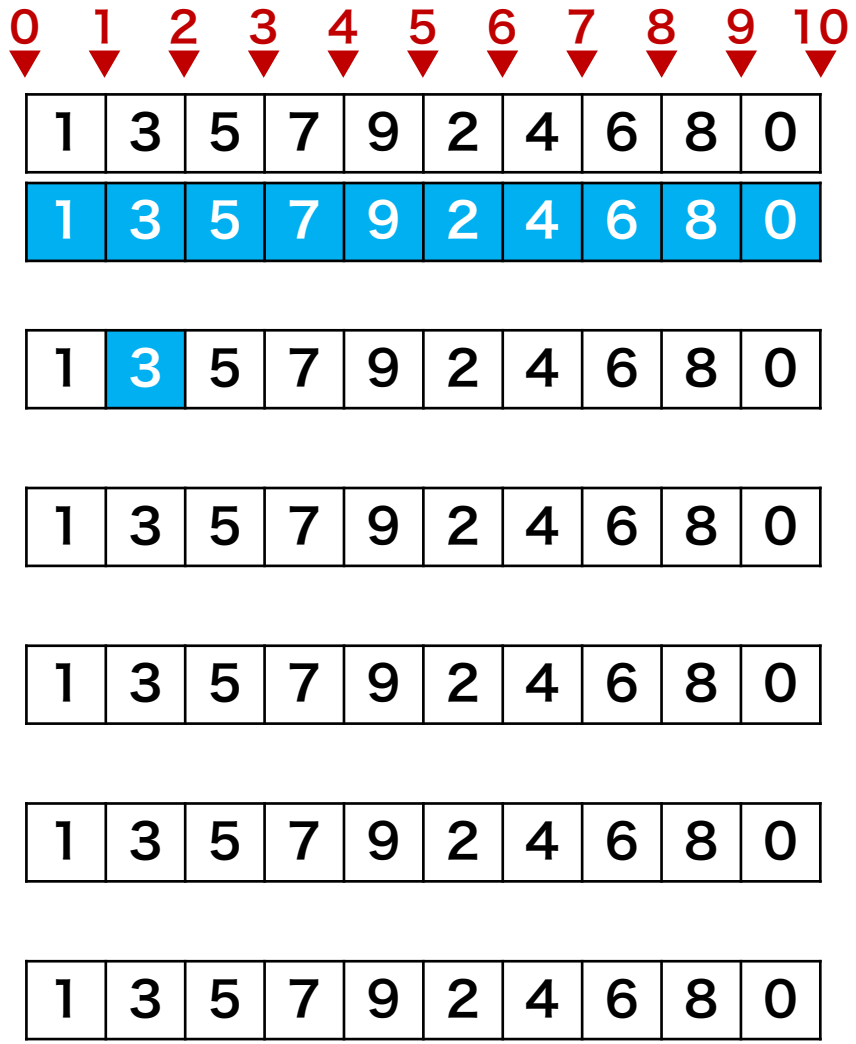
```
w = [0, 2, 4, 6, 8]

w[4] = w[0] + w[3]
w[3] = w[0] + w[2]
w[2] = w[0] + w[1]
w[1] = w[0] + w[0]

w
```

リスト

リストから連続して並んでいる要素をまとめて取り出す（スライス）とき、":" を使うと便利な場合がある。

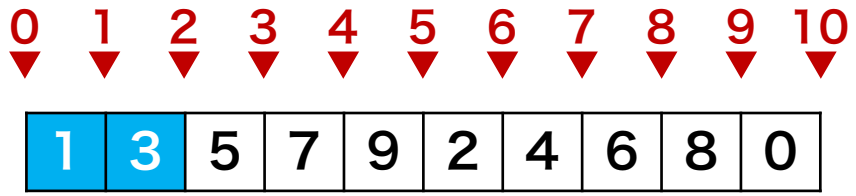
```
a = [1, 3, 5, 7, 9, 2, 4, 6, 8, 0]


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


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


a[2:6]


a[:4]


a[5:]
```

リスト



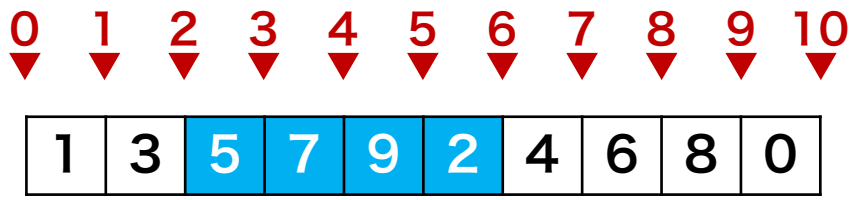

```
a = [1, 3, 5, 7, 9, 2, 4, 6, 8, 0]


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


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


a[2:6]


a[:4]


a[5:]
```

リスト



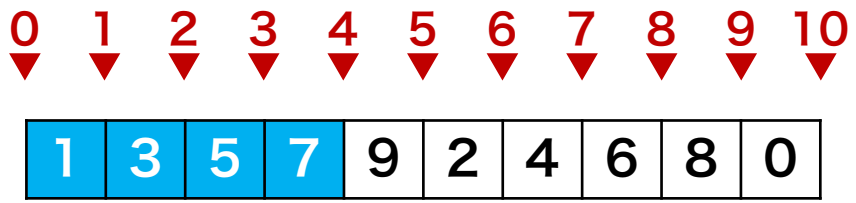

```
a = [1, 3, 5, 7, 9, 2, 4, 6, 8, 0]


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


a[1]
# 3
a[0:2]
# [1, 3]
a[2:6]


a[:4]


a[5:]
```

リスト



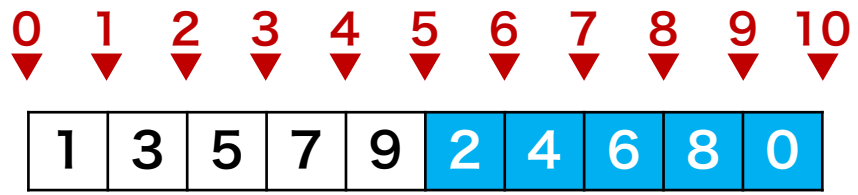

```
a = [1, 3, 5, 7, 9, 2, 4, 6, 8, 0]


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


a[1]
# 3
a[0:2]
# [1, 3]
a[2:6]
# [5, 7, 9, 2]
a[:4]


a[5:]
```

リスト




```
a = [1, 3, 5, 7, 9, 2, 4, 6, 8, 0]


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


a[1]
# 3
a[0:2]
# [1, 3]
a[2:6]
# [5, 7, 9, 2]
a[:4]
# [1, 3, 5, 7]
a[5:]
```

リスト




```
a = [1, 3, 5, 7, 9, 2, 4, 6, 8, 0]


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


a[1]
# 3
a[0:2]
# [1, 3]
a[2:6]
# [5, 7, 9, 2]
a[:4]
# [1, 3, 5, 7]
a[5:]
# [2, 4, 6, 8, 0]
```

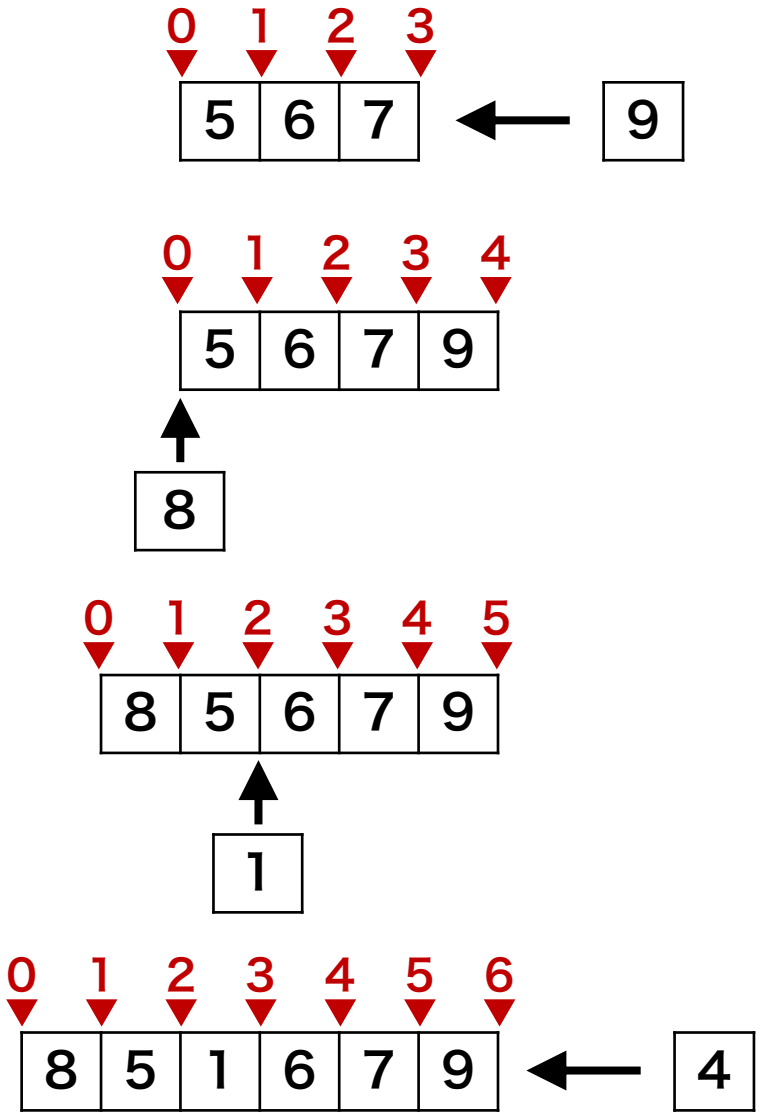
リスト操作

すでに作られたリストに新たな要素を追加することができる。リストの後尾に要素を追加するには append 関数（メソッド）を使用する。また、リストの先頭あるいは指定した位置に要素を挿入するには insert 関数（メソッド）を使用する。

関数	機能
append	リストの後尾に要素を 1 つだけ追加する。
extend	リストの後尾に要素を複数個追加する。
insert	リストの位置 i に要素を 1 つだけ挿入する。
pop	リストの位置 i にある要素を削除する。

```
a = [5, 6, 7]


a.append(9)
a


a.insert(0, 8)
a


a.insert(2, 1)
a


a.append(4)
a
```

リスト操作



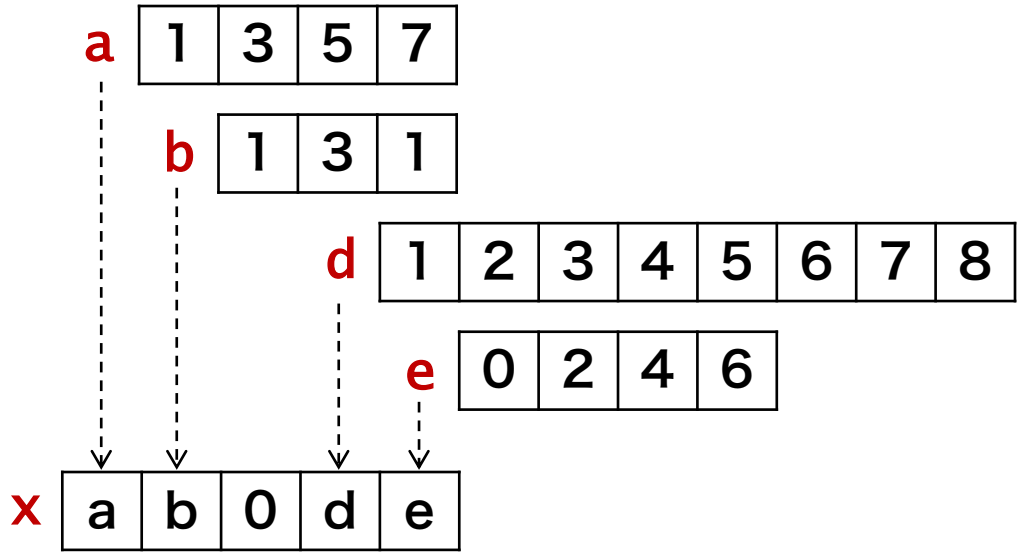

```
a = [5, 6, 7]


a.append(9)
a
# [5, 6, 7, 9]
a.insert(0, 8)
a
# [8, 5, 6, 7, 9]
a.insert(2, 1)
a
# [8, 5, 1, 6, 7, 9]
a.append(4)
a
# [8, 5, 1, 6, 7, 9, 4]
```

二次元リスト

リストは複数の要素を含むことができる。これまでに要素が数値の場合を見てきた。スカラーで見たように、スカラーは数値と文字列の両方を保持できる。これと同様に、リストの要素も数値だけでなく、文字列にすることもできる。また、正確に言えば、リストの各要素は、オブジェクトとすることができる。

リストはオブジェクトの 1 種である。リストの要素をオブジェクトにすることができる。つまり、リストの中にリストを入れることができる。このようなリストを二次元リストなどと呼んでいる。

二次元リストは、表型データの解析ではあまり使われていない。その代わり、二次元リストを拡張した NumPy 二次元配列あるいは Pandas データフレームがよく使われている。二次元配列とデータフレームは、データを行列のように扱えるため、非常に取り扱いやすい。

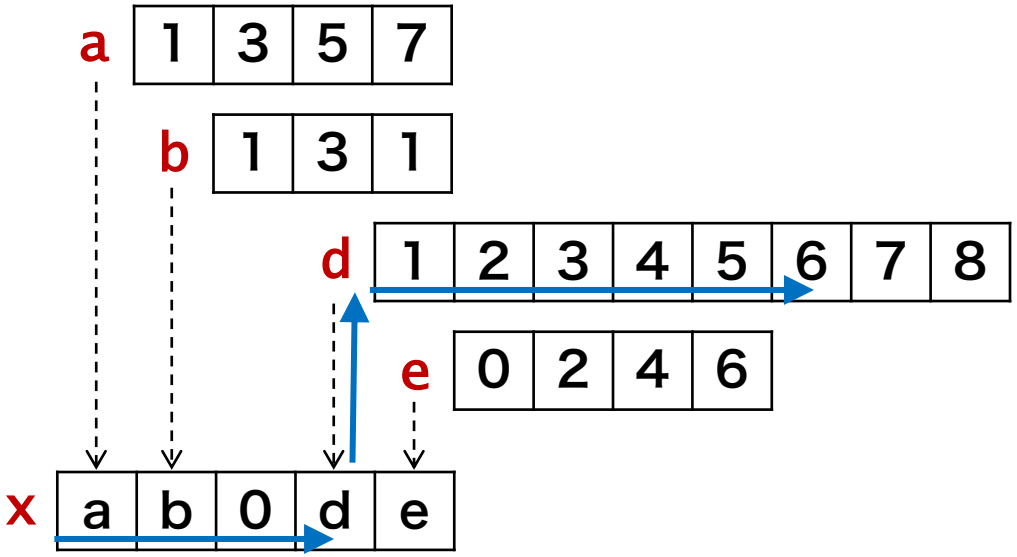



```
a = [1, 3, 5, 7]
b = [1, 3, 1]
d = [1, 2, 3, 4, 5, 6, 7, 8]
e = [0, 2, 4, 6]


x = [a, b, 0, d, e]
```

二次元リスト

二次元リストから要素を取り出すときは、親リストに対して位置を指定して要素を取り出し、次に取り出した要素（子リスト）に対して再度位置を指定して要素を取り出す。

```
a = [1, 3, 5, 7]
b = [1, 3, 1]
d = [1, 2, 3, 4, 5, 6, 7, 8]
e = [0, 2, 4, 6]
x = [a, b, 0, d, e]


x[3][5]
# 6
```

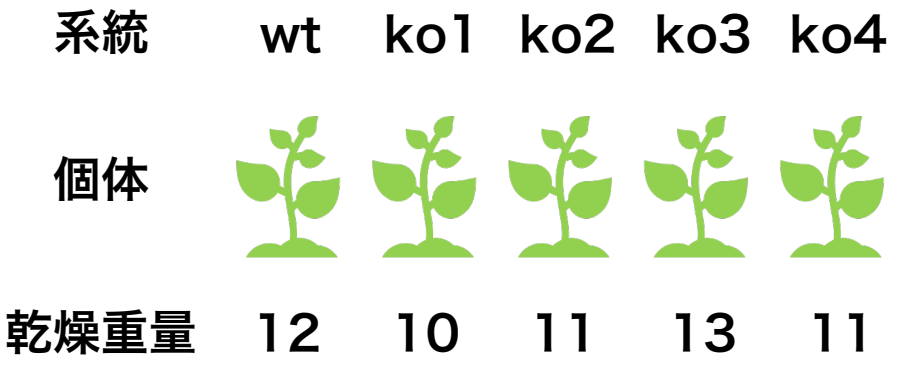


基本オブジェクト

- スカラー
- リスト
- ディクショナリ

ディクショナリ

ディクショナリは、リストと同じく、複数の要素（オブジェクト）を一括に保存できるオブジェクトである。リストは、各要素を位置番号で管理しているのに対して、ディクショナリは各要素を名前（キー）で管理している。




```
weights = {
    'wt': 12,
    'ko1': 10,
    'ko2': 11,
    'ko3': 13,
    'ko4': 11
}
```

key value

ディクショナリ

ディクショナリの要素を取り出すときは、キーを使う。キーは文字列であるから、両端に引用符を与える必要がある。

```
weights = {
    'wt': 12,
    'ko1': 10,
    'ko2': 11
}

weights['wt']
# 12

weights['ko1']



weights['ko5']
```

ディクショナリ

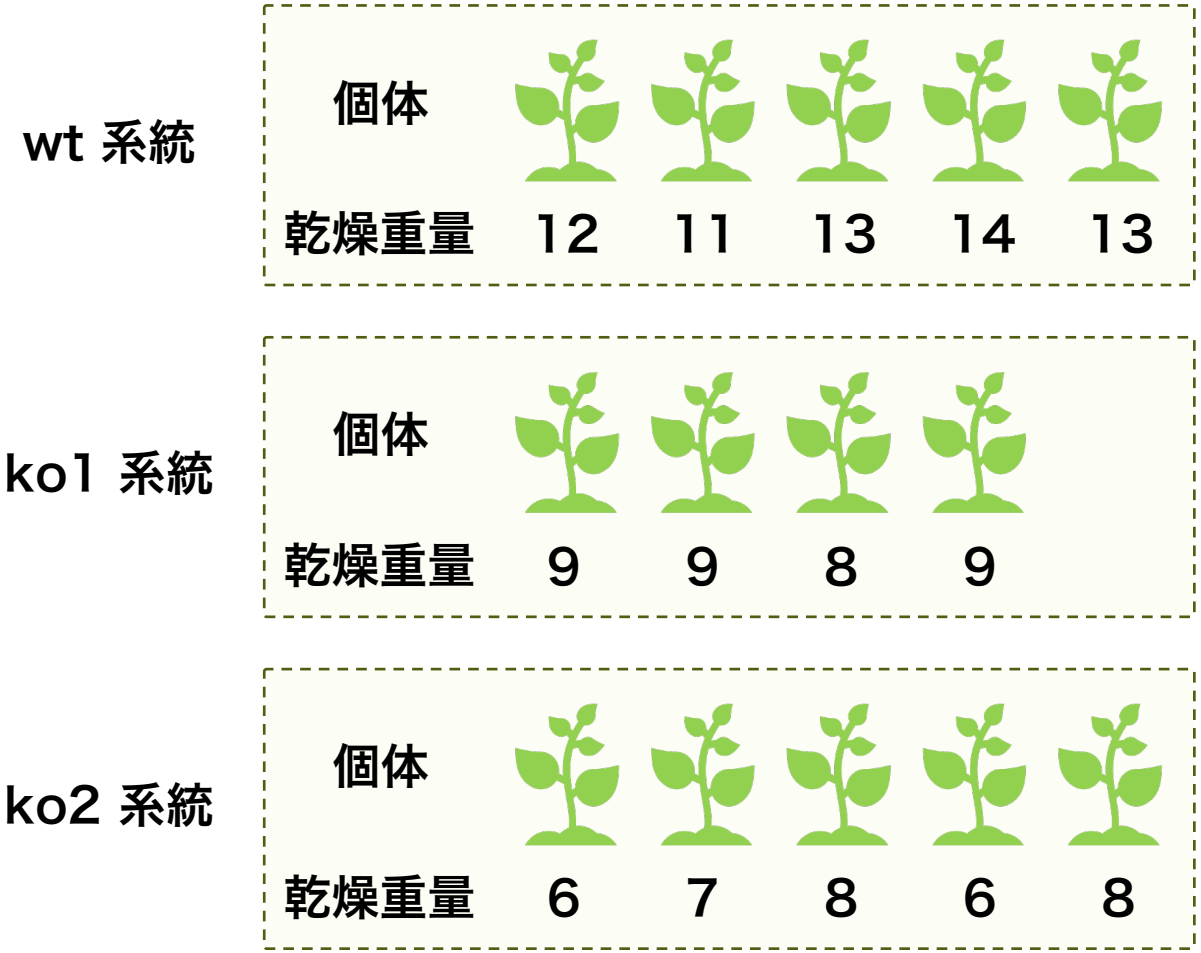
ディクショナリの要素を取り出すときは、キーを使う。キーは文字列であるから、両端に引用符を与える必要がある。

```
weights = {
    'wt': 12,
    'ko1': 10,
    'ko2': 11
}

weights['wt']
# 12

weights['ko1']
# 10

weights['ko5']
# KeyError
```

ko5 が定義されていないためにエラーが起こる。

ディクショナリ

リストの中にリストを代入できると同様に、ディクショナリの中にリストを代入することもできる。

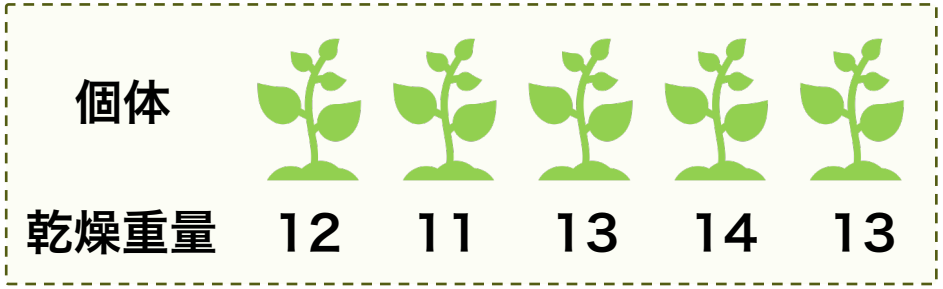



```
weights = {
    'wt': [12, 11, 13, 14, 13],
    'ko1': [9, 9, 8, 9],
    'ko2': [6, 7, 8, 6, 8]
}
```

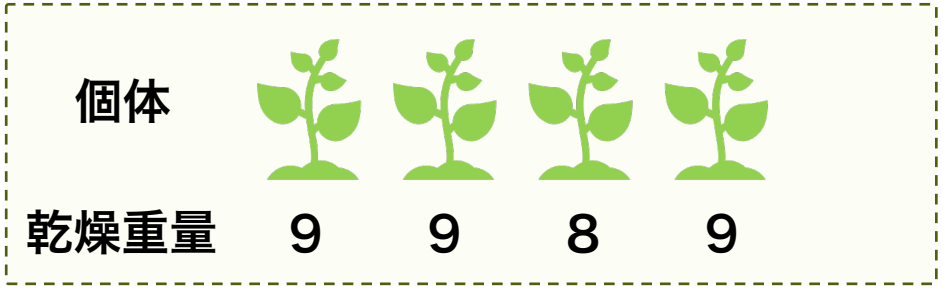
ディクショナリ

ディクショナリの要素を取り出すときは、キーを使う。キーは文字列であるから、両端に引用符を与える必要がある。

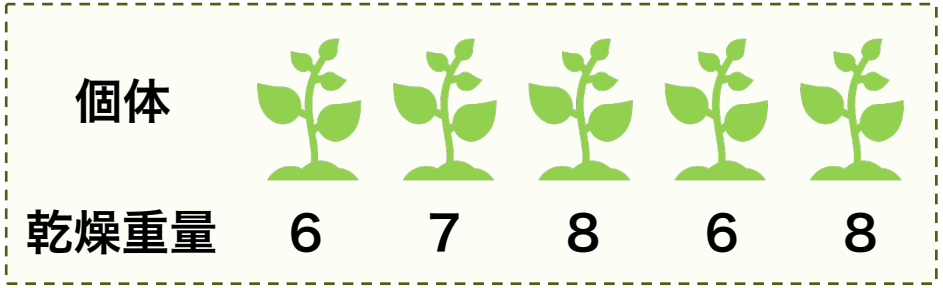
wt 系統



ko1 系統



ko2 系統




```
weights = {
    'wt': [12, 11, 13, 14, 13],
    'ko1': [9, 9, 8, 9],
    'ko2': [6, 7, 8, 6, 8]
}


weights['wt']
# [12, 11, 13, 14, 13]
weights['wt'][2]
# 13
weights['ko2'][1:4]
# [7, 8, 6]
```

ディクショナリ

ディクショナリの要素を取り出すときは、キーを使う。キーは文字列であるから、両端に引用符を与える必要がある。ディクショナリの各要素にリストが代入されている場合は、ディクショナリとリストの要素の取り出し方法を併用すればいい。ディクショナリの要素の値を変更するとき、キーを指定して変更する。

```
weights = {
    'wt': [12, 11, 13, 14, 13],
    'ko1': [9, 9, 8, 9],
    'ko2': [6, 7, 8, 6, 8]
}
weights['wt'] = [2, 2, 3]
weights
# {'wt': [2, 2, 3], 'ko1': [9, 9, 8, 9],
'ko2': [6, 7, 8, 6, 8]}
```

ディクショナリ

ディクショナリの要素の値を変更するとき、キーを指定して変更する。また、ディクショナリに要素を追加するときは、update 関数（メソッド）を使用する。ただし、キーがすでに存在している場合は、その値は上書きされる。

```
weights = {
    'wt': [12, 11, 13, 14, 13],
    'ko1': [9, 9, 8, 9],
    'ko2': [6, 7, 8, 6, 8]
}
weights.update({'ko3': [1, 3, 1]})
weights
# {'wt': [12, 11, 13, 14, 13], 'ko1': [9, 9, 8, 9], 'ko2': [6, 7, 8, 6, 8], 'ko3': [1, 3, 1]}


weights.update({'wt': [0, 0, 0]})
weights
# {'wt': [0, 0, 0], 'ko1': [9, 9, 8, 9], 'ko2': [6, 7, 8, 6, 8], 'ko3': [1, 3, 1]}
```

ディクショナリ

ディクショナリから要素を削除するときは、pop 関数（メソッド）を使用する。

```
weights = {
    'wt': [12, 11, 13, 14, 13],
    'ko1': [9, 9, 8, 9],
    'ko2': [6, 7, 8, 6, 8]
}
weights.pop('ko1')
weights
# {'wt': [12, 11, 13, 14, 13], 'ko2': [6, 7, 8, 6, 8]}



weights.pop('ko1')
# KeyError
```

ko1 がすでに削除されて、ディクショナリ weights に存在していない。

参考

リスト
- 複数の要素を0から始まる整数のインデックス番号で管理する
- 順序情報が含まれている
- 同じ値を複数保持できる

ディレクショナリ
- 複数の要素を名前（キー）で管理する
- 順序情報は含まれていないが、キーまたは値で並べ替えが可能
- 複数の同じ値を保持できるが、キーは 1 つだけしか保持できない

タプル
- リスト同じ
- ただし、一度作成すると、あとで要素の値を変更できなくなる

セット
- リストと同じ
- ただし、同じ値を複数保持できない

Python 基礎

- プログラミング言語
- 基本オブジェクト
- 基本文法

基本文法

- 予約語
- 条件構文
- 繰り返し構文
- 関数
- パッケージ

基本文法

- 予約語
- 条件構文
- 繰り返し構文
- 関数
- パッケージ

予約語

状態判定		条件構文	繰り返し構文		関数		例外処理	いろいろ	
True	is	if	for		def	import	try	lambda	async
False	not	elif	while		return	from	except	del	await
None	in	else	pass		yield	as	finally	zip	global
	or		break			class	raise	assert	with
	and		continue					nonlocal	
			enumerate						

基本文法

- 予約語
- 条件構文
- 繰り返し構文
- 関数
- パッケージ

if 構文

Python の条件構文は if、elif、else などの単語を使う。Python の条件構文は必ず if から始まる。条件構文の 1 行目には、if とともに条件を書く。条件判別後の処理は、2 行目以降に書く。ただし、条件判別後の処理は、条件構文の一部であることを明示するために、行の先頭にインデントを入れる。右の例は、「もし a > 1ならば、b に 1 を代入し、c に 2 を代入する。」ことを表している。

```
a = 2
b = 0
c = 0

if (a > 1) is True :
    b = 1
    c = 2

d = b + c
d
```

if 構文

```
a = 2
b = 0
c = 0

if (a > 1) is True :
    b = 1
    c = 2

d = b + c
d
```

判定条件

条件構文の開始

インデント

条件構文の終了

条件構文ブロック

インデントの数で、条件構文ブロックが終了しているかどうかを判定している。

インデント

インデントは、タブまたはスペースキーで入力する。

- タブ 1 個分
- スペース 4 個分 ✓
- スペース 2 個分

英語キーボード

日本語キーボード

if 構文

```
a = 2
b = 0
c = 0

        判定条件

if (a > 1) is True :

    b = 1
    c = 2


d = b + c
d
```

真（True）判定の場合は、省略するのが一般的である。

```
a = 2
b = 0
c = 0

if a > 1:

    b = 1
    c = 2


d = b + c
d
```

論理演算子

条件構文は、与えられた条件が真（True）であるかどうかを判定して、分岐処理を行う構文である。条件は不等式で与えるのが一般的である。

論理演算子	処理
a == b	a と b が等しいならば True
a != b	a と b が等しくなければ True
a > b	a が b よりも大きければ True
a >= b	a が b 以上ならば True
a < b	a が b よりも小さければ True
a <= b	a が b 以下ならば True

※表中の論理演算子のほか、is と is not も理論演算に使われる。"a is b" は a と b が同じ属性（かつ同じ値）を持ったオブジェクトである場合に True を返す演算子である。気になる方は、各自で調べてください。

```
a = 4
b = 3



if a > 3:
    c = 1
c
# 1



if b > 10:
    d = 1
d
# 0
```

確認問題

赤色で書かれたオブジェクトが保持している値を答えよ。

```
a = 1
b = 0

if a > 1:
    b = 1




b
```

```
a = 1
b = 1
c = 0

if a > 0:
    if b > 1:
        c = 1


c = 3


c
```

論理演算

条件構文は、与えられた条件が真（True）であるかどうかを判定して、分岐処理を行う構文である。複数の条件を同時に判断することもできる。論理演算で最もよく使われる演算として論理積と論理和である。論理積は、英語の AND と日本語の「かつ」に相当するものである。論理和は、英語の OR と日本語の「または」に相当するものである。

論理演算子	演算
a and b	論理積
a & b	論理積
a or b	論理和
a | b	論理和
a ^ b	排他的論理和

```
a = 4
b = 3
c = 0

x = (a > 0)
y = (b > 0)

if x and y:
  c = 1

c
# 1
```

論理演算例

条件 1	条件 2	論理積	論理和	排他的論理和
x	y	x & y	x | y	x ^ y
真	真	真	真	偽
真	偽	偽	真	真
偽	真	偽	真	真
偽	偽	偽	偽	偽

確認問題

赤色で書かれたオブジェクトが保持している値を答えよ。

```
a = 1
b = 0
c = 0

if (a > 0) and (b > 0):
    c = 1

c
```

```
a = 0
b = 1
c = 0

if (a > 0) or (b < 2):
    c = 1

c
```

if-else 構文

これまでに見てきた条件構文は「もし〜ならば・・・をする」のように、条件を満たせば処理を行うものであった。

これに対して、ある条件を満たした時に処理 A を、満たさなかった時に処理 B を行いたい場合は、if 文を続けて 2 つ書けばいい。しかし、if 文を 2 つ続けて書くことで、両者は独立に扱われる。つまり、同じ条件を 2 回判断していることに等しい。理論的にも、処理的にも煩雑である。このような場合は、if と else を用いて条件構文を作成すると、理論的にも処理的にもシンプルになる。

もし a > 60 ならば b に passing を代入する。その後、もし a ≤ 60 ならば b に failing を代入する。

```
a = 81
b = ''
if a > 60 :
    b = 'passing'
if a <= 60:
    b = 'failing'
b
# passing
```

もし a > 60 ならば b に passing を代入し、さもければ b に failing を代入する。

```
a = 81
b = ''
if a > 60 :
    b = 'passing'
else:
    b = 'failing'
b
# passing
```

確認問題

赤色で書かれたオブジェクトが保持している値を答えよ。

```
a = 1
b = 0

if a > 0:
    b = 1
else:
    b = 2
```

b

```
a = 1
b = 0
c = 0

if a > 0:
    if b > 0:
        c = 1
    else:
        c = 2
```

c

確認問題

赤色で書かれたオブジェクトが保持している値を答えよ。

```
a = 1
b = 1
c = 0


if a > 0:
    if b => 1:
        d = 1
    else:
        d = 2
else:
    d = 3


e = c + d
```

e

```
a = 1
b = 1
c = 0


if a > 0:
    if b > 1:
        c = 1
        d = 2

    b = 2



c = b + 1
```

c

if-elif-else 構文

1 つのオブジェクトに対して複数の閾値を設けて条件判定したい場合は、複数の if 文を使って書くことができる。しかし、この場合、ロジックも複雑になることに注意する必要がある。

得点	成績
90 点超え	excellent
80 点超え 90 以下	good
60 点超え 80 以下	passing
60 以下	failing

```
a = 81
b = ''

if a <= 60:
    b = 'failing'

if a > 60:
    b = 'passing'

if a > 80:
    b = 'good'

if a > 90 :
    b = 'excellent'
b
```

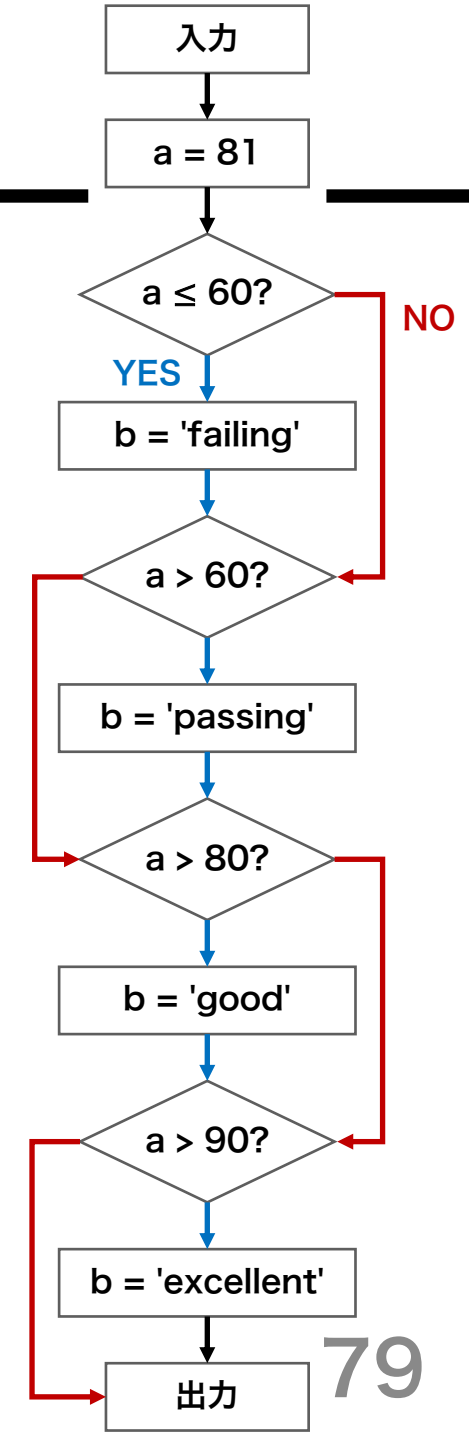
```
a = 81
b = ''

if a > 90 :
    b = 'excellent'
elif a > 80:
    b = 'good'
elif a > 60:
    b = 'passing'
else:
    b = 'failing'

b
```

if-elif-else 構文

1 つのオブジェクトに対して複数の閾値を設けて条件判定したい場合は、複数の if 文を使って書くことができる。しかし、この場合、ロジックも複雑になることに注意する必要がある。

得点	成績
90 点超え	excellent
80 点超え 90 以下	good
60 点超え 80 以下	passing
60 以下	failing

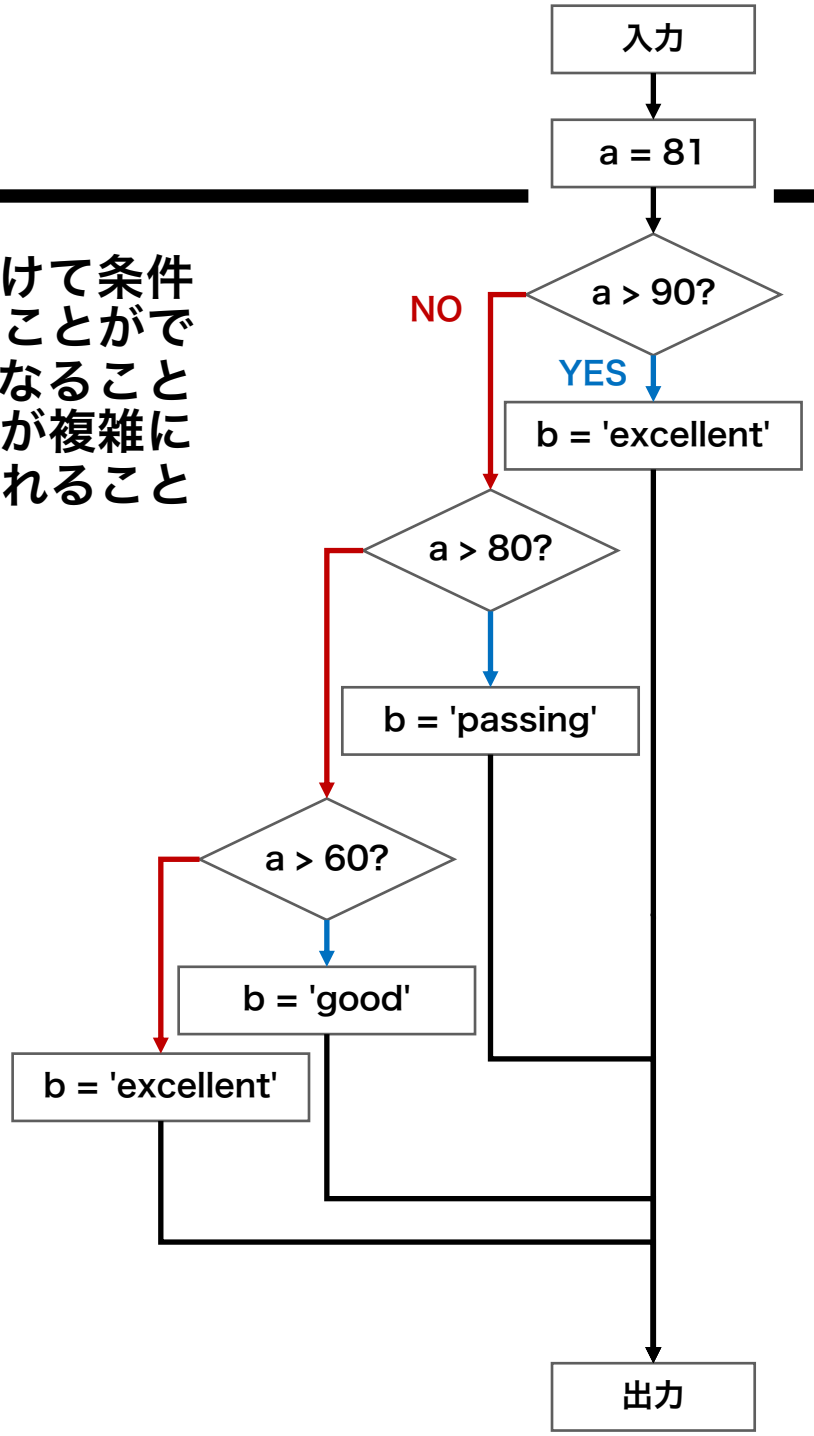
```
a = 81
b = ''

if a <= 60:
    b = 'failing'

if a > 60:
    b = 'passing'

if a > 80:
    b = 'good'

if a > 90 :
    b = 'excellent'
b
```



if-elif-else 構文

1 つのオブジェクトに対して複数の閾値を設けて条件判定したい場合は、複数の if 文を使って書くことができる。しかし、この場合、ロジックも複雑になることに注意する必要がある。 そのため、ロジックが複雑にならないようにするために、if-else 文が使われることが多い。

得点	成績
90 点超え	excellent
80 点超え 90 以下	good
60 点超え 80 以下	passing
60 以下	failing




```
a = 81
b = ''

if a > 90 :
    b = 'excellent'
elif a > 80:
    b = 'good'
elif a > 60:
    b = 'passing'
else:
    b = 'failing'
```

```
b
```

80

if-elif-else 構文

1 つのオブジェクトに対して複数の閾値を設けて条件判定したい場合は、複数の if 文を使って書くことができる。しかし、この場合、ロジックも複雑になることに注意する必要がある。 そのため、ロジックが複雑にならないようにするために、if-else 文が使われることが多い。

得点	成績
90 点超え	excellent
80 点超え 90 以下	good
60 点超え 80 以下	passing
60 以下	failing

```
a = 81
b = ''

if a <= 60:
    b = 'failing'

if a > 60:
    b = 'passing'

if a > 80:
    b = 'good'

if a > 90 :
    b = 'excellent'
b
```

```
a = 81
b = ''

if a > 90 :
    b = 'excellent'
elif a > 80:
    b = 'good'
elif a > 60:
    b = 'passing'
else:
    b = 'failing'

b
```

if-elif-else 構文（順序に注意）

1 つのオブジェクトに対して複数の閾値を設けて条件判定したい場合は、複数の if 文を使って書くことができる。しかし、この場合、ロジックも複雑になることに注意する必要がある。ロジックが複雑にならないようにするために、if-else 文が使われることが多い。ただし、いずれの場合も、条件判定の順序に注意する必要がある。

得点	成績
90 点超え	excellent
80 点超え 90 以下	good
60 点超え 80 以下	passing
60 以下	failing

```
a = 81
b = ''
if a > 90 :
    b = 'excellent'
if a > 80:
    b = 'good'
# good
if a > 60:
    b = 'passing'
# passing
if a <= 60:
    b = 'failing'

b
# passing
```

```
a = 81
b = ''

if a > 60:
    b = 'passing'
elif a > 80:
    b = 'good'
elif a > 90:
    b = 'excellent'
else:
    b = 'failing'


b
# passing
```

確認問題

赤色で書かれたオブジェクトが保持している値を答えよ。

```
a = 11
s = ''
if (3 <= a) and (a <= 5):
    s = 'spring'
elif (6 <= a) and (a <= 8):
    s = 'summer'
elif (9 <= a) and (a <= 11):
    s = 'autumn'
else:
    s = 'winter'
s
```

```
a = 4
d = ''
if a == 1:
    d = 'Mon'
elif a == 2:
    d = 'Tue'
elif a = 3:
    d = 'Wed'
elif a == 4:
    d = 'Thu'
ellf a == 5:
    d = 'Fri'
elif a == 6:
    d = Sat'
elif a == 7:
    d = 'Sun'
d
```

基本文法

- 予約語
- 条件構文
- 繰り返し構文
- 関数
- パッケージ

while 構文

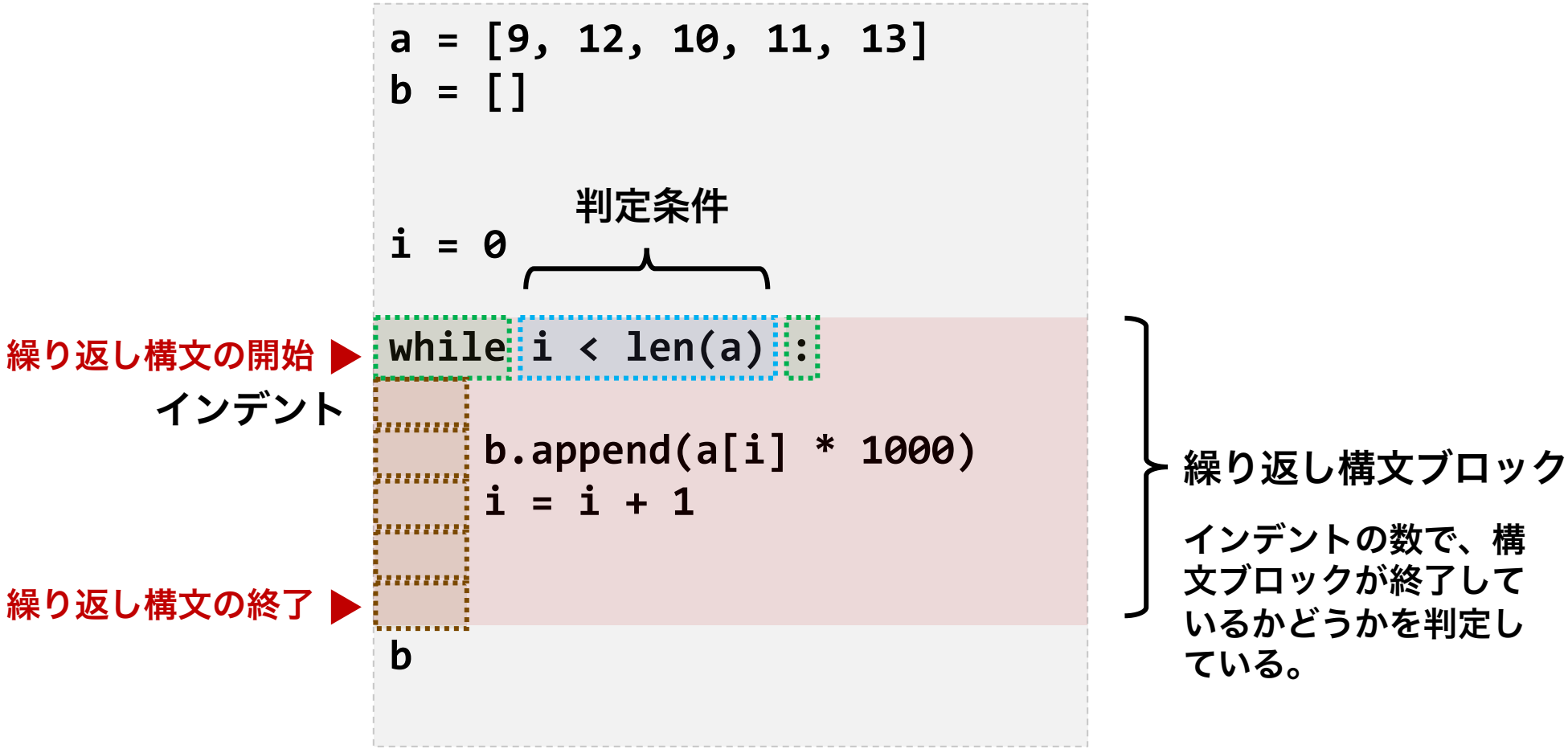
繰り返し構文は、複数のオブジェクトに対して、同じ処理を繰り返したい場合に使う構文である。そのためには、それらの複数のオブジェクトをあらかじめリスト等に代入しておく必要がある。

例えば、あるリスト中の各要素を 1000 倍する処理を行いたいとき、繰り返し構文を使わない場合は、各要素それぞれに対して 1000 倍の計算を行う必要がある。この操作は、リストの要素数分だけ処理しなければならない。これに対して、while文を用いると、1000 倍の計算式を 1 度書くだけで十分。

```
a = [9, 12, 10, 11, 13]
b = [0,  0,  0,  0,  0]
b[0] = a[0] * 1000
b[1] = a[1] * 1000
b[2] = a[2] * 1000
b[3] = a[3] * 1000
b[4] = a[4] * 1000
b
```

```
a = [9, 12, 10, 11, 13]
b = []
i = 0
while i < len(a):
    b.append(a[i] * 1000)
    i = i + 1
b
```

while 構文


a = [9, 12, 10, 11, 13]
b = []
判定条件
i = 0
繰り返し構文の開始
while i < len(a) :
インデント
b.append(a[i] * 1000)
i = i + 1
繰り返し構文の終了
b
繰り返し構文ブロック
インデントの数で、構文ブロックが終了しているかどうかを判定している。

確認問題

赤色で書かれたオブジェクトが保持している値を答えよ。

プログラムの実行が完了しない場合は Jupyter Notebook の実行停止ボタン（■）をクリックして下さい。

```
a = [1, 1, 2, 3, 4, 5]
b = []
i = 0
while i < len(a):
    x = a[i]
    if x % 2 == 0:
        z = x / 2
    else:
        z = (x + 1) / 2
    b.append(z)
    i = i + 1
b
```

```
a = [1, 1, 2, 3, 4, 5]
b = []
i = 0
while i < len(a):
    x = a[i] * i
    b.append(x)
    i = a[i] - 1
b
```

確認問題 解答

赤色で書かれたオブジェクトが保持している値を答えよ。

```
a = [1, 1, 2, 3, 4, 5]
b = []
i = 0
while i < len(a):
    x = a[i]
    if x % 2 == 0:
        z = x / 2
    else:
        z = (x + 1) / 2
    b.append(z)
    i = i + 1
b
# [1, 1, 1, 2, 2, 3]
```

```
a = [1, 1, 2, 3, 4, 5]
b = []
i = 0
while i < len(a):
    x = a[i] * i
    b.append(x)
    i = a[i] - 1
b
```

i が常に 0 であるため、
無限ループになる。

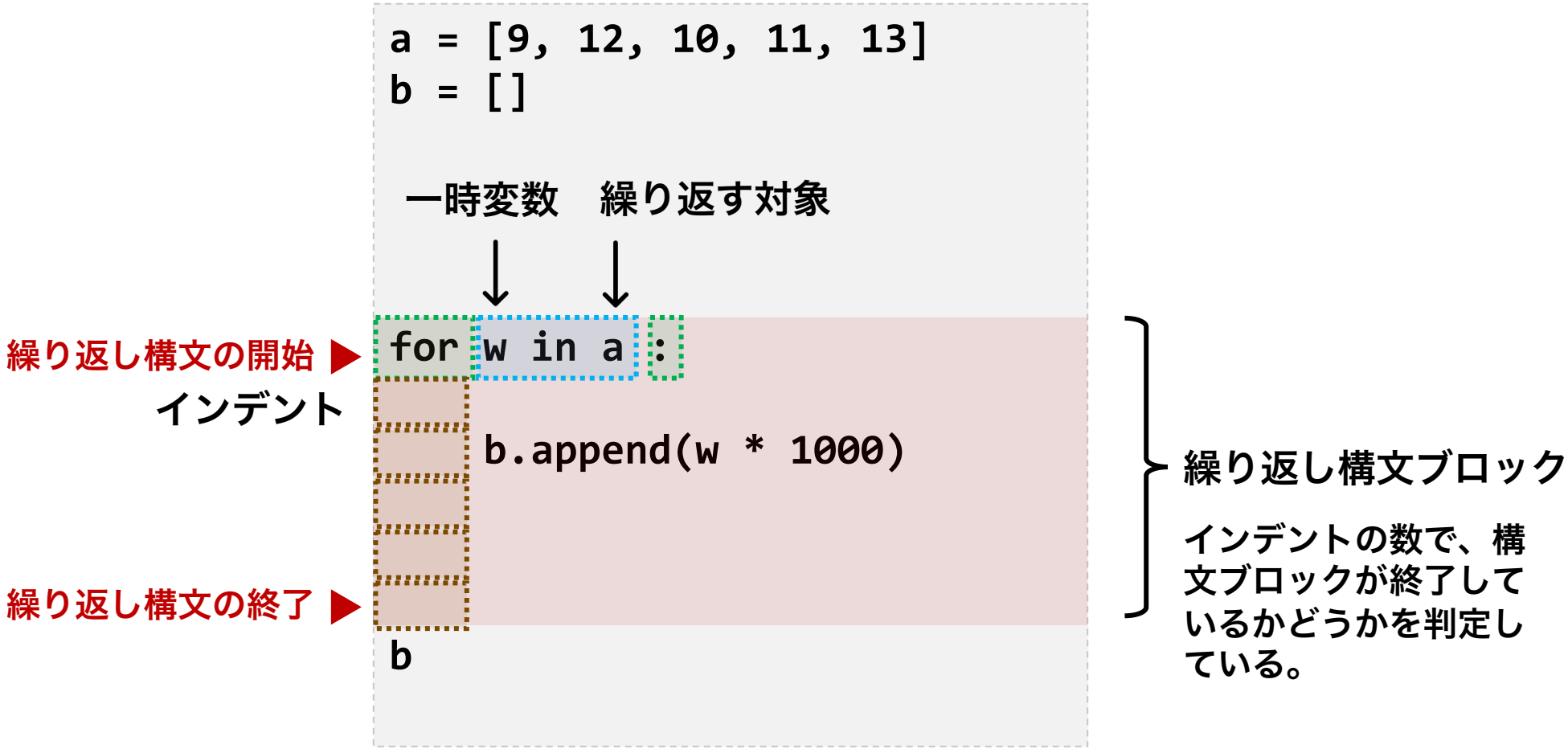
for 構文

無限ループにならないように while 文の代わりに for 文が一般的に使われている。for 文は、リストの要素数だけしか繰り返さない。

```
a = [9, 12, 10, 11, 13]
b = []
i = 0
while i < len(a):
    b.append(a[i] * 1000)
    i = i + 1
b
```

```
a = [9, 12, 10, 11, 13]
b = []
for w in a:
    b.append(w * 1000)
b
```

for 構文


a = [9, 12, 10, 11, 13]
b = []
一時変数
繰り返す対象
for w in a :
b.append(w * 1000)
b
繰り返し構文の開始
インデント
繰り返し構文の終了
繰り返し構文ブロック
インデントの数で、構文ブロックが終了しているかどうかを判定している。

for 構文

無限ループにならないように while 文の代わりに for 文が一般的に使われている。for 文は、リストの要素数だけしか繰り返さない。

for 文の中で、位置番号でリストの各要素を操作する場合は、len 関数と range 関数を組み合わせて使う。例えば range(4) の場合は、0, 1, 2, 3 が出力される。

```
a = [1, 1, 2, 3, 4, 5]
b = []
for w in a:
    b.append(w * 2)

b
```

```
a = [1, 1, 2, 3, 4, 5]
b = []
for i in range(len(a)):
    b.append(a[i] * 2)
b
```

for 構文

無限ループにならないように while 文の代わりに for 文が一般的に使われている。for 文は、リストの要素数だけしか繰り返さない。

for 文の中で、位置番号でリストの各要素を操作する場合は、len 関数と range 関数を組み合わせて使う。例えば range(4) の場合は、0, 1, 2, 3 が出力される。

for 文で、リスト中の要素とその位置番号を同時に取得したい場合は enumerate 関数を使用する。右例の場合、i には位置番号、w には要素の値が代入される。

```
a = [1, 1, 2, 3, 4, 5]
b = []
for w in a:
    b.append(w * 2)

b
```

```
a = [1, 1, 2, 3, 4, 5]
b = []
for i in range(len(a)):
    b.append(a[i] * 2)
b
```

```
a = [1, 1, 2, 3, 4, 5]
b = []
for i, w in enumerate(a):
    b.append(w * 2)
b
```

確認問題

リスト z の位置 i の要素が、リスト a の位置 i の要素とリスト b の位置 i の要素の和となるようなリスト z を作成せよ。

```
a = [1, 1, 2, 3, 4, 5]
b = [1, 3, 5, 7, 9, 10]
z = []









z
# [2, 4, 7, 10, 13, 15]
```

基本文法

- 予約語
- 条件構文
- 繰り返し構文
- 関数
- パッケージ

関数

同じ処理を繰り返すとき、その処理をまとめてパッケージ化することができる。このパッケージ化された処理群を関数という。Python で関数を作るには def と return をセットとして使う。右下は、平均を求める機能を関数として定義した例である。

```
a = [11, 12, 10, 11, 13]
b = [22, 23, 20, 16, 28]
c = [42, 41, 36, 41, 39]
d = [24, 21, 18, 21, 22]
e = [52, 54, 55, 46, 51]

sum(a) / len(a)
sum(b) / len(b)
sum(c) / len(c)
sum(d) / len(d)
sum(e) / len(e)


def mean(x):
    m = sum(x) / len(x)
    return m

mean(a)
mean(b)
mean(c)
mean(d)
mean(e)
```

関数

関数の定義の開始 ▶

インデント

関数の定義の終了 ▶

```
def mean(x):
    m = sum(x) / len(x)
    return m

a = [12, 11]
mean(a)

b = [5, 6]
mean(b)
```

関数ブロック

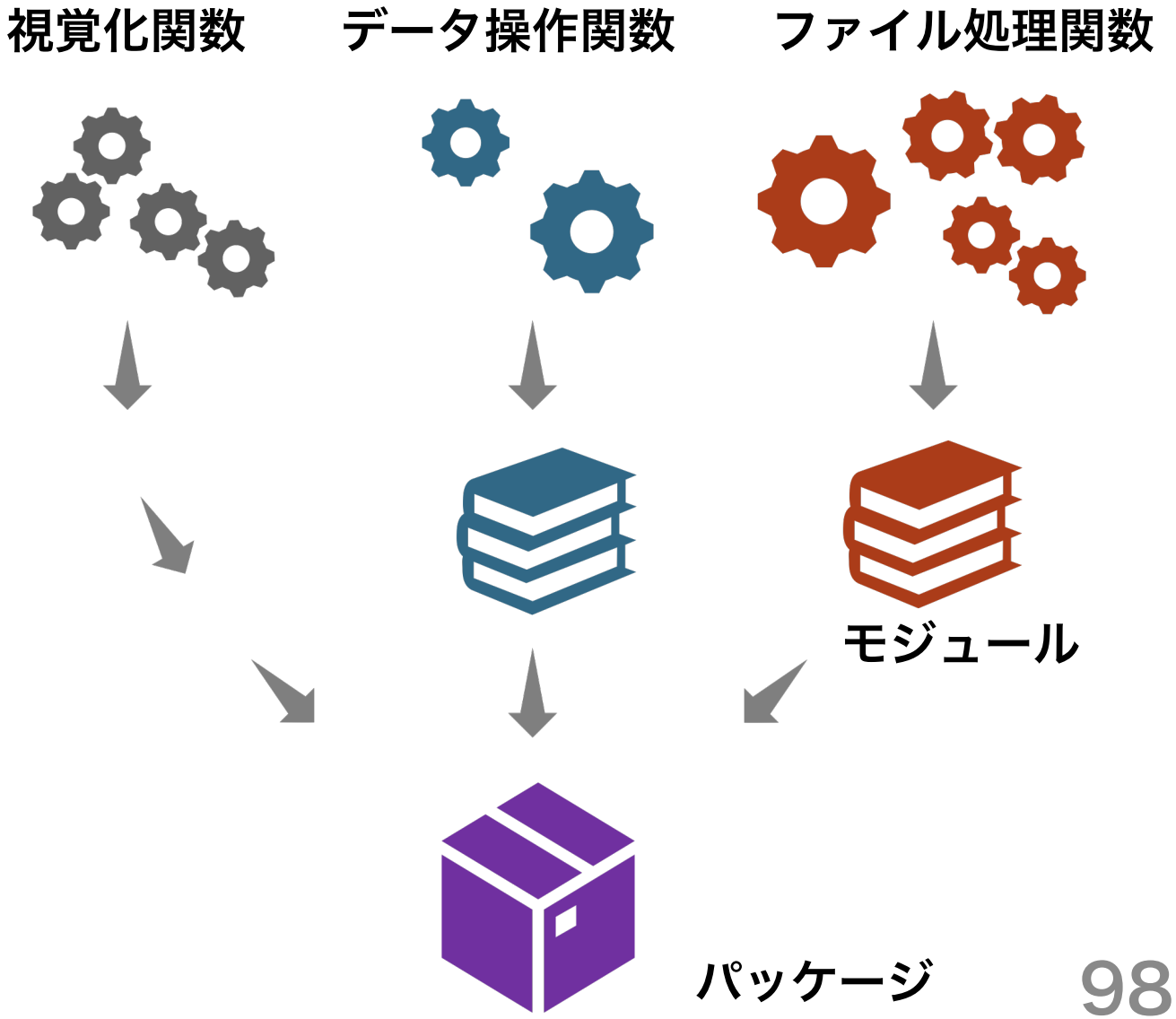
インデントの数で、関数のブロックが終了しているかどうかを判定している。

基本文法

- 予約語
- 条件構文
- 繰り返し構文
- 関数
- パッケージ

パッケージ

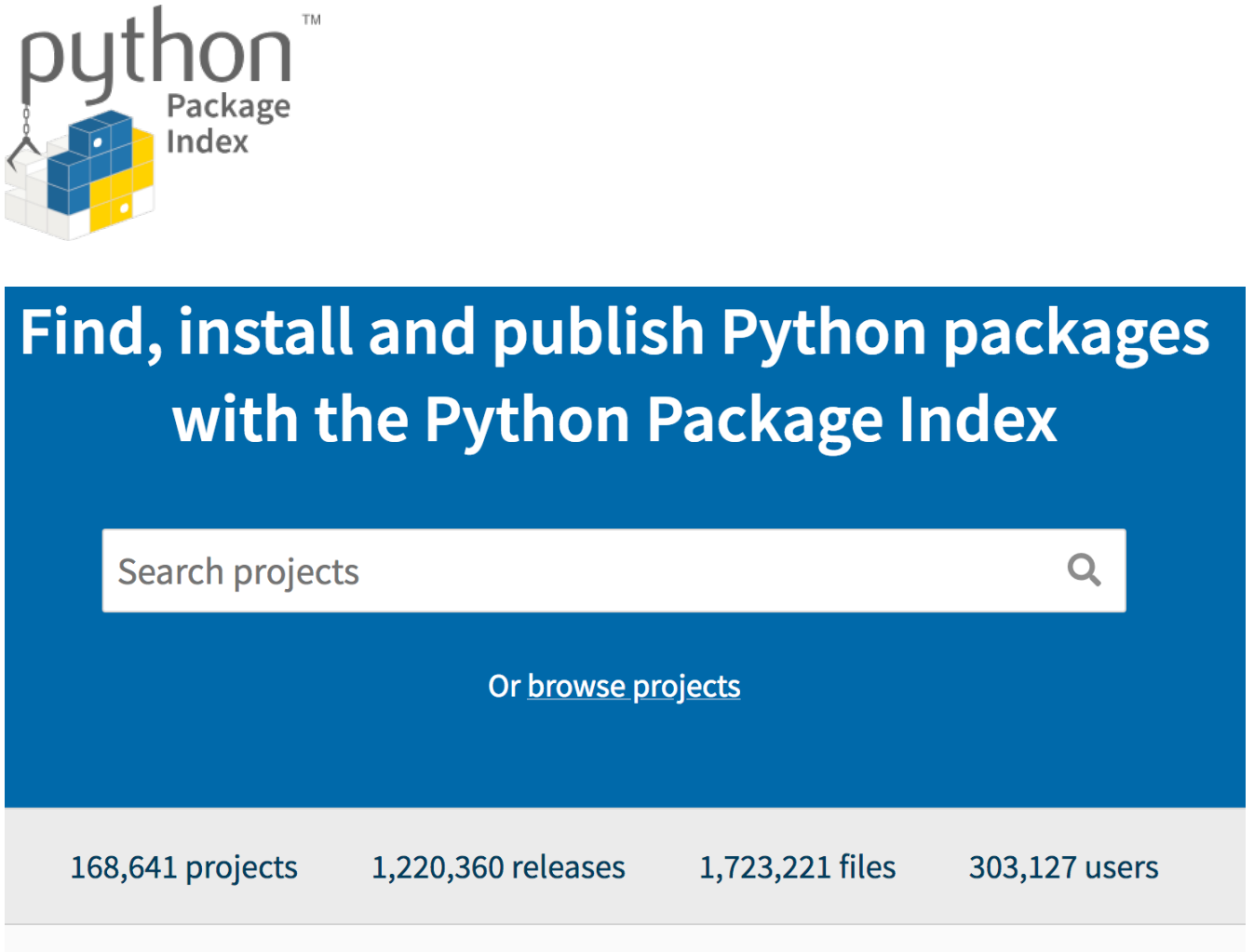
自分で定義している関数の他に、他の開発者が定義した関数を使用することもできる。開発者が開発した複数の関数をひとつ塊にまとめたものをライブラリー（モジュール）という。さらに複数のモジュールをひとつの塊にまとめたものをパッケージという。



パッケージ

自分で定義している関数の他に、他の開発者が定義した関数を使用することもできる。開発者が開発した複数の関数をひとつ塊にまとめたものをライブラリー（モジュール）という。さらに複数のモジュールをひとつの塊にまとめたものをパッケージという。

パッケージは、各開発者・開発団体の公式ウェブサイトあるいは GitHub や PyPI などのレポジトリで配布されている。例えば、PyPI では現在 16.9 万のパッケージが登録されている。個人でも簡単に登録できるので、研究目的の場合は、あまりマイナーなパッケージを使わないようにすること。



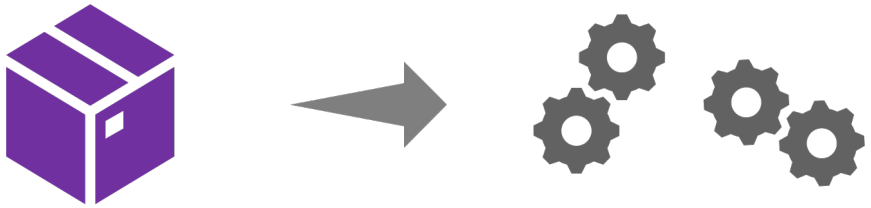
パッケージ

- NumPy
  - ベクトルの高速演算
  - 線形代数
- Pandas
  - データ操作
- SciPy
  - 統計処理
  - 微分方程式・積分
  - パラメーター最適化
- matplotlib
  - 視覚化
- Seaborn
  - 視覚化
- OpenCV
  - 画像解析
  - 動画解析
  - 機械学習
- scikit-image
  - 画像処理
  - 機械学習
- PIL / Pillow
  - 簡易画像処理
- scikit-learn
  - モデリング
  - 統計的機械学習
- tensorflow
  - 深層学習
- PyTorch
  - 深層学習

パッケージ

パッケージを使うとき、import を使ってパッケージの全機能を読み込むことをできる。例えば、データ処理用ライブラリーの NumPy の全機能を読み込むとき、右のようにする。パッケージ中の関数（メソッド）を使うとき、パッケージ名の後に . をつけて、必要な関数を呼び出す（調達する）。例えば、NumPy 中の array 関数を使いたい場合は、右のようにする。

パッケージ名が長くて、読み込み後に略名を与えることができる。この際に、右のように as を使用する。略名を与えた以降、略名を用いて関数などの呼び出しが可能になる。



```
import numpy
a = numpy.array([1, 3, 5])
b = numpy.array([2, 4, 6])



import numpy as np
a = np.array([1, 3, 5])
b = np.array([2, 4, 6])
```

NumPy

- １ 次元配列
- ２ 次元配列
- ３ 次元配列

NumPy

- １ 次元配列
- ２ 次元配列
- ３ 次元配列

1次元配列

NumPy の1次元配列は、Python の1次元リストを拡張したようなオブジェクトである。NumPy 配列を作成するとき、リストを作成し、そのリストを np.array メソッドに代入することで作成する。

```
import numpy as np


a = [1, 1, 2, 3, 5, 8]
a
# [1, 1, 2, 3, 5, 8]

b = np.array(a)
b
# array([1, 1, 2, 3, 5, 8])

c = np.array([1, 1, 2, 3, 5, 8])
c
# array([1, 1, 2, 3, 5, 8])
```

1次元配列

NumPy の1次元配列は、Python の1次元リストを拡張したようなオブジェクトである。NumPy 配列を作成するとき、リストを作成し、そのリストを np.array メソッドに代入することで作成する。すべての要素が 0 または 1 の配列を作成するには、np.zeros または np.ones メソッドを使用すると便利である。 また、np.full メソッドを使用すると、配列のすべての要素を任意の値に初期化することができる。

新しい空の配列を用意するとき、すべての要素を 0 または 1 に初期化するよりも、すべての要素を np.nan （欠損値・非数値）にした方が、バグ等や計算ミス等に気付きやすい。

```
import numpy as np


a = np.zeros(5)
a
# array([0, 0, 0, 0, 0])



b = np.ones(8)
b
# array([1, 1, 1, 1, 1, 1, 1, 1])



c = np.full(5, np.nan)
c
# array([nan, nan, nan, nan, nan])
```

1次元配列

NumPy の1次元配列は、Python の1次元リストを拡張したようなオブジェクトである。NumPy 配列を作成するとき、リストを作成し、そのリストを np.array メソッドに代入することで作成する。すべての要素が 0 または 1 の配列を作成するには、np.zeros または np.ones メソッドを使用すると便利である。また、np.full メソッドを使用すると、配列のすべての要素を任意の値に初期化することができる。 NumPy 配列では、リストで行えなかった演算や統計量計算どを効率よく高速に行うことができる。

NumPy 配列に対して行える演算一覧は次のページで確認できる。
https://docs.scipy.org/doc/numpy/reference/ufuncs.html

```
import numpy as np


a = np.array([1, 5, 10, 50, 100])


np.log10(a)
# array([0.,0.69897,1., 1.69897,2.])


np.sqrt(a)
# array([1.,2.2360,3.1622,7.0710,10.])


np.mean(a)
# 33.2


np.std(a)
# 37.722142038860945
```

1次元配列

NumPy の 1 次元配列は、リストほぼ同じように取り扱うことができる。配列は基本的に全要素を束ねて使うことが多いが、一部の要素だけを取り出したり、変更したりすることができる。その際、配列の先頭から数えて何番目の要素を取り出したいのかを、整数で指定する必要がある。ただし、Python では 0 番から数え始めることに注意。

```
import numpy as np
w = np.array([12, 10, 11, 13, 11])



w[0]


w[1]


w[2]


w[2] = 9


w
```

1次元配列

NumPy の 1 次元配列は、リストほぼ同じように取り扱うことができる。配列は基本的に全要素を束ねて使うことが多いが、一部の要素だけを取り出したり、変更したりすることができる。その際、配列の先頭から数えて何番目の要素を取り出したいのかを、整数で指定する必要がある。ただし、Python では 0 番から数え始めることに注意。

```
import numpy as np
w = np.array([12, 10, 11, 13, 11])



w[0]
# 12
w[1]
# 10
w[2]
# 11
w[2] = 9


w
# array([12, 10, 9, 13, 11])
```

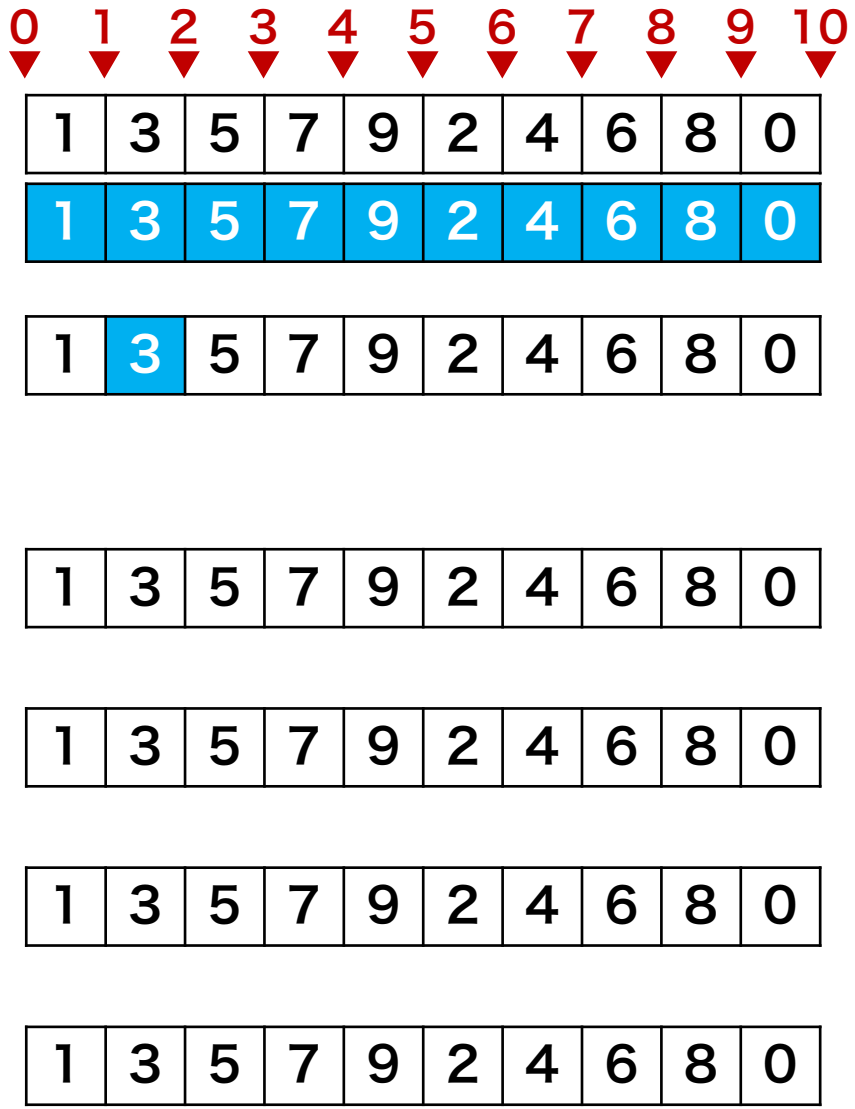
※w の2番目の位置に
9を代入する。

1次元配列（スライス）

NumPy の 1 次元配列は、リストほぼ同じように取り扱うことができる。配列は基本的に全要素を束ねて使うことが多いが、一部の要素だけを取り出したり、変更したりすることができる。その際、配列の先頭から数えて何番目の要素を取り出したいのかを、整数で指定する必要がある。ただし、Python では 0 番から数え始めることに注意。

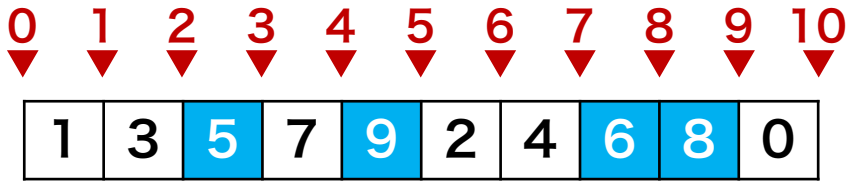
```
a = np.array([1, 3, 5, 7, 9, 2, 4, 6, 8, 0])
a
# array([1, 3, 5, 7, 9, 2, 4, 6, 8, 0])


a[1]
# 3


a[[2, 4, 7, 8]]


a[2:6]


a[:4]


a[5:]
```

1次元配列（スライス）



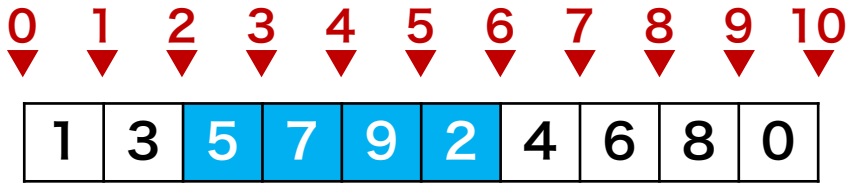

```
a = np.array([1, 3, 5, 7, 9, 2, 4, 6, 8, 0])
a
# array([1, 3, 5, 7, 9, 2, 4, 6, 8, 0])


a[1]
# 3


a[[2, 4, 7, 8]]


a[2:6]


a[:4]


a[5:]
```

1次元配列（スライス）



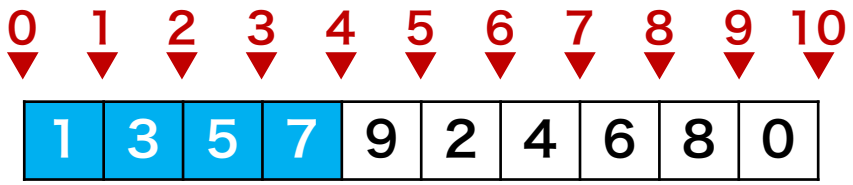

```
a = np.array([1, 3, 5, 7, 9, 2, 4, 6, 8, 0])
a
# array([1, 3, 5, 7, 9, 2, 4, 6, 8, 0])


a[1]
# 3


a[[2, 4, 7, 8]]
# array([5, 9, 6, 8])
a[2:6]


a[:4]


a[5:]
```

1次元配列（スライス）



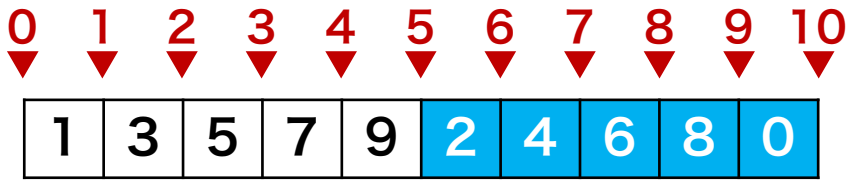

```
a = np.array([1, 3, 5, 7, 9, 2, 4, 6, 8, 0])
a
# array([1, 3, 5, 7, 9, 2, 4, 6, 8, 0])

a[1]
# 3

a[[2, 4, 7, 8]]
# array([5, 9, 6, 8])
a[2:6]
# array([5, 7, 9, 2])
a[:4]

a[5:]
```

1次元配列（スライス）



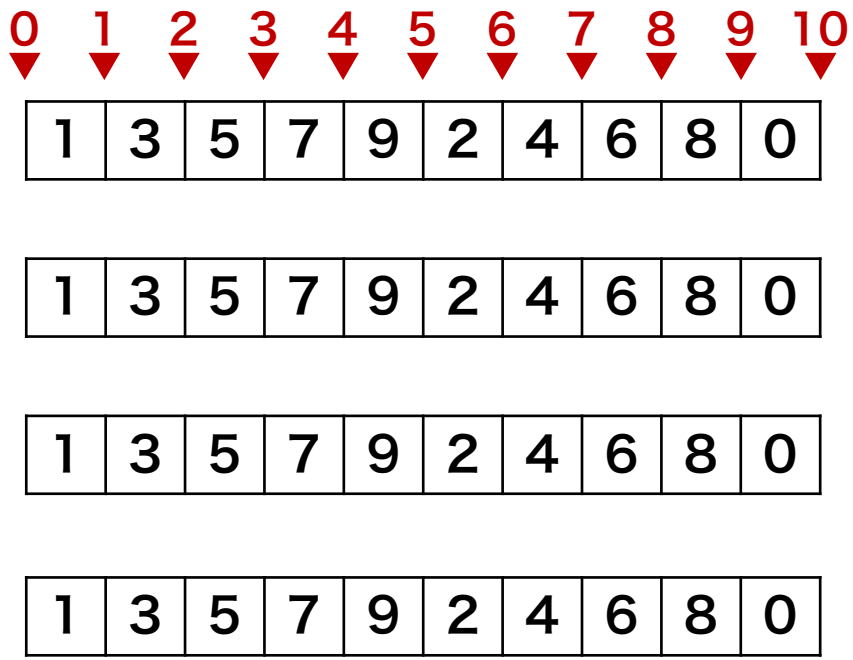

```
a = np.array([1, 3, 5, 7, 9, 2, 4, 6, 8, 0])
a
# array([1, 3, 5, 7, 9, 2, 4, 6, 8, 0])


a[1]
# 3


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

1次元配列（スライス）




```
a = np.array([1, 3, 5, 7, 9, 2, 4, 6, 8, 0])
a
# array([1, 3, 5, 7, 9, 2, 4, 6, 8, 0])


a[1]
# 3


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

確認問題




```
import numpy as np

a = np.array([1, 3, 5, 7, 9, 2, 4, 6, 8, 0])
```

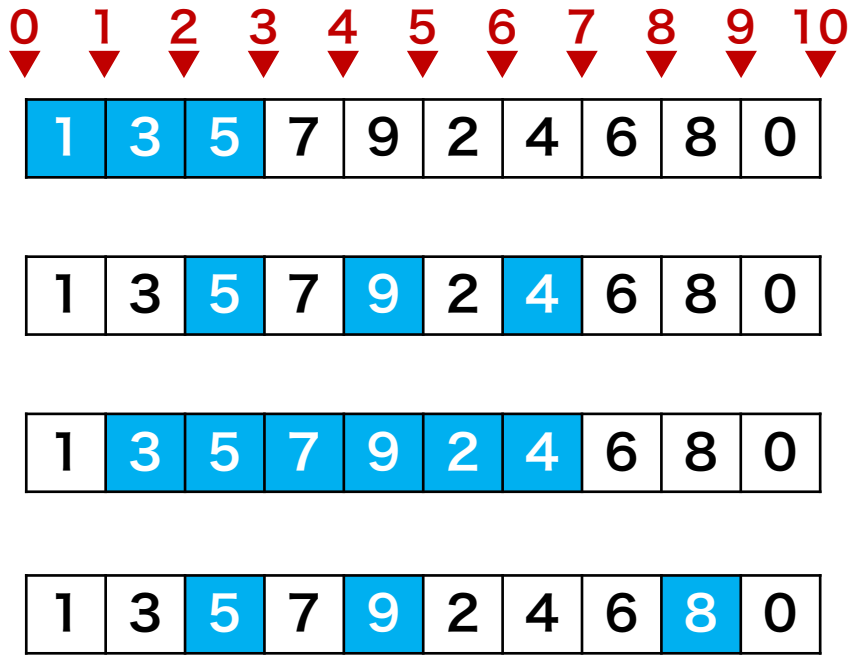
```
a[:3]
```

```
a[[2, 4, 6]]
```

```
a[1:7]
```

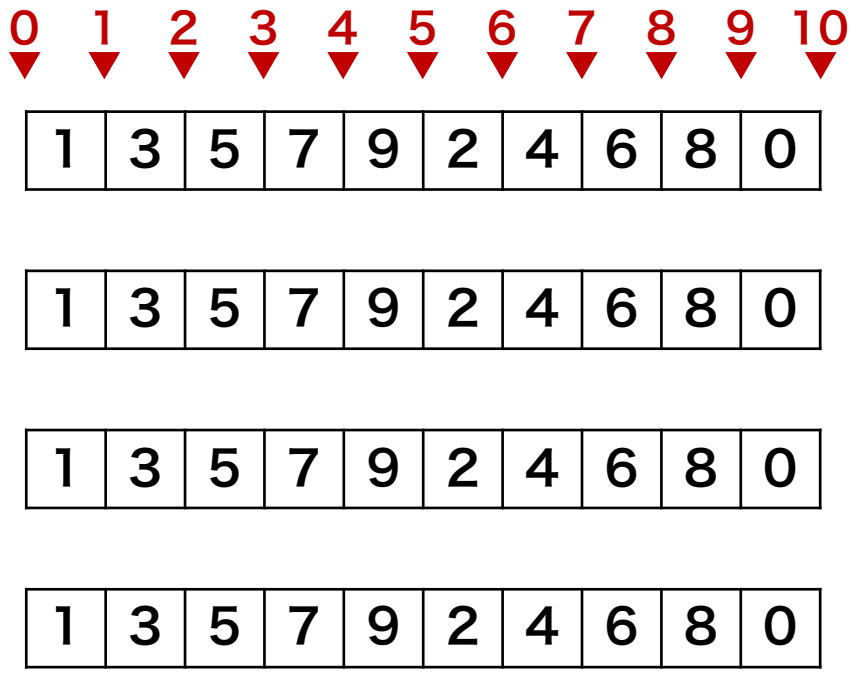
```
a[[8, 4, 2]]
```

確認問題 解答



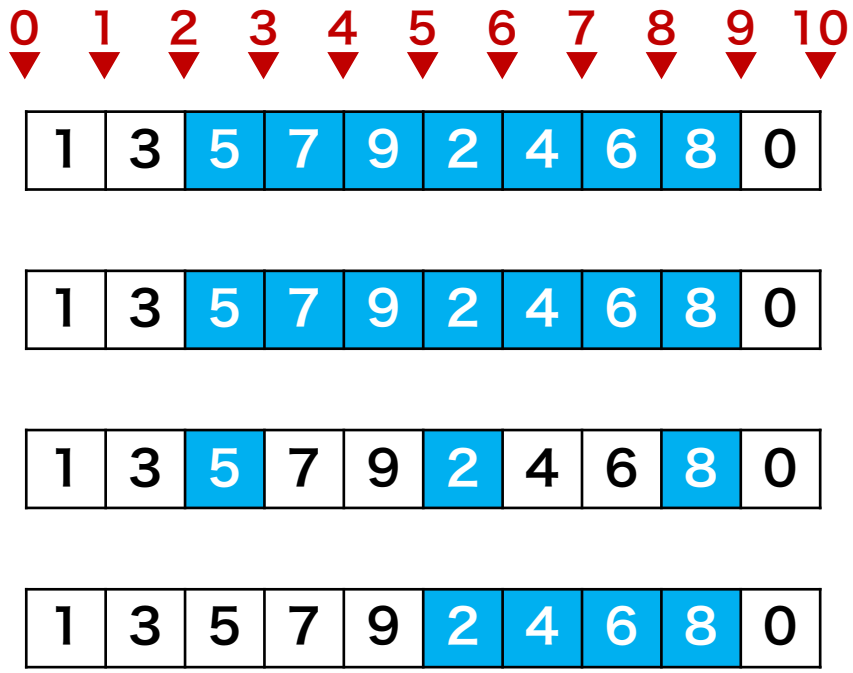

```
import numpy as np

a = np.array([1, 3, 5, 7, 9, 2, 4, 6, 8, 0])



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

1次元配列（スライス）




```
import numpy as np

a = np.array([1, 3, 5, 7, 9, 2, 4, 6, 8, 0])


a[2:9]

a[2:9:1]

a[2:9:3]

a[9:4:-1]
```

1次元配列（スライス）




```
import numpy as np

a = np.array([1, 3, 5, 7, 9, 2, 4, 6, 8, 0])



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

1次元配列

配列から要素を取り出すときに位置番号を指定する方法の他に、True または False からなる配列で指定することもできる。このとき、True/False からなる配列の長さは、操作対象となる配列の長さと同じでなければならない。

```
import numpy as np


a = np.array([   2,    4,     6,    8])
k = np.array([True, True, False, True])


a[k]




a = np.array([   2,    4,     6,    8])
k = np.array([False, True, False, True])


a[k]
```

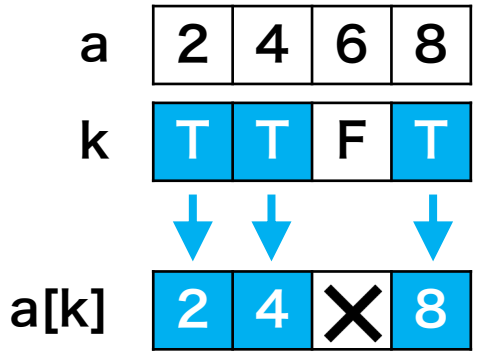
1次元配列

a	2	4	6	8
k	T	T	F	T

```
import numpy as np

a = np.array([   2,    4,     6,    8])
k = np.array([True, True, False, True])

a[k]



a = np.array([   2,    4,     6,    8])
k = np.array([False, True, False, True])

a[k]
```

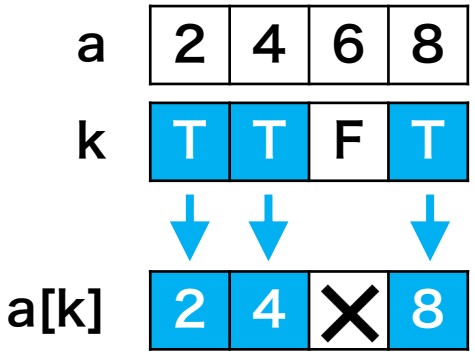
1次元配列

フィルター k を用いて、ベクトル a の要素をフィルタリングしているイメージ。

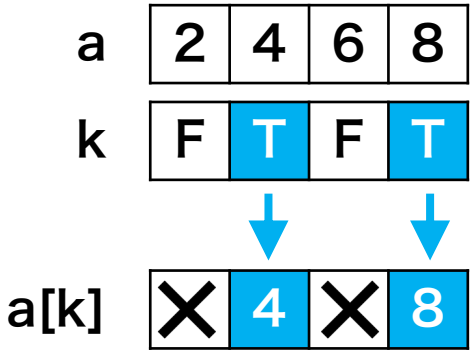



```
import numpy as np

a = np.array([   2,    4,     6,    8])
k = np.array([True, True, False, True])

a[k]
# array([2, 4, 8])



a = np.array([   2,    4,     6,    8])
k = np.array([False, True, False, True])

a[k]
```

1次元配列




```
import numpy as np

a = np.array([   2,    4,     6,    8])
k = np.array([True, True, False, True])

a[k]
# array([2, 4, 8])
```



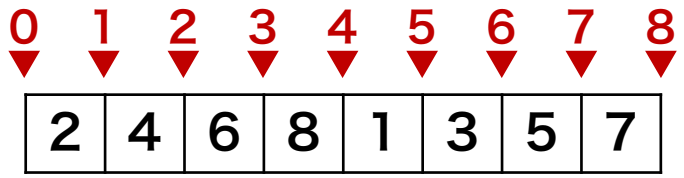

```
a = np.array([   2,    4,     6,    8])
k = np.array([False, True, False, True])

a[k]
# array([4, 8])
```

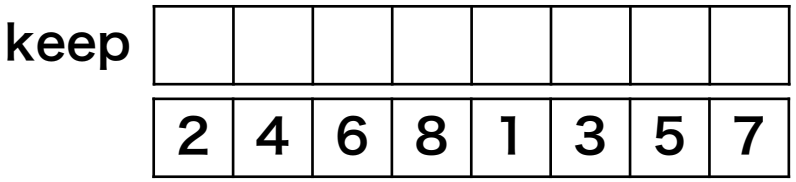
1次元配列

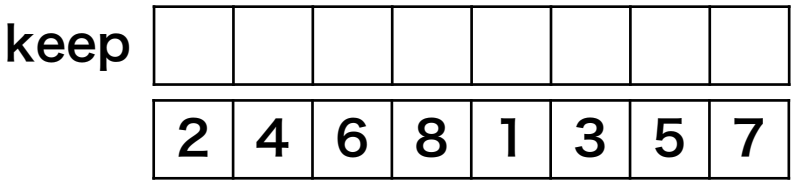
True または False からなる配列は、条件判定によって作ることもできる。

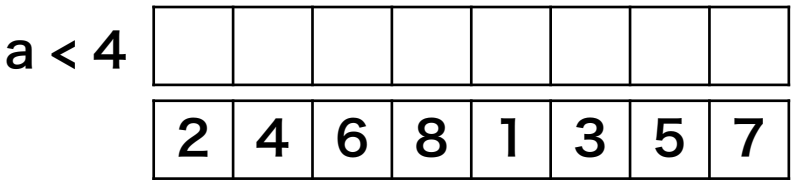
```
import numpy as np
a = np.array([2, 4, 6, 8, 1, 3, 5, 7])


keep = (a > 5)


a[keep]



keep = (a % 2 == 1)


a[keep]



a[(a < 4)]
```

1次元配列



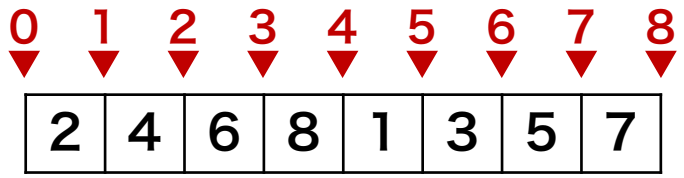


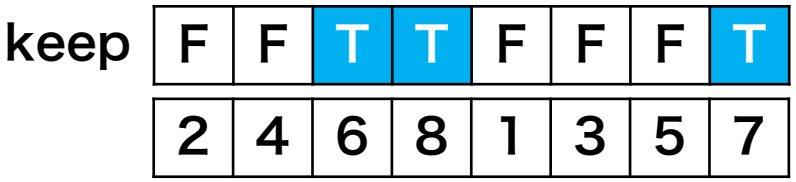


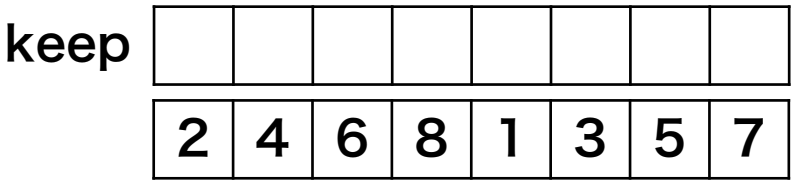


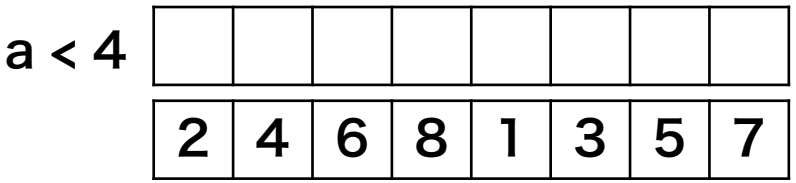

```
import numpy as np
a = np.array([2, 4, 6, 8, 1, 3, 5, 7])

keep = (a > 5)

a[keep]



keep = (a % 2 == 1)

a[keep]



a[(a < 4)]
```

1次元配列



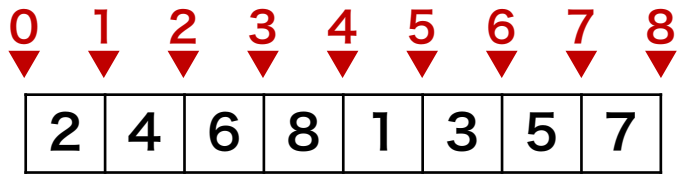


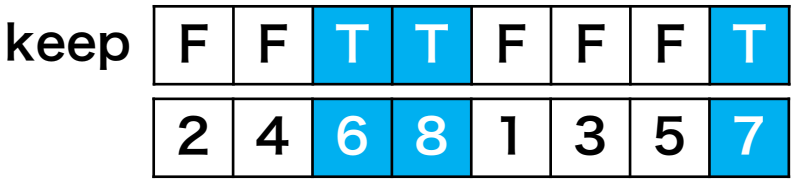


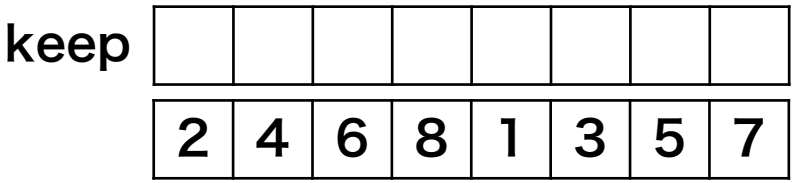


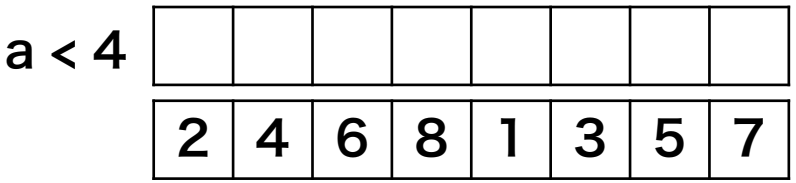

```
import numpy as np
a = np.array([2, 4, 6, 8, 1, 3, 5, 7])

keep = (a > 5)
# array([F, F, T, T, F, F, F, T])
a[keep]



keep = (a % 2 == 1)

a[keep]



a[(a < 4)]
```

1次元配列



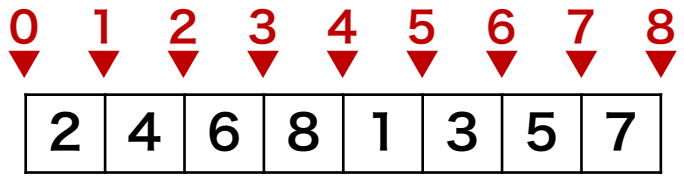


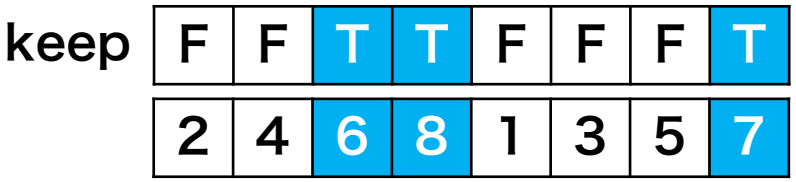


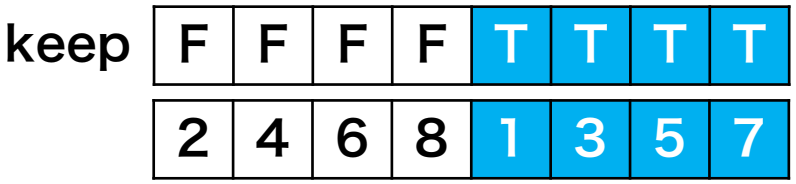


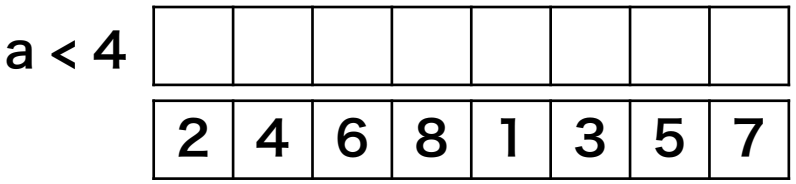

```
import numpy as np
a = np.array([2, 4, 6, 8, 1, 3, 5, 7])

keep = (a > 5)
# array([F, F, T, T, F, F, F, T])
a[keep]
# array([6, 8, 7])

keep = (a % 2 == 1)

a[keep]



a[(a < 4)]
```

1次元配列



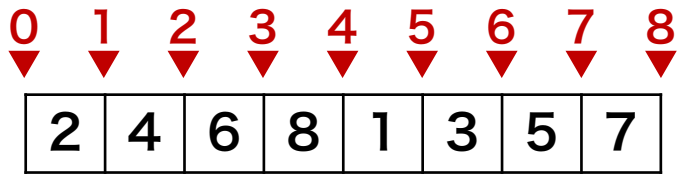


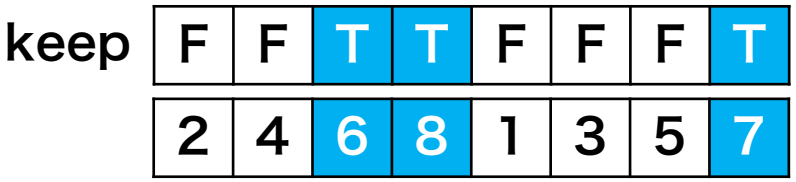


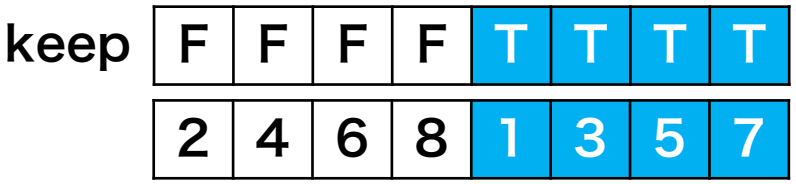


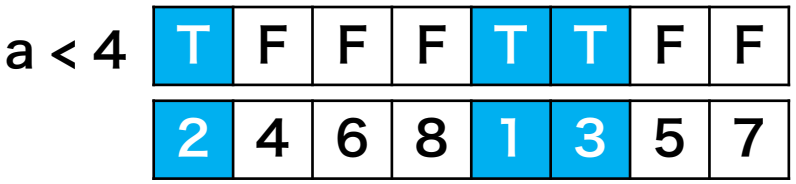

```
import numpy as np
a = np.array([2, 4, 6, 8, 1, 3, 5, 7])

keep = (a > 5)
# array([F, F, T, T, F, F, F, T])
a[keep]
# array([6, 8, 7])

keep = (a % 2 == 1)
# array([F, F, F, F, T, T, T, T])
a[keep]
# array([1, 3, 5, 7])

a[(a < 4)]
```

1次元配列



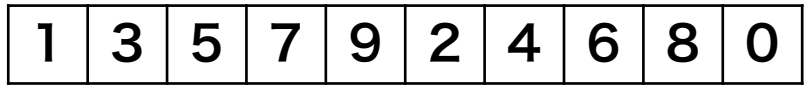




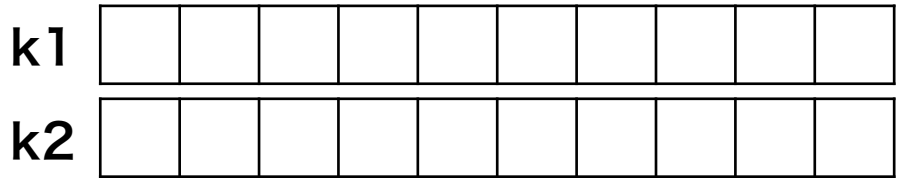



```
import numpy as np
a = np.array([2, 4, 6, 8, 1, 3, 5, 7])

keep = (a > 5)
# array([F, F, T, T, F, F, F, T])
a[keep]
# array([6, 8, 7])

keep = (a % 2 == 1)
# array([F, F, F, F, T, T, T, T])
a[keep]
# array([1, 3, 5, 7])

a[(a < 4)]
# array([2, 1, 3])
```

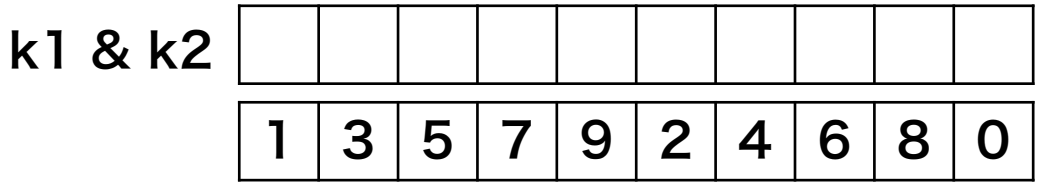
1次元配列



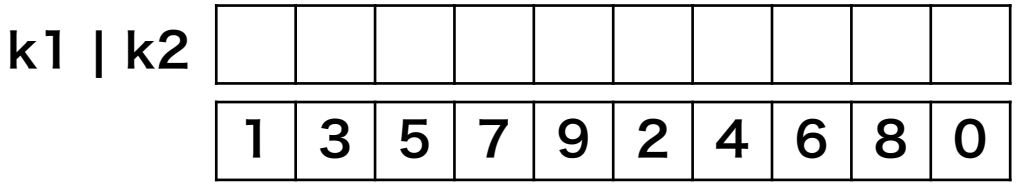

```
import numpy as np
a = np.array([1, 3, 5, 7, 9, 2, 4, 6, 8, 0])
```




```
k1 = (a % 2 == 1)
k2 = (a > 5)
```

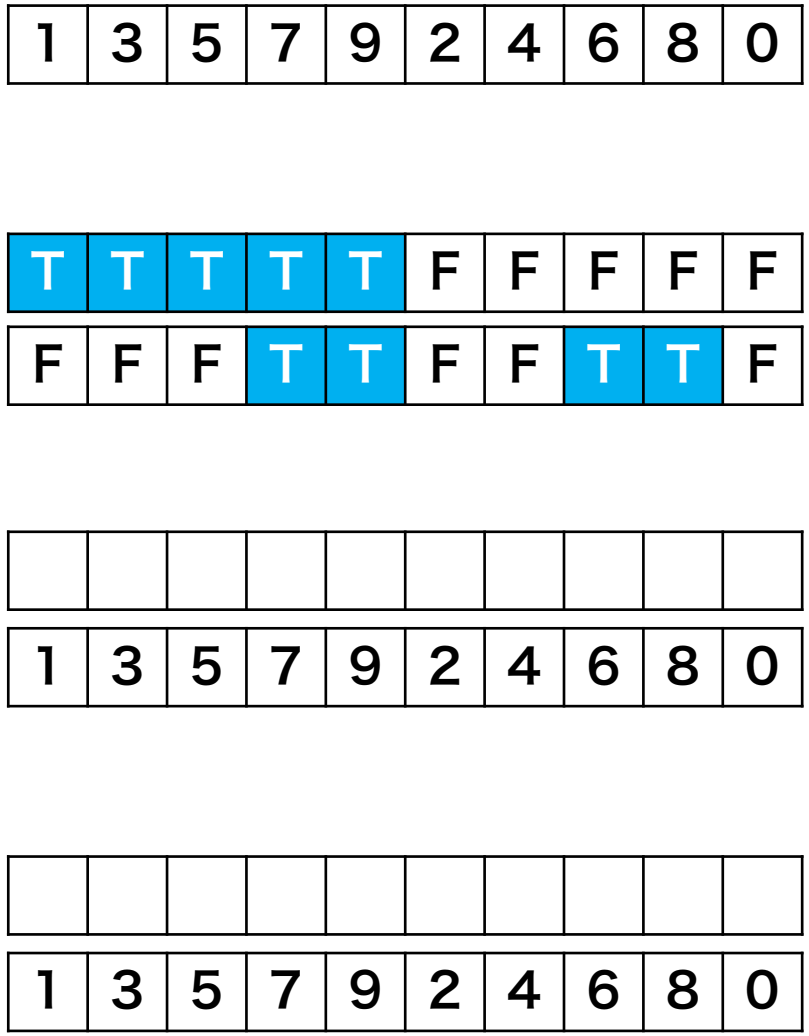



```
a[k1 & k2]
```



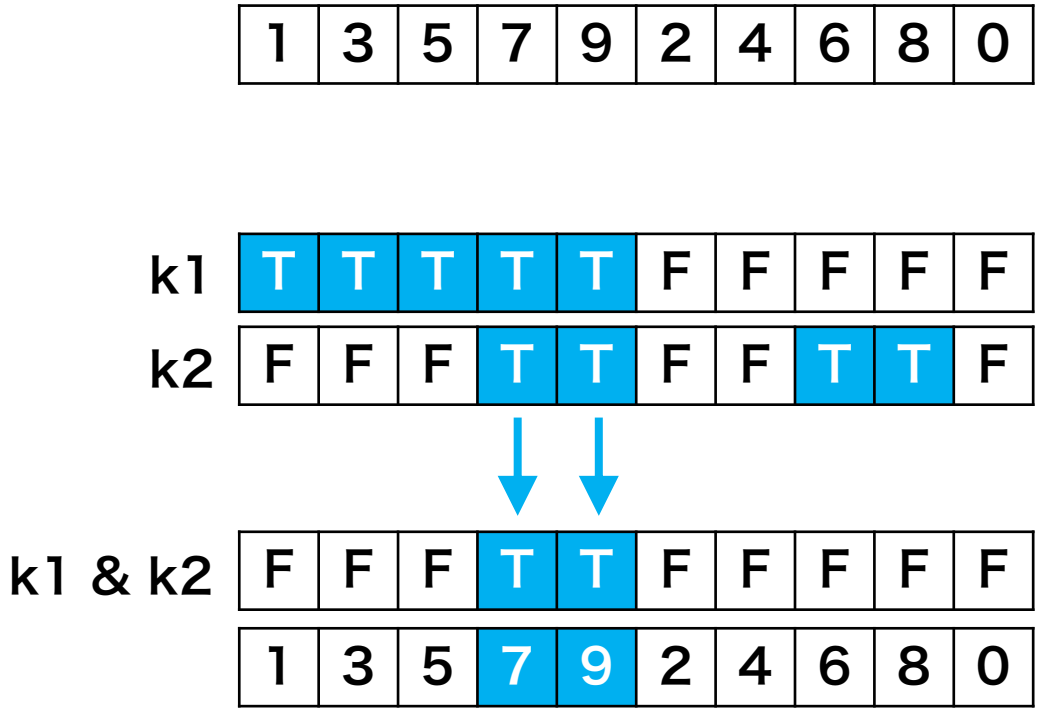

```
a[k1 | k2]
```

1次元配列



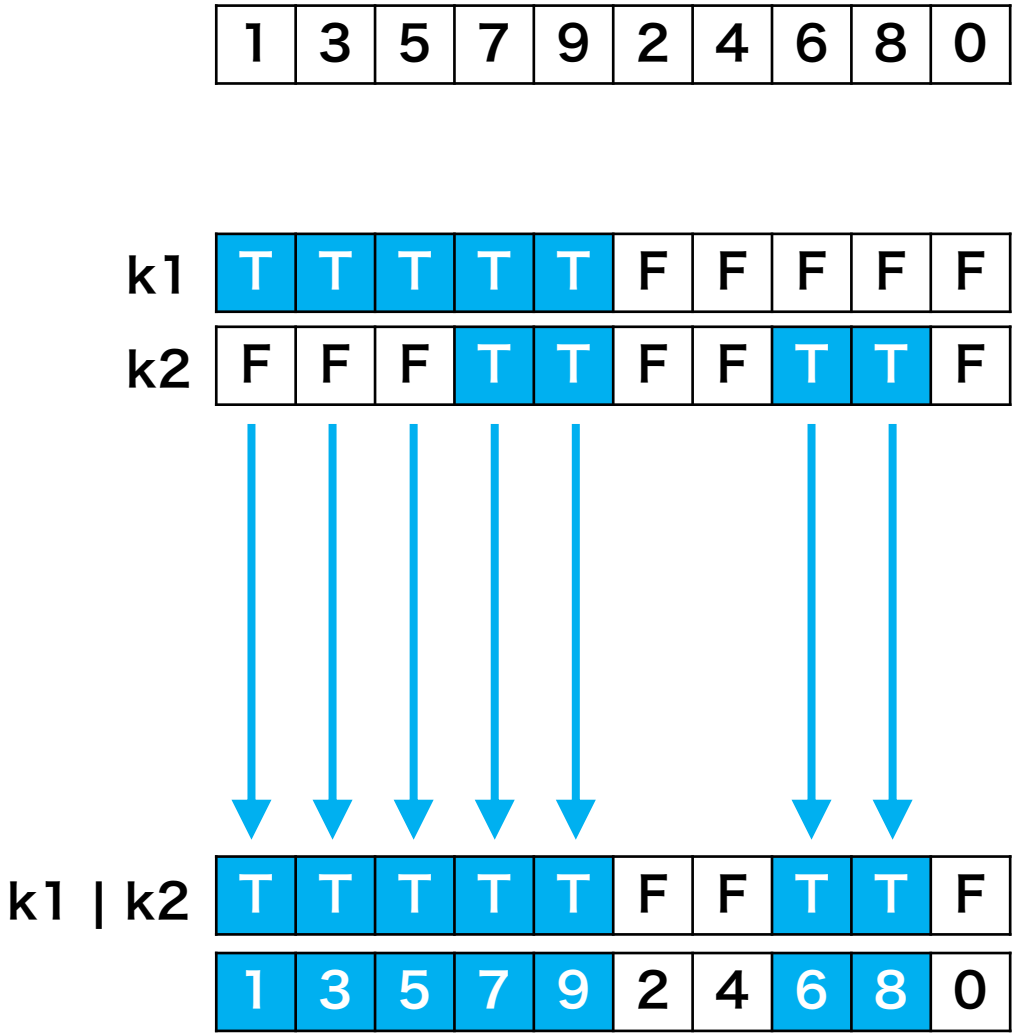

```
import numpy as np
a = np.array([1, 3, 5, 7, 9, 2, 4, 6, 8, 0])


k1 = (a % 2 == 1)
k2 = (a > 5)


a[k1 & k2]


a[k1 | k2]
```

1次元配列




```
import numpy as np
a = np.array([1, 3, 5, 7, 9, 2, 4, 6, 8, 0])


k1 = (a % 2 == 1)
k2 = (a > 5)



a[k1 & k2]
# array([7, 9])

a[k1 | k2]
```

1次元配列



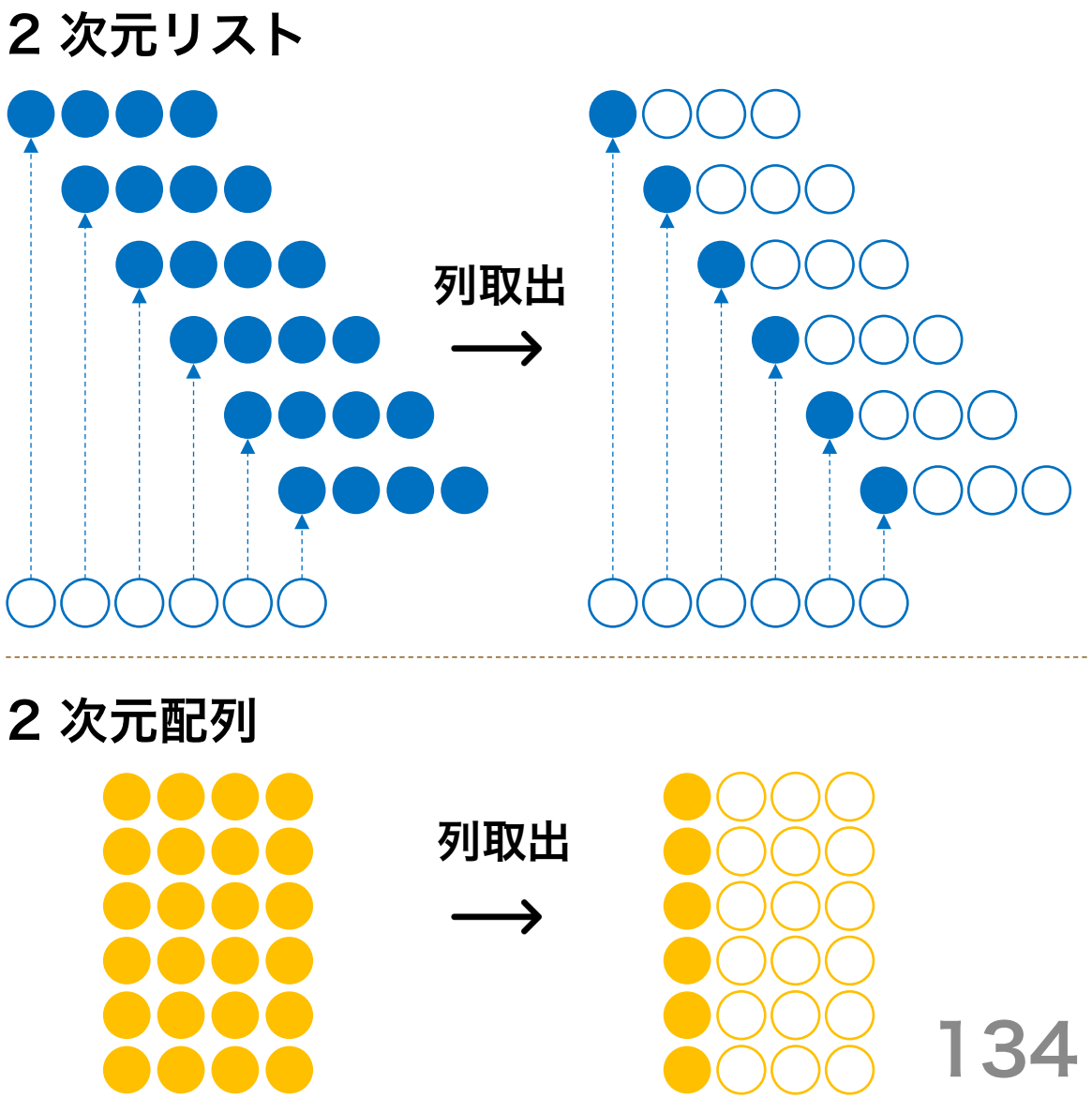

```
import numpy as np
a = np.array([1, 3, 5, 7, 9, 2, 4, 6, 8, 0])


k1 = (a % 2 == 1)
k2 = (a > 5)



a[k1 & k2]


a[k1 | k2]
# array([1, 3, 5, 7, 9, 6, 8])
```

NumPy

- １ 次元配列
- ２ 次元配列
- ３ 次元配列

2次元配列

２次元リストがあるように、NumPy にも２次元の配列が定義できる。２次元リストは、リストの中にリストが保存されている構造である。これに対して、２次元配列は、縦と横を持つ構造を保っている。そのため、２次元配列は数学の行列のように扱える。



2次元配列

２次元配列を作成するには、2次元のリストを作成して、それを np.array 関数に代入することで２次元配列が生成される。

```
a = np.array([[11, 12, 13, 14, 15, 16],
              [21, 22, 23, 24, 25, 26],
              [31, 32, 33, 34, 35, 36],
              [41, 42, 43, 44, 45, 46],
              [51, 52, 53, 54, 55, 56],
              [61, 62, 63, 64, 65, 66]])
```

11	12	13	14	15	16
21	22	23	24	25	26
31	32	33	34	35	36
41	42	43	44	45	46
51	52	53	54	55	56
61	62	63	64	65	66

2次元配列

２次元配列を作成するには、2次元のリストを作成して、それを np.array 関数に代入することで２次元配列が生成される。また、１次元配列と同様に、np.zeros または np.ones を使用することで、すべての要素が 0 または 1 からなる２次元配列を作成することができる。

```
b = np.zeros((2, 5))
```

0	0	0	0	0
0	0	0	0	0

```
c = np.ones((5, 3))
```

1	1	1
1	1	1
1	1	1
1	1	1
1	1	1

2次元配列

２次元配列を作成するには、2次元のリストを作成して、それを np.array 関数に代入することで２次元配列が生成される。また、１次元配列と同様に、np.zeros または np.ones を使用することで、すべての要素が 0 または 1 からなる２次元配列を作成することができる。さらに２次元配列の場合は、np.identity 関数を使用すると単位行列を作成することができる。

```
d = np.identity(5)
```

1	0	0	0	0	0
0	1	0	0	0	0
0	0	1	0	0	0
0	0	0	1	0	0
0	0	0	0	1	0
0	0	0	0	0	1

2次元配列

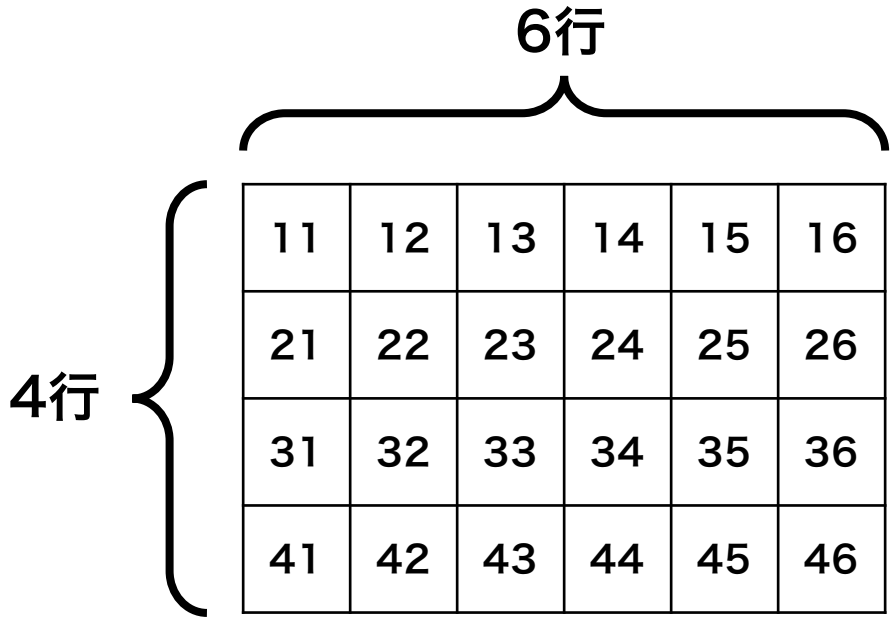
２次元配列を作成するには、2次元のリストを作成して、それを np.array 関数に代入することで２次元配列が生成される。また、１次元配列と同様に、np.zeros または np.ones を使用することで、すべての要素が 0 または 1 からなる２次元配列を作成することができる。さらに２次元配列の場合は、np.identity 関数を使用すると単位行列を作成することができる。すべての要素が np.ana となる２次元配列も作成できる。

```
e = np.full((5, 3), np.nan)
```

nan	nan	nan
nan	nan	nan
nan	nan	nan
nan	nan	nan
nan	nan	nan

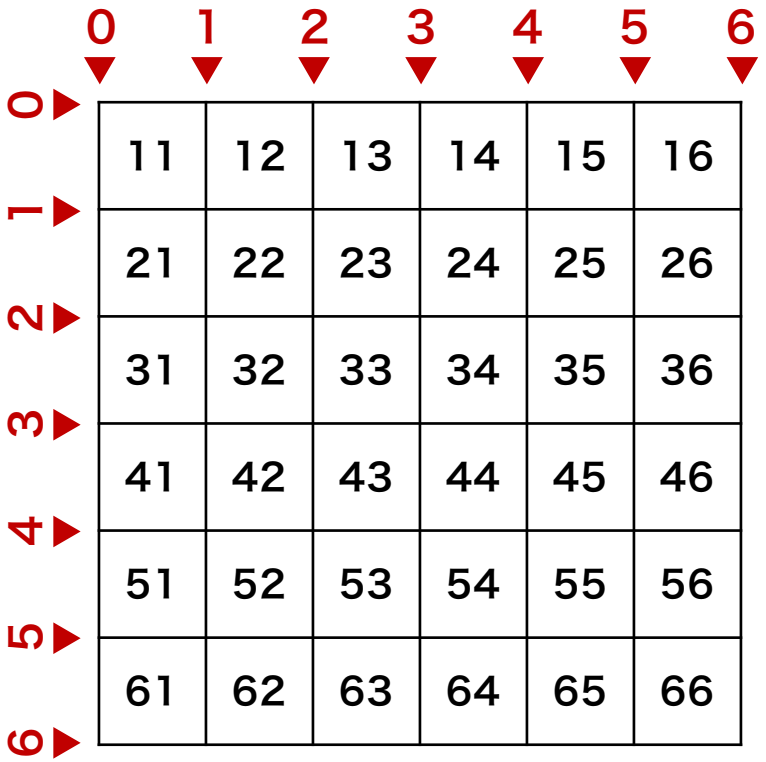
2次元配列

２次元行列の行数と列数を調べるとき、shape 属性を取得するか、np.size 関数を使用する。np.size 関数を使用するとき、axis 引数を指定する必要があり、axis=0 は行（1次元方向）、axis=1 は列（2次元方向）を表す。



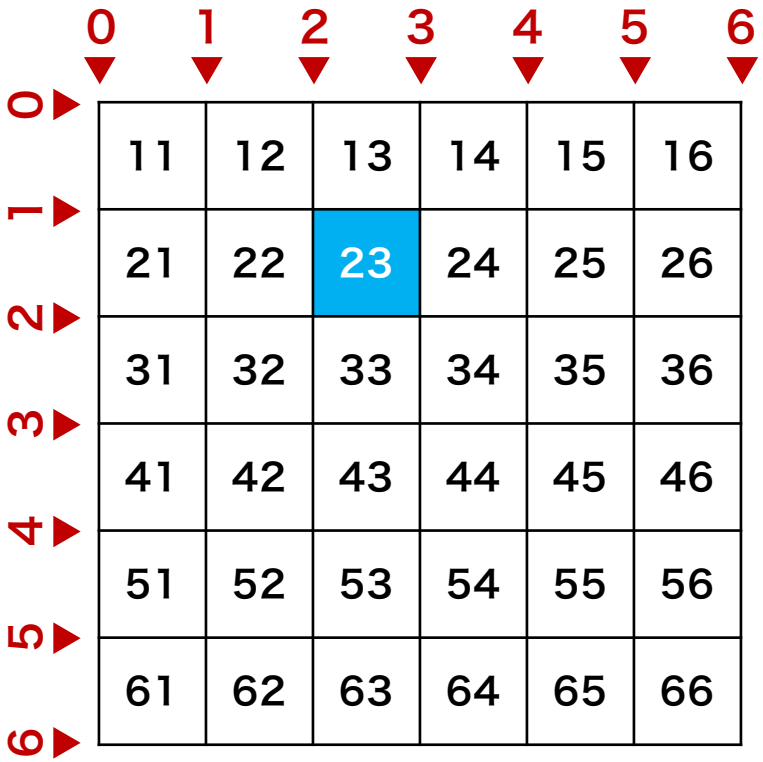

```
a = np.array([[11, 12, 13, 14, 15, 16],
              [21, 22, 23, 24, 25, 26],
              [31, 32, 33, 34, 35, 36],
              [41, 42, 43, 44, 45, 46]])



a.shape
# (4, 6)

np.size(a, axis=0)
# 4

np.size(a, axis=1)
# 6
```

2次元配列



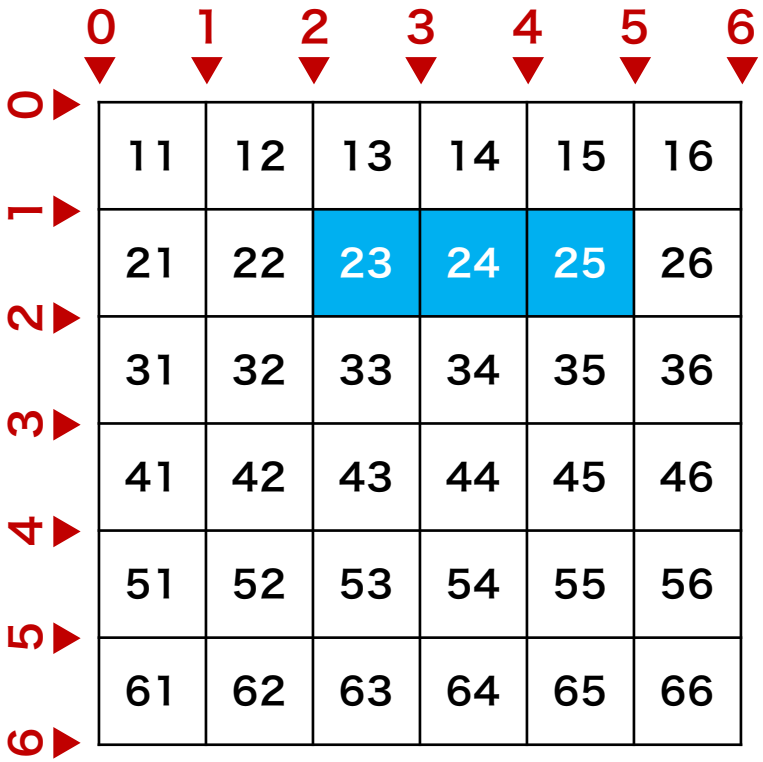

```
b = np.array([[11, 12, 13, 14, 15, 16],
              [21, 22, 23, 24, 25, 26],
              [31, 32, 33, 34, 35, 36],
              [41, 42, 43, 44, 45, 46],
              [51, 52, 53, 54, 55, 56],
              [61, 62, 63, 64, 65, 66]])
b[1, 2]

b[1, 2:5]

b[1:5, 3]

b[2:4, 1:6]
```

2次元配列



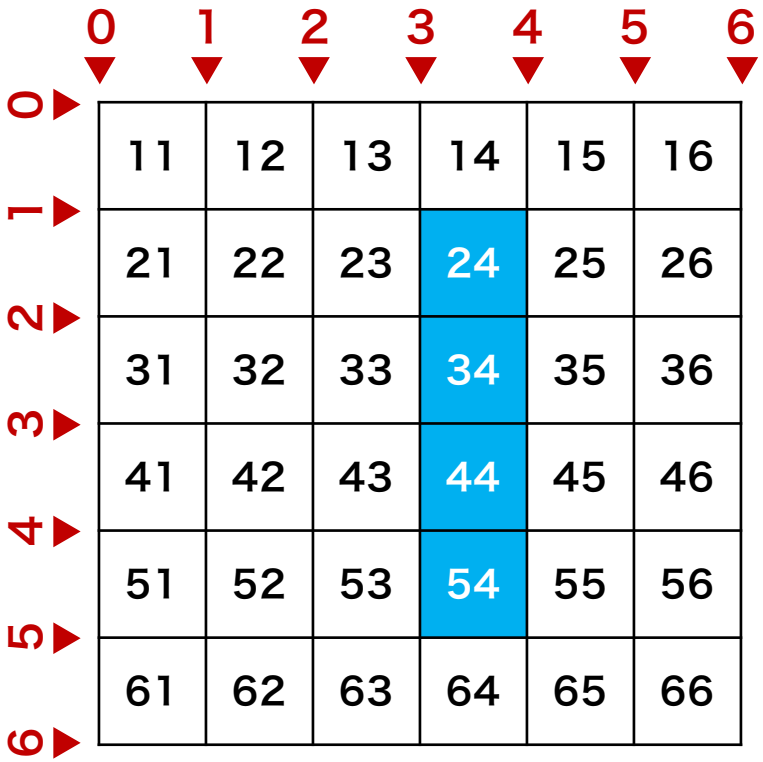

```
b = np.array([[11, 12, 13, 14, 15, 16],
              [21, 22, 23, 24, 25, 26],
              [31, 32, 33, 34, 35, 36],
              [41, 42, 43, 44, 45, 46],
              [51, 52, 53, 54, 55, 56],
              [61, 62, 63, 64, 65, 66]])
b[1, 2]
# 23
b[1, 2:5]

b[1:5, 3]

b[2:4, 1:6]
```

2次元配列



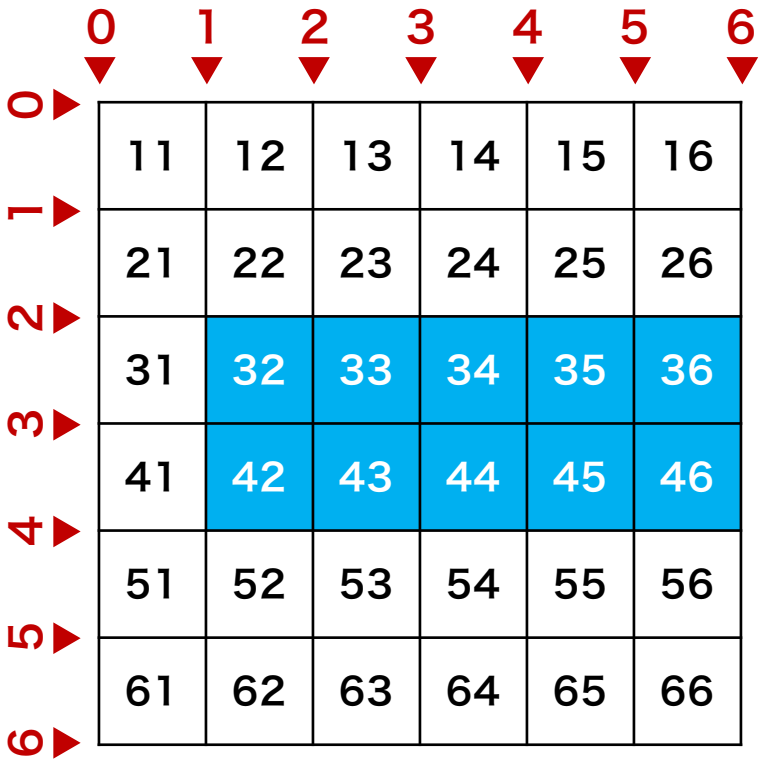

```
b = np.array([[11, 12, 13, 14, 15, 16],
              [21, 22, 23, 24, 25, 26],
              [31, 32, 33, 34, 35, 36],
              [41, 42, 43, 44, 45, 46],
              [51, 52, 53, 54, 55, 56],
              [61, 62, 63, 64, 65, 66]])
b[1, 2]
# 23
b[1, 2:5]
# array([23, 24, 25])
b[1:5, 3]


b[2:4, 1:6]
```

2次元配列



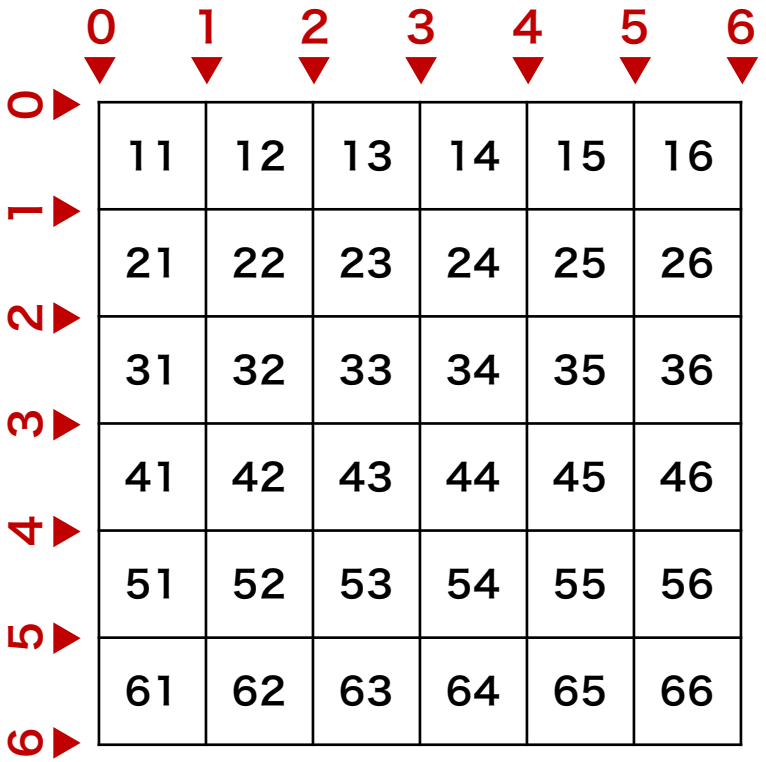

```
b = np.array([[11, 12, 13, 14, 15, 16],
              [21, 22, 23, 24, 25, 26],
              [31, 32, 33, 34, 35, 36],
              [41, 42, 43, 44, 45, 46],
              [51, 52, 53, 54, 55, 56],
              [61, 62, 63, 64, 65, 66]])
b[1, 2]
# 23
b[1, 2:5]
# array([23, 24, 25])
b[1:5, 3]
# array([24, 34, 44, 54])
b[2:4, 1:6]
```

2次元配列



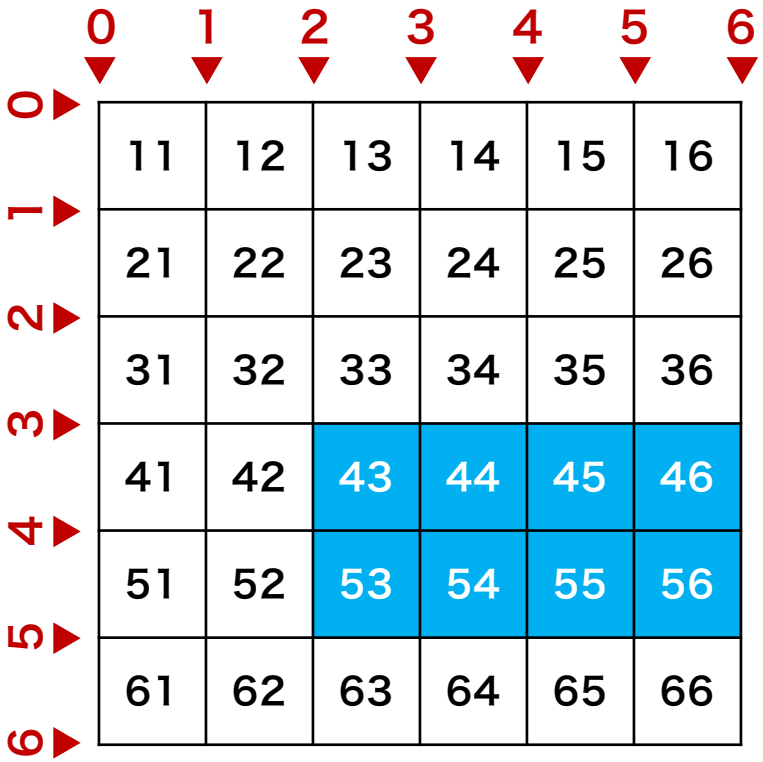

```
b = np.array([[11, 12, 13, 14, 15, 16],
              [21, 22, 23, 24, 25, 26],
              [31, 32, 33, 34, 35, 36],
              [41, 42, 43, 44, 45, 46],
              [51, 52, 53, 54, 55, 56],
              [61, 62, 63, 64, 65, 66]])
b[1, 2]
# 23
b[1, 2:5]
# array([23, 24, 25])
b[1:5, 3]
# array([24, 34, 44, 54])
b[2:4, 1:6]
# array([[32, 33, 34, 35, 36],
#        [42, 43, 44, 45, 46]])
```

2次元配列



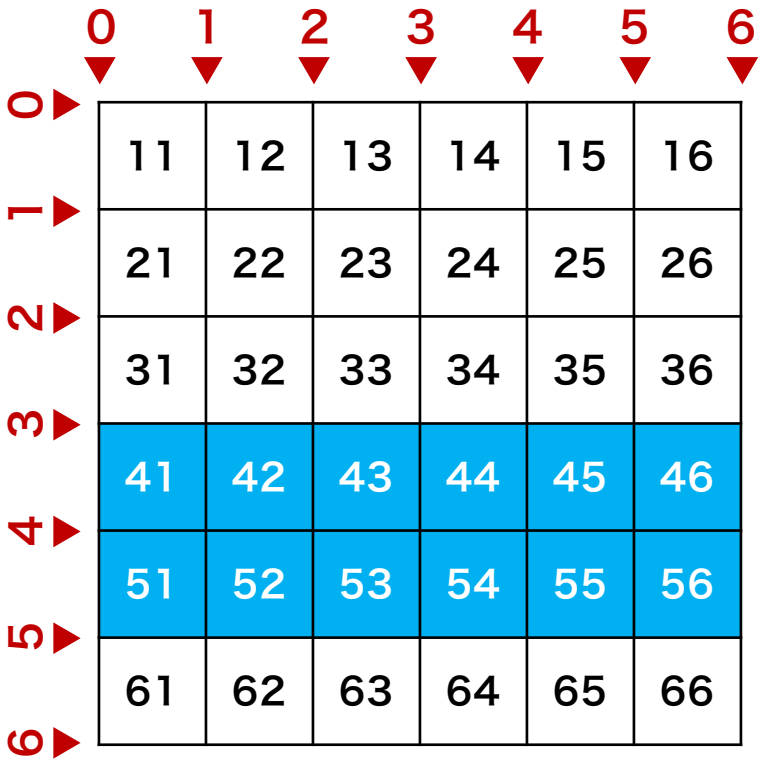

```
b = np.array([[11, 12, 13, 14, 15, 16],
              [21, 22, 23, 24, 25, 26],
              [31, 32, 33, 34, 35, 36],
              [41, 42, 43, 44, 45, 46],
              [51, 52, 53, 54, 55, 56],
              [61, 62, 63, 64, 65, 66]])
b[3:5, 2:]


b[3:5, :]


b[:, 2]


b[:, 1:3]
```

2次元配列



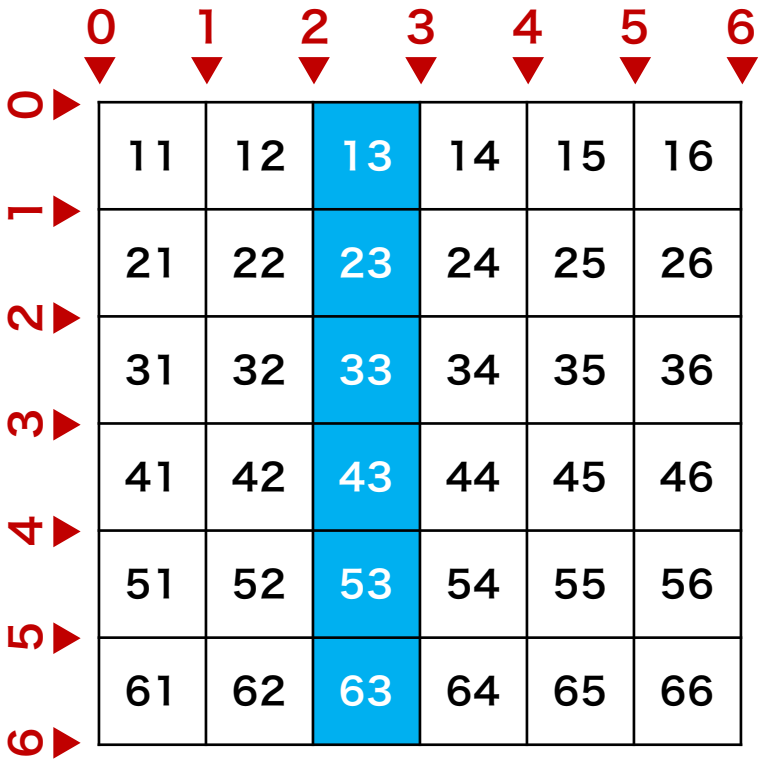

```
b = np.array([[11, 12, 13, 14, 15, 16],
              [21, 22, 23, 24, 25, 26],
              [31, 32, 33, 34, 35, 36],
              [41, 42, 43, 44, 45, 46],
              [51, 52, 53, 54, 55, 56],
              [61, 62, 63, 64, 65, 66]])
b[3:5, 2:]
# array([[43, 44, 45, 46],
#        [53, 54, 55, 56]])
b[3:5, :]


b[:, 2]


b[:, 1:3]
```

2次元配列



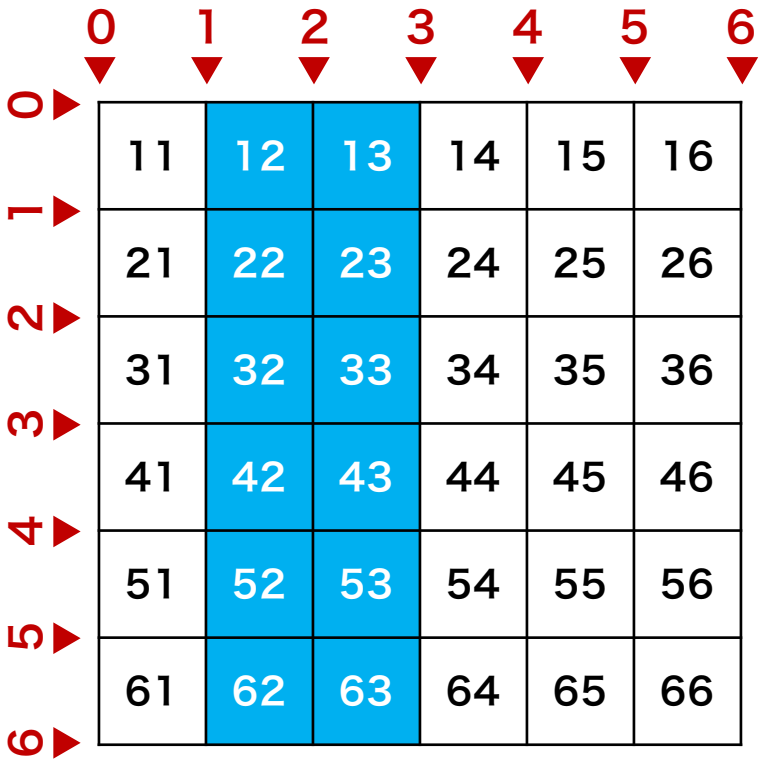

```
b = np.array([[11, 12, 13, 14, 15, 16],
              [21, 22, 23, 24, 25, 26],
              [31, 32, 33, 34, 35, 36],
              [41, 42, 43, 44, 45, 46],
              [51, 52, 53, 54, 55, 56],
              [61, 62, 63, 64, 65, 66]])
b[3:5, 2:]
# array([[43, 44, 45, 46],
#        [53, 54, 55, 56]])
b[3:5, :]
# array([[41, 42, 43, 44, 45, 46],
#        [51, 52, 53, 54, 55, 56]])
b[:, 2]



b[:, 1:3]
```

2次元配列



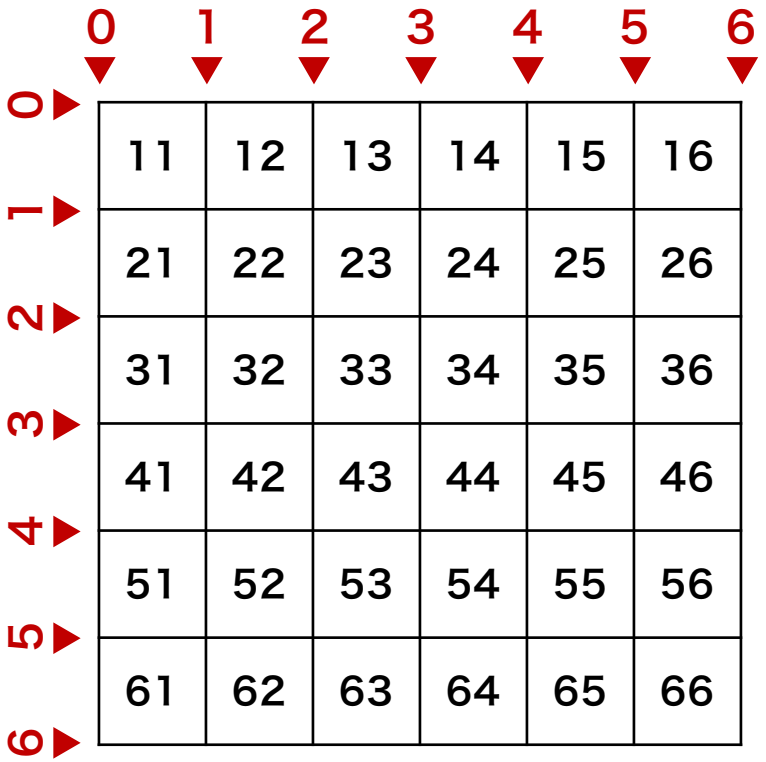

```
b = np.array([[11, 12, 13, 14, 15, 16],
              [21, 22, 23, 24, 25, 26],
              [31, 32, 33, 34, 35, 36],
              [41, 42, 43, 44, 45, 46],
              [51, 52, 53, 54, 55, 56],
              [61, 62, 63, 64, 65, 66]])
b[3:5, 2:]
# array([[43, 44, 45, 46],
#        [53, 54, 55, 56]])
b[3:5, :]
# array([[41, 42, 43, 44, 45, 46],
#        [51, 52, 53, 54, 55, 56]])
b[:, 2]
# array([13, 23, 33, 43, 53, 63])

b[:, 1:3]
```

2次元配列



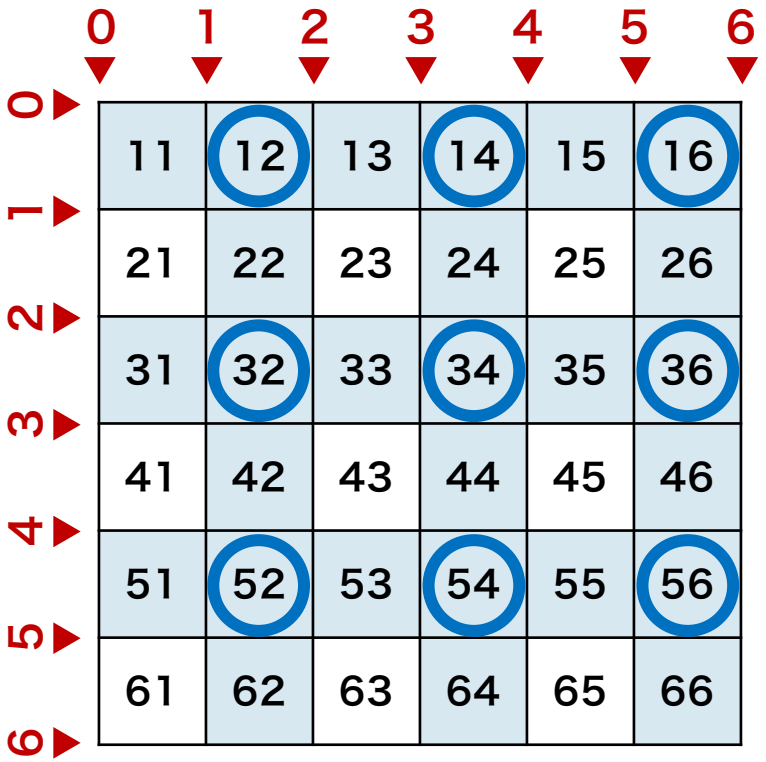

```
b = np.array([[11, 12, 13, 14, 15, 16],
              [21, 22, 23, 24, 25, 26],
              [31, 32, 33, 34, 35, 36],
              [41, 42, 43, 44, 45, 46],
              [51, 52, 53, 54, 55, 56],
              [61, 62, 63, 64, 65, 66]])
b[:, 1:3]
# array([[12, 13], [22, 23], [32, 33],
#        [42, 43], [52, 53], [62, 63]])
```

2次元配列（注意点）



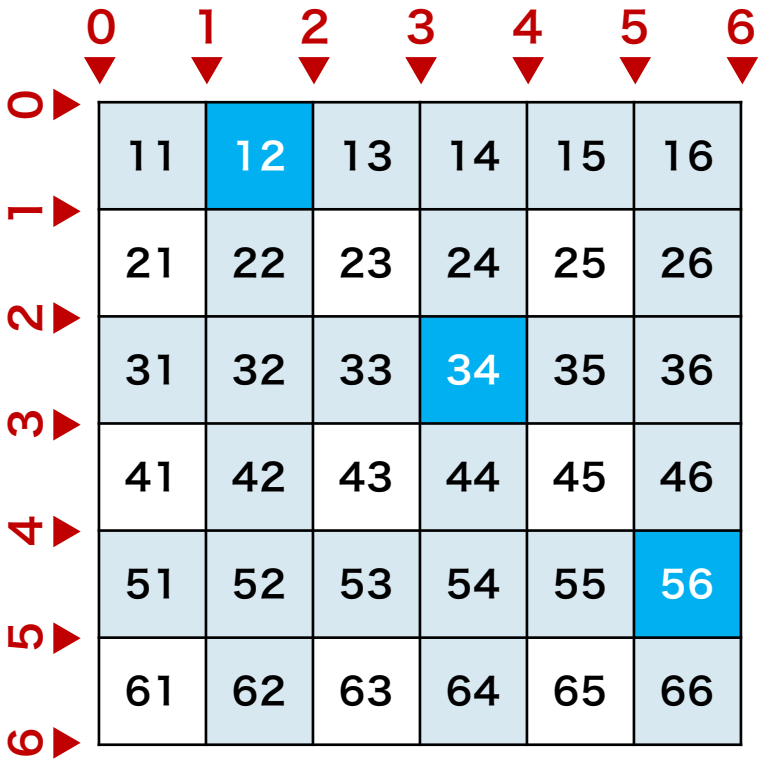

```
b = np.array([[11, 12, 13, 14, 15, 16],
              [21, 22, 23, 24, 25, 26],
              [31, 32, 33, 34, 35, 36],
              [41, 42, 43, 44, 45, 46],
              [51, 52, 53, 54, 55, 56],
              [61, 62, 63, 64, 65, 66]])
r = [0, 2, 4]
c = [1, 3, 5]
b[r, c]
```

2次元配列（注意点）



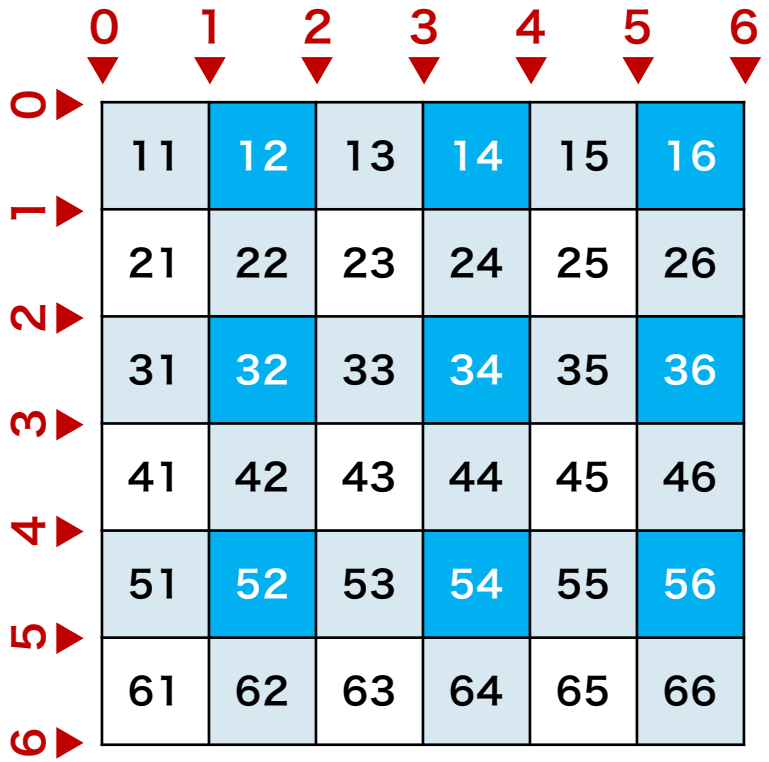

```
b = np.array([[11, 12, 13, 14, 15, 16],
              [21, 22, 23, 24, 25, 26],
              [31, 32, 33, 34, 35, 36],
              [41, 42, 43, 44, 45, 46],
              [51, 52, 53, 54, 55, 56],
              [61, 62, 63, 64, 65, 66]])
r = [0, 2, 4]
c = [1, 3, 5]
b[r, c]
```

2次元配列（注意点）




```
b = np.array([[11, 12, 13, 14, 15, 16],
              [21, 22, 23, 24, 25, 26],
              [31, 32, 33, 34, 35, 36],
              [41, 42, 43, 44, 45, 46],
              [51, 52, 53, 54, 55, 56],
              [61, 62, 63, 64, 65, 66]])
r = [0, 2, 4]
c = [1, 3, 5]
b[r, c]
# array([12, 34, 56])
```

2次元配列（注意点）




```
b = np.array([[11, 12, 13, 14, 15, 16],
              [21, 22, 23, 24, 25, 26],
              [31, 32, 33, 34, 35, 36],
              [41, 42, 43, 44, 45, 46],
              [51, 52, 53, 54, 55, 56],
              [61, 62, 63, 64, 65, 66]])
r = [0, 2, 4]
c = [1, 3, 5]
b[r, c]
# array([12, 34, 56])

b[np.ix_(r, c)]
# array([[12, 14, 16],
#        [32, 34, 36],
#        [52, 54, 56]])
```

行列計算

NumPy の 2 次元配列同士では、行列のように演算を行うことができる。配列の各要素同士の加減乗除の他に、NumPy のメソッドを利用することで、内積や外積も簡単に求めることができる。

計算式	計算内容
a + b	各要素の足し算
a - b	各要素の引き算
a * b	各要素の掛け算
a / b	各要素の割り算
np.dot(a, b)	行列同士の内積
np.outer(a, b)	行列同士の外積（テンソル積）
np.sum(a)	全要素の和
a.T	行列 a の転置行列

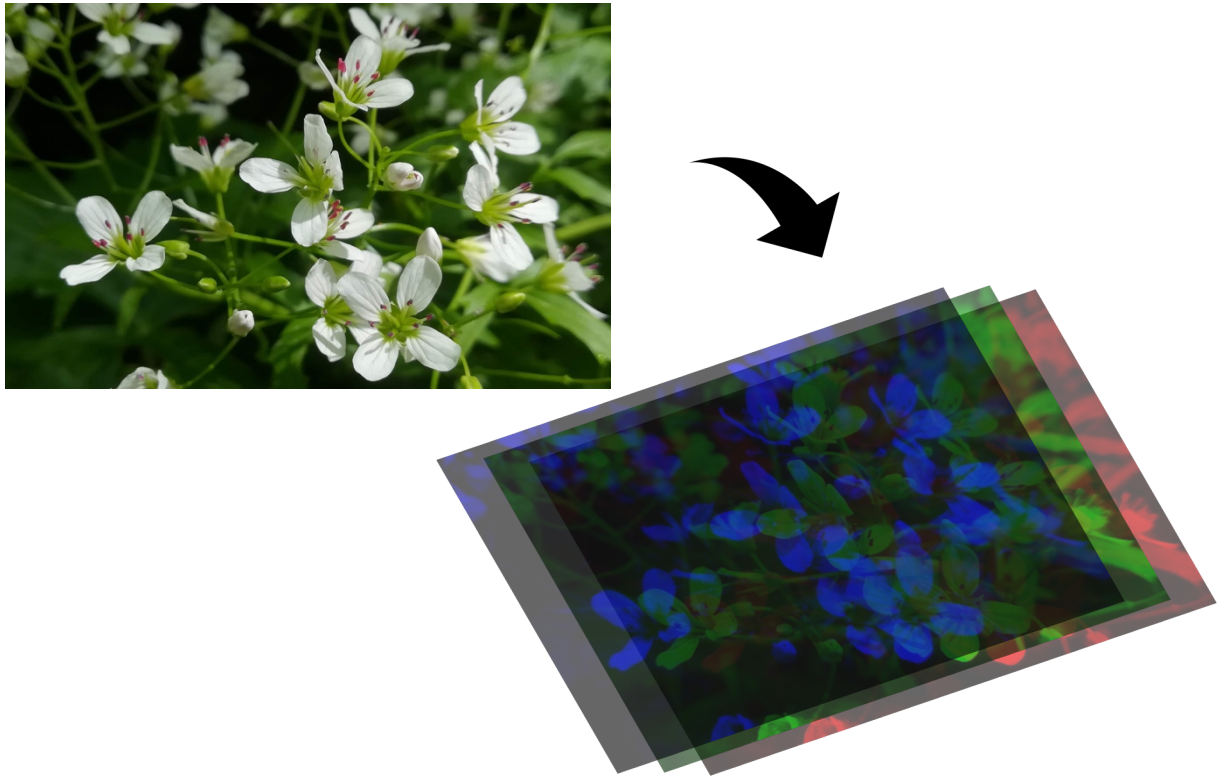
```
import numpy as np
a = np.array([[1, 2, 3],
              [4, 5, 6],
              [7, 8, 9]])
b = np.array([[1, 1, 1],
              [0, 1, 1],
              [0, 0, 1]])
a + b
# array([[ 2, 3, 4],
#        [ 4, 6, 7],
#        [ 7, 8, 10]])

np.dot(a, b)
array([[ 1,  3,  6],
       [ 4,  9, 15],
       [ 7, 15, 24]])
```

NumPy

- １ 次元配列
- ２ 次元配列
- ３ 次元配列

3次元配列

三次元配列は、画像解析のときに使われる。デジタルカメラで撮られている画像の色は、赤・緑・青の 3 原色によって表される。そのため、縦 n 横 m の画像は、n x m の配列がまずあり、その各 (n, m) 要素には RGB の三つの要素が含まれているようになる。ただし、OpenCV で読み込まれた画像の色の並び順は BGR となっているので、注意すること。



```
b = np.array([
    [[1, 0, 1], [1, 1, 0], [1, 1, 1]],
    [[1, 0, 1], [1, 1, 0], [1, 1, 1]],
    [[1, 0, 1], [1, 1, 0], [1, 1, 1]],
    [[1, 0, 1], [1, 1, 0], [1, 1, 1]],
    [[1, 0, 1], [1, 1, 0], [1, 1, 1]],
])


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

Pandas

- シリーズ
- データフレーム
- 表データ処理

python™

アプリ開発やシステム管理など様々な用途で利用できるような汎用機能を提供する。

- リスト
- 2 次元リスト

Pandas

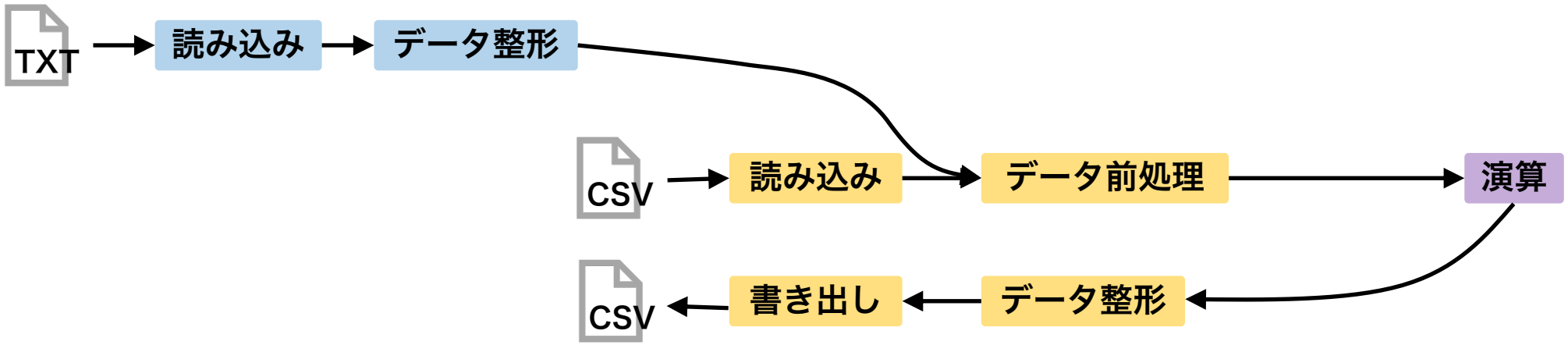
CSV ファイルを読み書きや表データの操作や整形などに特化した機能を提供する。

- シリーズ
- データフレーム

NumPy

整形されたデータに対して、情報量をできるだけ落とさずに、高速に演算を行う機能を提供する。

- 配列
- 2 次元配列



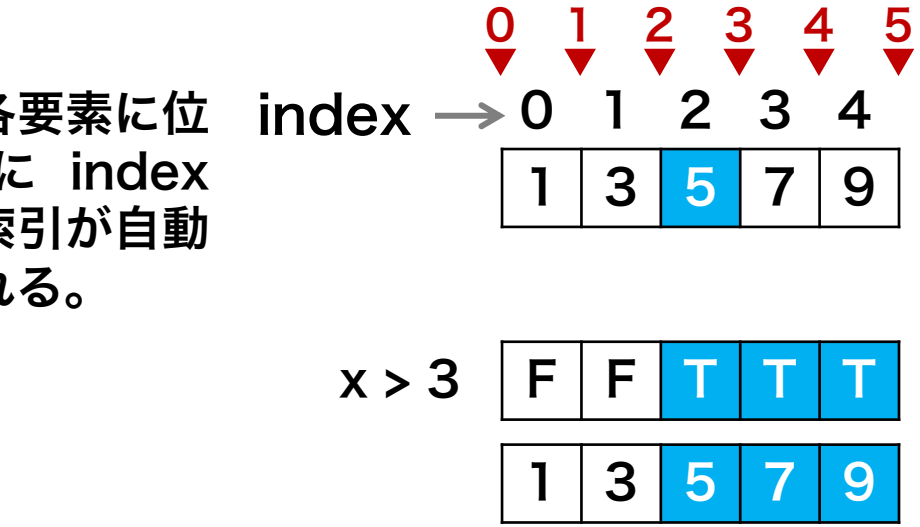
Pandas

- シリーズ
- データフレーム
- 表データ処理

シリーズ

1 次元リストあるいは 1 次元配列にあたるデータ構造を、Pandas ではシリーズという。シリーズは Series メソッド（関数）で作成する。シリーズから各要素の値を取り出すときは、NumPy と同様に、位置番号あるいは条件を指定して取り出す。

シリーズの各要素に位置番号の他に index と呼ばれる索引が自動的に付けられる。

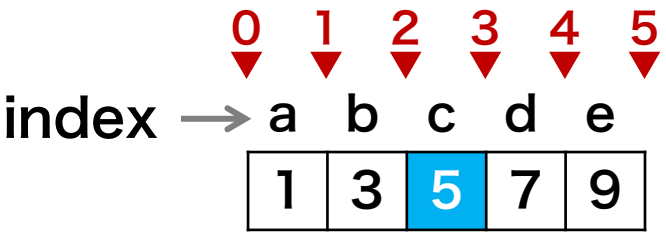



```
import pandas as pd


x = pd.Series([1, 3, 5, 7, 9])




x[2]
# 5


x[x > 3]
# 2    5
# 3    7
# 4    9
# dtype: int64
```

シリーズ

シリーズを作成時に index は、1, 2, 3, ... のように自動的に作られる。しかし、シリーズを作成するときに、index を明示的に与えることで、数値以外の名前をつけることができる。index が与えられた要素は、位置番号のほかに名前で取得することもできる。

シリーズの各要素に位置番号の他に index と呼ばれる索引が自動的に付けられる。




```
import pandas as pd


x = pd.Series([1, 3, 5, 7, 9],
              index=['a', 'b', 'c', 'd', 'e'])



x[2]
# 5


x['c']
# 5
```

シリーズ

Pandas のシリーズの各要素に名前（index）を付けることができる。名前が付けられたシリーズの各要素の値は、名前で取得できるようになる。

```
import pandas as pd
x = pd.Series([1, 3, 5, 7, 9],
              index=['a', 'b', 'c', 'd', 'e'])
x
# a    1
# b    3
# c    5
# d    7
# e    9
# dtype: int64
```

全要素の値を NumPy 配列として取得 ▶

```
x.values
# array([1, 3, 5, 7, 9])
```

全要素の名前を取得 ▶

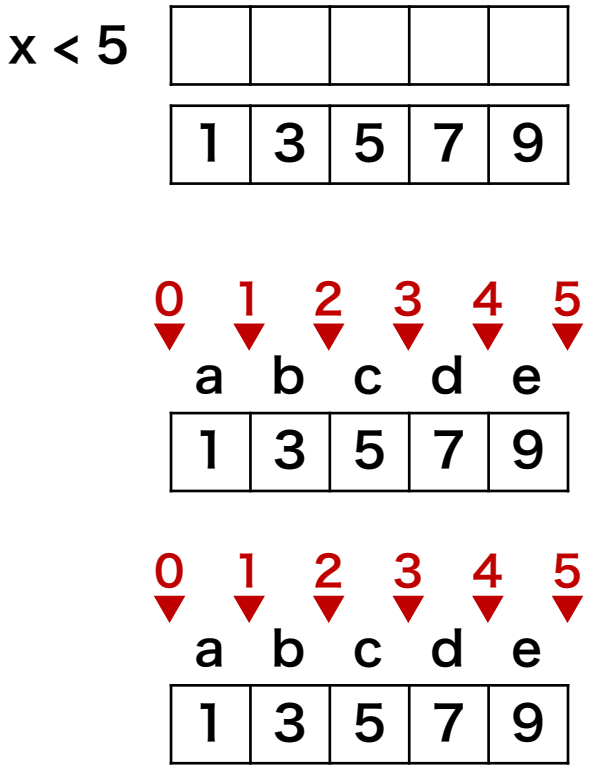
```
x.index
# Index(['a', 'b', 'c', 'd', 'e'],
dtype='object')
```

全要素の名前を NumPy 配列として取得 ▶

```
x.index.values
# array(['a', 'b', 'c', 'd', 'e'],
dtype=object)
```

確認問題

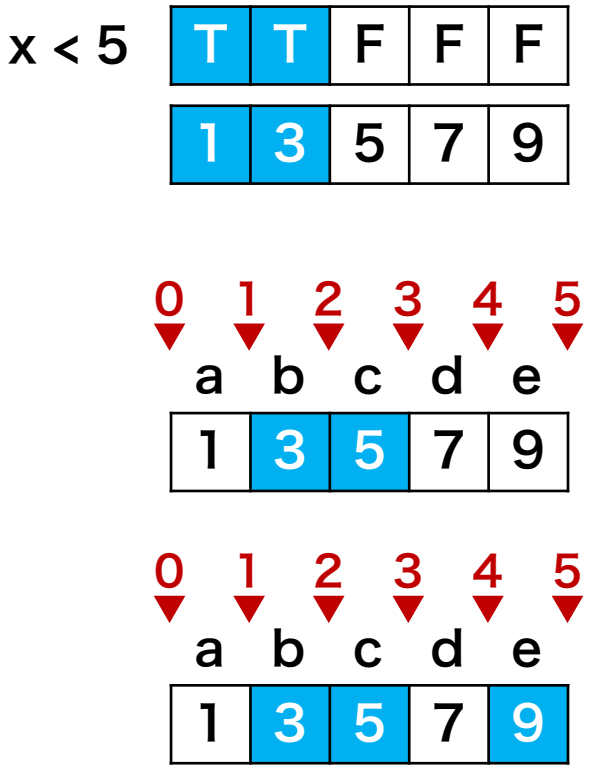
赤字で書かれたスクリプトの実行結果を答えよ。




```
import pandas as pd
x = pd.Series([1, 3, 5, 7, 9],
              index=['a', 'b', 'c', 'd', 'e'])

x[x < 5]


x[1:3]


k = ['b', 'c', 'e']
x[k]
```

確認問題 解答

赤字で書かれたスクリプトの実行結果を答えよ。




```
import pandas as pd
x = pd.Series([1, 3, 5, 7, 9],
              index=['a', 'b', 'c', 'd', 'e'])

x[x < 5]
# a    1
# b    3
# dtype: int64

x[1:3]
# b    3
# c    5
# dtype: int64


k = ['b', 'c', 'e']
x[k]
# b    3
# c    5
# e    9
# dtype: int64
```

数値計算

シリーズに名前がつけられている場合、シリーズ同士の計算は、順番ではなく、名前を基準として計算される。

```
import pandas as pd

x = pd.Series([1, 3, 5, 7, 9],
        index=['a', 'b', 'c', 'd', 'e'])
y = pd.Series([2, 4, 6, 8, 10],
        index=['a', 'b', 'c', 'e', 'd'])

a = x + y
a.values
# array([ 3, 7, 11, 17, 17])

b = x * y
b.values
# array([ 2, 12, 30, 70, 72])
```

数値計算

シリーズに名前がつけられている場合、シリーズ同士の計算は、順番ではなく、名前を基準として計算される。どちらか一方にしか含まれていないような名前がある場合は、ない方のシリーズは欠損値として扱われる。

```
import pandas as pd

x = pd.Series([1, 5, 7, 9],
              index=['a', 'c', 'd', 'e'])
y = pd.Series([2, 4, 6, 8],
              index=['a', 'b', 'c', 'e'])

a = x + y
a.values
# array([ 3., nan, 11., nan, 17.])

b = x * y
b.values
# array([ 2., nan, 30., nan, 72.])
```

数値計算（ブロードキャスト）

２ つのシリーズを対象とした数値演算では、２ つのシリーズの長さが同じでない場合は、ブロードキャスト機能が働く。

```
import pandas as pd

x = pd.Series([1, 3, 5, 7, 9],
              index=['a', 'b', 'c', 'd', 'e'])
y = 2

a = x + y
a.values
# array([ 3, 5, 7, 9, 11])

b = x * y
b.values
# array([ 2, 6, 10, 14, 18])
```

要約統計量

Pandas のシリーズに対して、平均、分散、中央値などの要約統計量を計算するメソッドが多く用意されている。

```
import numpy as np
import pandas as pd

x = pd.Series([1, 2, 3, 4, 5])

x.count()
# 5

x.min()
# 1

x.max()
# 5

x.idxmax()
# 4

x.quantile(0.25)
# 2.0
```

```
x.sum()
# 15

x.mean()
# 3.0

x.median()
# 3.0

x.var()
# 2.5

x.std()
# 1.58113883

x.cumsum()
# 1.0, 3.0, 6.0, 10.0, 15.0
```

欠損値

Pandas のシリーズに欠損値・非数値（np.nan）を含めることができる。欠損値は、Pandas で用意されたメソッドを使って取り除いたり、その位置を調べたりすることができる。

メソッド	動作
dropna	シリーズ中の np.nan を取り除く。
fillna	シリーズ中の np.nan を指定した値で埋める。
isnull	シリーズ中の各要素が np.nan かどうかを True と False で表す。
notnull	is.null と反対の動作を行う。

欠損値除去後のシリーズの要素数が変化するので、複数のシリーズを同時に解析するとき、要素数の違いによりブロードキャスト機能が働き、想定外の計算が行われる可能があることに注意。

```
import numpy as np
import pandas as pd


x = pd.Series([1, 3, np.nan, 7, np.nan])



y = x.dropna()
y
# 0 1.0
# 1 3.0
# 3 7.0
```

欠損値

Pandas のシリーズに欠損値・非数値（np.nan）を含めることができる。欠損値は、Pandas で用意されたメソッドを使って取り除いたり、その位置を調べたりすることができる。

メソッド	動作
dropna	シリーズ中の np.nan を取り除く。
fillna	シリーズ中の np.nan を指定した値で埋める。
isnull	シリーズ中の各要素が np.nan かどうかを True と False で表す。
notnull	is.null と反対の動作を行う。

合理的な根拠なしに、すべての欠損値をある定数に置き換えることはしてはならない。fillna メソッドを使用するときは十分に注意すること。

```
import numpy as np
import pandas as pd


x = pd.Series([1, 3, np.nan, 7, np.nan])



y = x.fillna(0)
y
# 0 1.0
# 1 3.0
# 2 0.0
# 3 7.0
# 4 0.0
```

欠損値

Pandas のシリーズに欠損値・非数値（np.nan）を含めることができる。欠損値は、Pandas で用意されたメソッドを使って取り除いたり、その位置を調べたりすることができる。

メソッド	動作
dropna	シリーズ中の np.nan を取り除く。
fillna	シリーズ中の np.nan を指定した値で埋める。
isnull	シリーズ中の各要素が np.nan かどうかを True と False で表す。
notnull	is.null と反対の動作を行う。

```
import numpy as np
import pandas as pd

x = pd.Series([1, 3, np.nan, 7, np.nan])



y = x.isnull()
y
# 0 False
# 1 False
# 2 True
# 3 False
# 4 True
```

欠損値

Pandas のシリーズに欠損値・非数値（np.nan）を含めることができる。欠損値は、Pandas で用意されたメソッドを使って取り除いたり、その位置を調べたりすることができる。

メソッド	動作
dropna	シリーズ中の np.nan を取り除く。
fillna	シリーズ中の np.nan を指定した値で埋める。
isnull	シリーズ中の各要素が np.nan かどうかを True と False で表す。
notnull	is.null と反対の動作を行う。

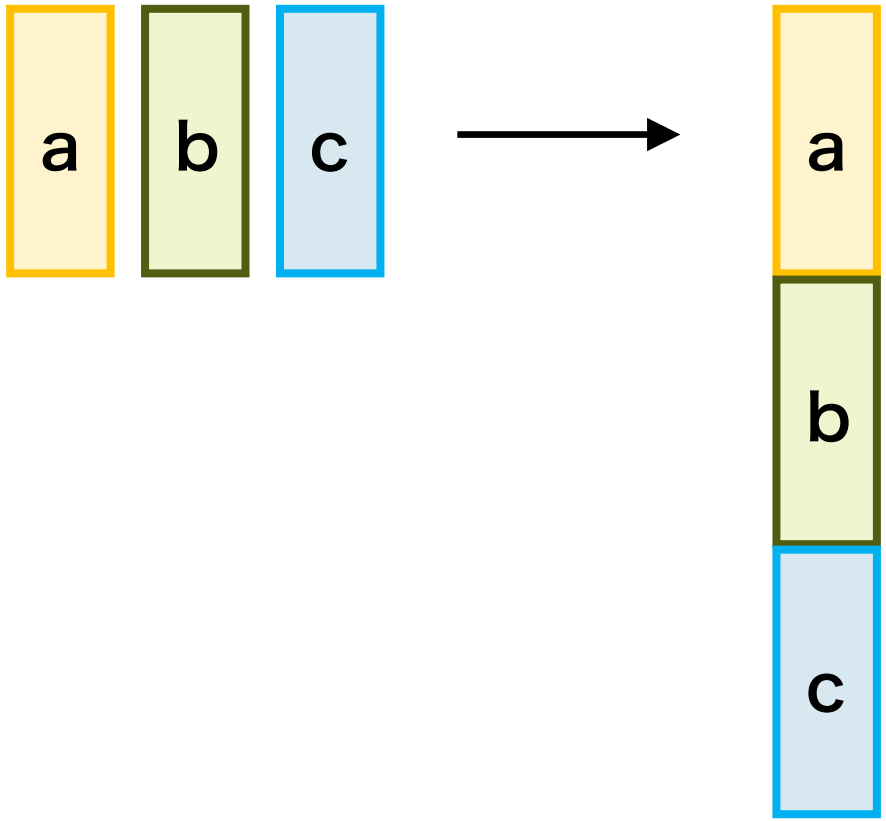
```
import numpy as np
import pandas as pd

x = pd.Series([1, 3, np.nan, 7, np.nan])



y = x.notnull()
y
# 0 True
# 1 True
# 2 False
# 3 True
# 4 False
```

Pandas

- シリーズ
- データフレーム
- 表データ処理

データフレーム（連結）

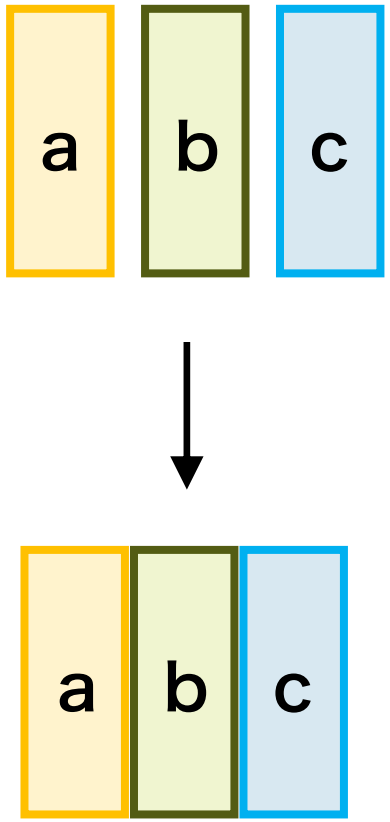
Pandas では行列型のデータをデータフレームと呼ぶ。データフレームは、シリーズを行方向あるいは列方向に束ねることでも作成される。このとき pd.concat 関数を使用する。デフォルトでは行方向に束ねられる。




```
import pandas as pd

a = pd.Series([1, 0, 1, 0])
b = pd.Series([1, 3, 5, 7])
c = pd.Series([0, 2, 4, 6])

d = pd.concat([a, b, c])
d
# 0 1
# 1 0
# 2 1
# 3 0
# 4 1
# 5 3
# 6 5
# 7 7
# : :
```

データフレーム（連結）

Pandas では行列型のデータをデータフレームと呼ぶ。データフレームは、シリーズを行方向あるいは列方向に束ねることでも作成される。このとき pd.concat 関数を使用する。デフォルトでは行方向に束ねられる。axis=1 を指定することで列方向に束ねることもできる。




```
import pandas as pd

a = pd.Series([1, 0, 1, 0])
b = pd.Series([1, 3, 5, 7])
c = pd.Series([0, 2, 4, 6])

d = pd.concat([a, b, c], axis=1)
d
#   0 1 2
# 0 1 1 0
# 1 0 3 2
# 2 1 5 4
# 3 0 7 6
```

データフレーム

データフレームは、複数のシリーズを束ねて作成することができる。例えば、右のように、シリーズを値に持つディクショナリからデータフレームを作成することができる。

```
import pandas as pd

g1 = pd.Series([1, 0, 1, 0, 1])
g2 = pd.Series([1, 3, 5, 7, 9])
g3 = pd.Series([0, 2, 4, 6, 8])

d = {'gene1': g1,
     'gene2': g2,
     'gene3': g3}

df = pd.DataFrame(d)
df
#   gene1 gene2 gene3
# 0     1     1     0
# 1     0     3     2
# 2     1     5     4
# 3     0     7     6
# 4     1     9     8
```

データフレーム

データフレームは、複数のシリーズを束ねて作成することができる。例えば、右のように、シリーズを値に持つディクショナリからデータフレームを作成することができる。シリーズにインデックスが付いている場合は、そのインデックスもデータフレームに反映される。

```
import pandas as pd

g1 = pd.Series([1, 0, 1, 0, 1],
        index=['a', 'b', 'c', 'd', 'e'])
g2 = pd.Series([1, 3, 5, 7, 9],
        index=['a', 'b', 'c', 'd', 'e'])
g3 = pd.Series([0, 2, 4, 6, 8],
        index=['a', 'b', 'c', 'd', 'e'])

d = {'gene1': g1, 'gene2': g2, 'gene3': g3}

df = pd.DataFrame(d)
df
#   gene1 gene2 gene3
# a     1     1     0
# b     0     3     2
# c     1     5     4
# d     0     7     6
# e     1     9     8
```

データフレーム

データフレームは、複数のシリーズを束ねて作成することができる。例えば、右のように、シリーズを値に持つディクショナリからデータフレームを作成することができる。シリーズにインデックスが付いている場合は、そのインデックスもデータフレームに反映される。

```
import pandas as pd

g1 = pd.Series([1, 0, 1, 0, 1],
         index=['a', 'b', 'c', 'd', 'f'])
g2 = pd.Series([1, 3, 5, 7, 9],
         index=['a', 'b', 'c', 'f', 'e'])
g3 = pd.Series([0, 2, 4, 6, 8],
         index=['a', 'b', 'c', 'd', 'e'])

d = {'gene1': g1, 'gene2': g2, 'gene3': g3}

df = pd.DataFrame(d)
df
#   gene1 gene2 gene3
# a   1.0   1.0   0.0
# b   0.0   3.0   2.0
# c   1.0   5.0   4.0
# d   0.0   NaN   6.0
# e   NaN   9.0   8.0
# f   1.0   7.0   NaN
```

データフレーム

データフレームは、シリーズを束ねることによって作成できるほか、二次元リストや二次元配列からも作成できる。この場合、作成したデータフレームに行名（index）と列名（columns）を付ける際に、それぞれ index および columns 引数を使用する。

```
import pandas as pd


d = pd.DataFrame([[11, 12, 13, 14],
                  [21, 22, 23, 24],
                  [31, 32, 33, 34]],
        index=['R1', 'R2', 'R3'],
      columns=['C1', 'C2', 'C3', 'C4'])

d
#     C1 C2 C3 C4
# R1 11 12 13 14
# R2 21 22 23 24
# R3 31 32 33 34
```

データフレーム

データフレームは、シリーズを束ねることによって作成できるほか、二次元リストや二次元配列からも作成できる。この場合、作成したデータフレームに行名（index）と列名（columns）を付ける際に、それぞれ index および columns 引数を使用する。

行名と列名の取得は、index と columns 属性を呼び出すことで行う。

```
import pandas as pd


d = pd.DataFrame([[11, 12, 13, 14],
                  [21, 22, 23, 24],
                  [31, 32, 33, 34]],
        index=['R1', 'R2', 'R3'],
      columns=['C1', 'C2', 'C3', 'C4'])

d
#     C1 C2 C3 C4
# R1 11 12 13 14
# R2 21 22 23 24
# R3 31 32 33 34


d.index
# Index(['R1', 'R2', 'R3'])

d.columns
# Index(['C1', 'C2', 'C3', 'C4'])
```

データフレーム

データフレームは、シリーズを束ねることによって作成できるほか、二次元リストや二次元配列からも作成できる。この場合、作成したデータフレームに行名（index）と列名（columns）を付ける際に、それぞれ index および columns 引数を使用する。

行名と列名の取得は、index と columns 属性を呼び出すことで行う。この場合、呼び出された行名と列名はそれぞれ Pandas の Index 型のオブジェクトになるので、これを配列型に変換するには、values 属性を続けて呼び出す。

```
import pandas as pd


d = pd.DataFrame([[11, 12, 13, 14],
                  [21, 22, 23, 24],
                  [31, 32, 33, 34]],
        index=['R1', 'R2', 'R3'],
      columns=['C1', 'C2', 'C3', 'C4'])

d
#     C1 C2 C3 C4
# R1 11 12 13 14
# R2 21 22 23 24
# R3 31 32 33 34


d.index.values
# array(['R1', 'R2', 'R3'])

d.columns.values
# array(['C1', 'C2', 'C3', 'C4'])
```

データフレーム

データフレームは、シリーズを束ねることによって作成できるほか、二次元リストや二次元配列からも作成できる。この場合、作成したデータフレームに行名（index）と列名（columns）を付ける際に、それぞれ index および columns 引数を使用する。

行名と列名の取得は、index と columns 属性を呼び出すことで行う。この場合、呼び出された行名と列名はそれぞれ Pandas の Index 型のオブジェクトになるので、これを配列型に変換するには、values 属性を続けて呼び出す。

データフレームの index と columns は、あとから変更することができる。後から変更する場合は、データフレームの index と columns 属性に直接新しい名前を代入する。

特定の行名または列名だけを新しい名前に変更したいとき、Pandas の rename メソッドを使用すると便利である。

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.rename.html

```
import pandas as pd

d = pd.DataFrame([[11, 12, 13, 14],
                  [21, 22, 23, 24],
                  [31, 32, 33, 34]],
        index=['R1', 'R2', 'R3'],
      columns=['C1', 'C2', 'C3', 'C4'])

d.index = ['x', 'y', 'z']
d.columns = ['h', 'i', 'j', 'k']

d
#    h  i  j  k
# x 11 12 13 14
# y 21 22 23 24
# z 31 32 33 34
```

データフレーム要素参照

データフレームから要素を取得する方法として、位置番号から取得する方法と行名・列名で取得する方法の２通りが方法が用意されている。位置番号で取得する場合は、iloc を使用し、その際に、0 から始まる位置番号を与える。

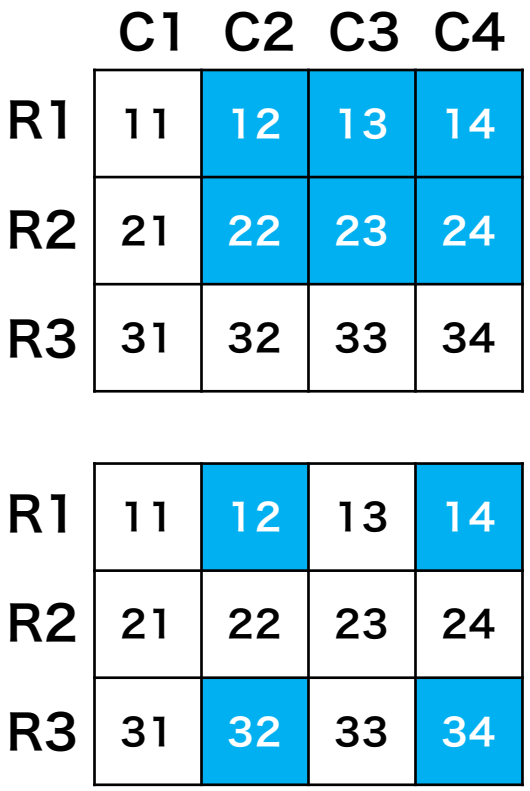
	C1	C2	C3	C4
R1	11	12	13	14
R2	21	22	23	24
R3	31	32	33	34

R1	11	12	13	14
R2	21	22	23	24
R3	31	32	33	34

```
import pandas as pd

d = pd.DataFrame([[11, 12, 13, 14],
                  [21, 22, 23, 24],
                  [31, 32, 33, 34]],
        index=['R1', 'R2', 'R3'],
      columns=['C1', 'C2', 'C3', 'C4'])


d.iloc[0, :]
# C1 11
# C2 12
# C3 13
# C4 14


d.iloc[:, 2]
# R1 13
# R2 23
# R3 33
```

データフレーム要素参照

データフレームから要素を取得する方法として、位置番号から取得する方法と行名・列名で取得する方法の２通りが方法が用意されている。位置番号で取得する場合は、iloc を使用し、その際に、0 から始まる位置番号を与える。




```
import pandas as pd

d = pd.DataFrame([[11, 12, 13, 14],
                  [21, 22, 23, 24],
                  [31, 32, 33, 34]],
        index=['R1', 'R2', 'R3'],
      columns=['C1', 'C2', 'C3', 'C4'])


d.iloc[0:2, 1:4]
#     C2 C3 C4
# R1 12 13 14
# R2 22 23 24



d.iloc[[0, 2], [1, 3]]
#     C2 C4
# R1 12 14
# R3 32 34
```

NumPy の動作と異なるので、両者を混同しないように。

データフレーム要素参照

行名・列名で取得する場合は、loc を使用し、その際に、行名および列名を与えて取得する。

	C1	C2	C3	C4
R1	11	12	13	14
R2	21	22	23	24
R3	31	32	33	34

R1	11	12	13	14
R2	21	22	23	24
R3	31	32	33	34

```
import pandas as pd

d = pd.DataFrame([[11, 12, 13, 14],
                  [21, 22, 23, 24],
                  [31, 32, 33, 34]],
        index=['R1', 'R2', 'R3'],
      columns=['C1', 'C2', 'C3', 'C4'])


d.loc['R1', :]
# C1 11
# C2 12
# C3 13
# C4 14


d.loc[:, 'C3']
# R1 13
# R2 23
# R3 33
```

データフレーム要素参照

行名・列名で取得する場合は、loc を使用し、その際に、行名および列名を与えて取得する。

	C1	C2	C3	C4
R1	11	12	13	14
R2	21	22	23	24
R3	31	32	33	34

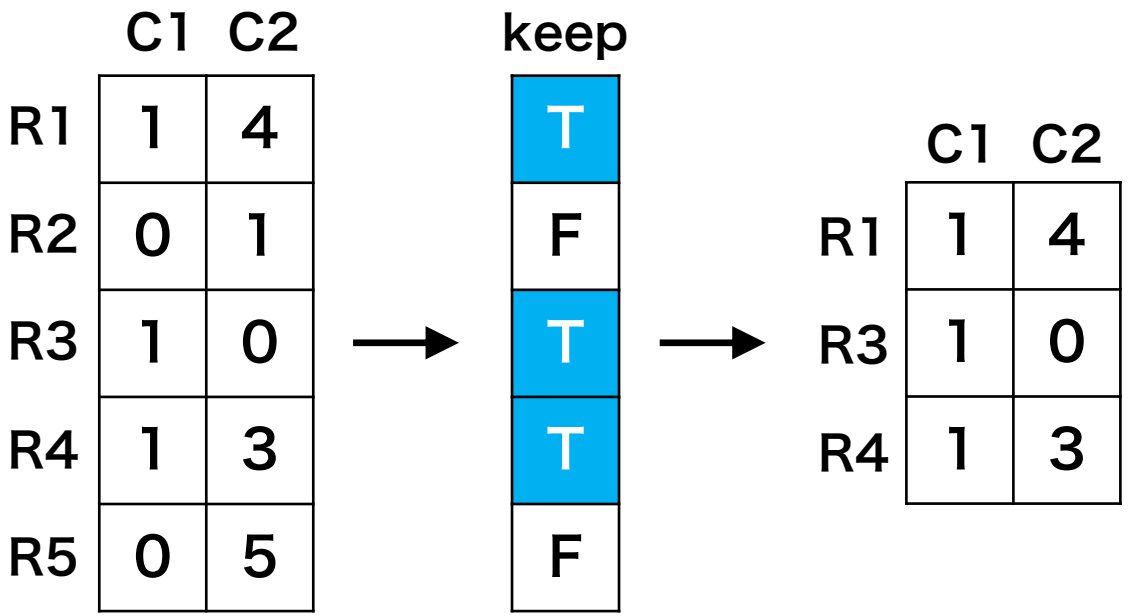
R1	11	12	13	14
R2	21	22	23	24
R3	31	32	33	34

```
import pandas as pd

d = pd.DataFrame([[11, 12, 13, 14],
                  [21, 22, 23, 24],
                  [31, 32, 33, 34]],
        index=['R1', 'R2', 'R3'],
      columns=['C1', 'C2', 'C3', 'C4'])


d.loc[['R1', 'R2'],['C2', 'C3', 'C4']]
#    C2 C3 C4
# R1 12 13 14
# R2 22 23 24



d.loc[['R1', 'R3'], ['C2', 'C4']]
#    C2 C4
# R1 12 14
# R3 32 34
```

NumPy の動作と異なるので、両者を混同しないように。

データフレーム要素参照

データフレームもシリーズと同様に、True と False を使って要素を取得することができる。例えば、C1 列の値が 0 よりも大きいときの行を抽出するなど。True/False を使用する場合は、loc または iloc の両方を使用することができる。




```
import pandas as pd

d = pd.DataFrame([[1, 4],
                  [0, 1],
                  [1, 0],
                  [1, 3],
                  [0, 5]],
   index=['R1', 'R2', 'R3', 'R4', 'R5'],
 columns=['C1', 'C2'])


keep = (d.loc[:, 'C1'] > 0)
d.loc[keep, :]
#     C1 C2
# R1   1  4
# R3   1  0
# R4   1  3
```

データフレーム要素参照

前出の .iloc および .loc 以外にも、.iat および .at を利用した要素参照や列名・行名属性を用いた参照方法などがある。

```
import pandas as pd
df = pd.DataFrame([[1, 4], [0, 1],
                   [1, 0], [1, 3],
                   [0, 5]],
  index=['R1','R2','R3','R4','R5'],
  columns=['C1','C2'])
df.iloc[0:1, 2:4]
df.loc[['R1'], ['C2', 'C4']]
df.iat[0, 3]
df.at['R2', 'C3']
df[0:1]
df[['C1', 'C3']]
df.C2
```

df.iloc	整数からなる位置番号を指定して、該当位置の要素を取得する。複数列や複数行の要素をまとめて取得できる。
df.loc	列名あるいは行名を指定して、該当位置の要素を取得する。複数列や複数行の要素をまとめて取得できる。
df.iat	.iloc と同じ使い方をする。ただし、要素を 1 つしか取得できない。
df.at	.loc と同じ使い方をする。ただし、要素を 1 つしか取得できない。
df[0:1]	行の位置番号をスライス表示で指定して、該当行の要素を取得する。複数行の要素をまとめて取得できる。
df[['C1']]	列の名前をリストで指定して、該当列の要素を取得する。複数列の要素をまとめて取得できる。ただし、スライス表記は使用できない。
df.C1	列の名前をオブジェクトの属性として、該当列の要素を取得することができる。

データフレーム（連結）

pd.concat 関数を使用することで、複数のデータフレームを結合させて 1 つのデータフレームにまとめることができる。

11	12	13	14
21	22	23	24

31	32	33	34
41	42	43	44

11	12	13	14
21	22	23	24
31	32	33	34
41	42	43	44

```
import pandas as pd

d1 = pd.DataFrame([[11, 12, 13, 14],
                   [21, 22, 23, 24]])
d2 = pd.DataFrame([[31, 32, 33, 34],
                   [41, 42, 43, 44]])

df = pd.concat([d1, d2])
df
#    0  1  2  3
# 0 11 12 13 14
# 1 21 22 23 24
# 2 31 32 33 34
# 3 41 42 43 44
```

データフレーム（連結）

pd.concat 関数を使用することで、複数のデータフレームを結合させて 1 つのデータフレームにまとめることができる。

11	12	13	14
21	22	23	24

31	32	33	34
41	42	43	44

11	12	13	14	31	32	33	34
21	22	23	24	41	42	43	44

```
import pandas as pd

d1 = pd.DataFrame([[11, 12, 13, 14],
                   [21, 22, 23, 24]])
d2 = pd.DataFrame([[31, 32, 33, 34],
                   [41, 42, 43, 44]])

df = pd.concat([d1, d2], axis=1)
df
#    0  1  2  3  0  1  2  3
# 0 11 12 13 14 31 32 33 34
# 1 21 22 23 24 41 42 43 44
```

データフレーム（連結）

pd.concat 関数を使用することで、複数のデータフレームを結合させて 1 つのデータフレームにまとめることができる。データフレームにインデックスと列名が存在するときは、インデックスと列名に基づいて結合されることに注意。

```
import pandas as pd

d1 = pd.DataFrame([[11, 12, 13, 14],
                   [21, 22, 23, 24]],
        index=['A', 'B'],
        columns=['a', 'b', 'c', 'd'])


d2 = pd.DataFrame([[31, 32, 33, 34],
                   [41, 42, 43, 44]],
        index=['X', 'Y'],
        columns=['a', 'b', 'c', 'e'])



df = pd.concat([d1, d2])
df
#    a  b  c    d    e
# A 11 12 13 14.0  NaN
# B 21 22 23 24.0  NaN
# X 31 32 33  NaN 34.0
# Y 41 42 43  NaN 44.0
```

データフレーム（結合）

複数のデータフレームをある列の値に基づいて、マージするすることができる。このとき merge 関数を使用する。

k	V1
a	1
b	1
c	0

k	V2
a	9
b	7
d	8

merge

k	V1	V2
a	1	9
b	1	7

データフレームの結合を行うとき、キーとなる列に重複要素が含まれると、予期しない挙動になる場合があるため、十分に注意すること。

```
import pandas as pd

d1 = pd.DataFrame([['a', 1],
                   ['b', 1],
                   ['c', 0]],
                  columns=['k', 'V1'])

d2 = pd.DataFrame([['a', 9],
                   ['b', 7],
                   ['d', 8]],
                  columns=['k', 'V2'])


d = pd.merge(d1, d2)   # how='inner'
d
#   k V1 V2
# 0 a  1  9
# 1 b  1  7
```

データフレーム（結合）

複数のデータフレームをある列の値に基づいて、マージするすることができる。このとき merge 関数を使用する。

k	V1
a	1
b	1
c	0

k	V2
a	9
b	7
d	8

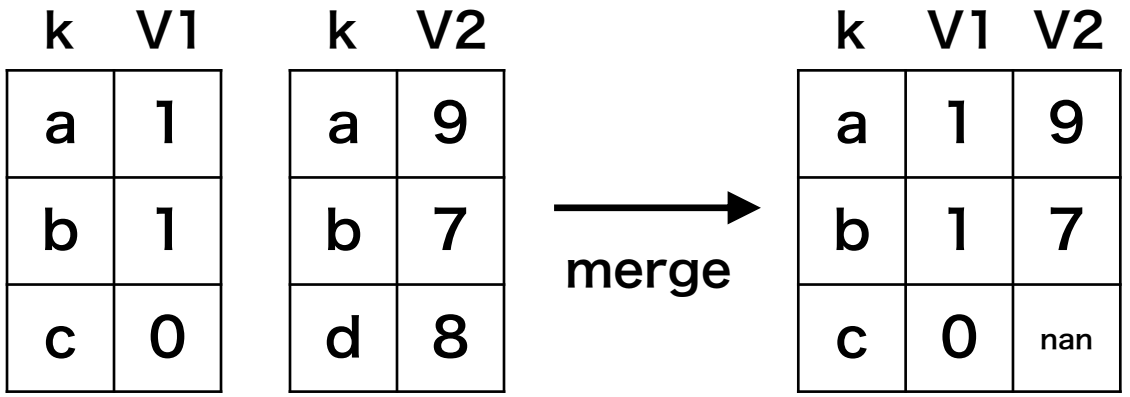
merge

k	V1	V2
a	1	9
b	1	7
c	0	nan
d	nan	8

```
import pandas as pd

d1 = pd.DataFrame([['a', 1],
                   ['b', 1],
                   ['c', 0]],
                  columns=['k', 'V1'])

d2 = pd.DataFrame([['a', 9],
                   ['b', 7],
                   ['d', 8]],
                  columns=['k', 'V2'])


d = pd.merge(d1, d2, how='outer')
d
#    k  V1  V2
# 0  a   1   9
# 1  b   1   7
# 2  c   0 nan
# 3  d nan   8
```

データフレーム（結合）

複数のデータフレームをある列の値に基づいて、マージするすることができる。このとき merge 関数を使用する。




```
import pandas as pd

d1 = pd.DataFrame([['a', 1],
                   ['b', 1],
                   ['c', 0]],
                  columns=['k', 'V1'])

d2 = pd.DataFrame([['a', 9],
                   ['b', 7],
                   ['d', 8]],
                  columns=['k', 'V2'])


d = pd.merge(d1, d2, how='left')
d
#   k  V1  V2
# 0 a   1   9
# 1 b   1   7
# 2 c   0 nan
```

データフレーム（結合）

複数のデータフレームをある列の値に基づいて、マージするすることができる。このとき merge 関数を使用する。

k	V1
a	1
b	1
c	0

k	V2
a	9
b	7
d	8

merge

k	V1	V2
a	1	9
b	1	7
d	nan	8

```
import pandas as pd

d1 = pd.DataFrame([['a', 1],
                   ['b', 1],
                   ['c', 0]],
                  columns=['k', 'V1'])

d2 = pd.DataFrame([['a', 9],
                   ['b', 7],
                   ['d', 8]],
                  columns=['k', 'V2'])


d = pd.merge(d1, d2, how='right')
d
#   k  V1  V2
# 0 a   1   9
# 1 b   1   7
# 2 d nan   8
```

データフレーム（結合）

複数のデータフレームをある列の値に基づいて、マージするすることができる。このとき merge 関数を使用する。

k	V1
a	1
b	1
c	0

f	V2
a	9
b	7
d	8

merge

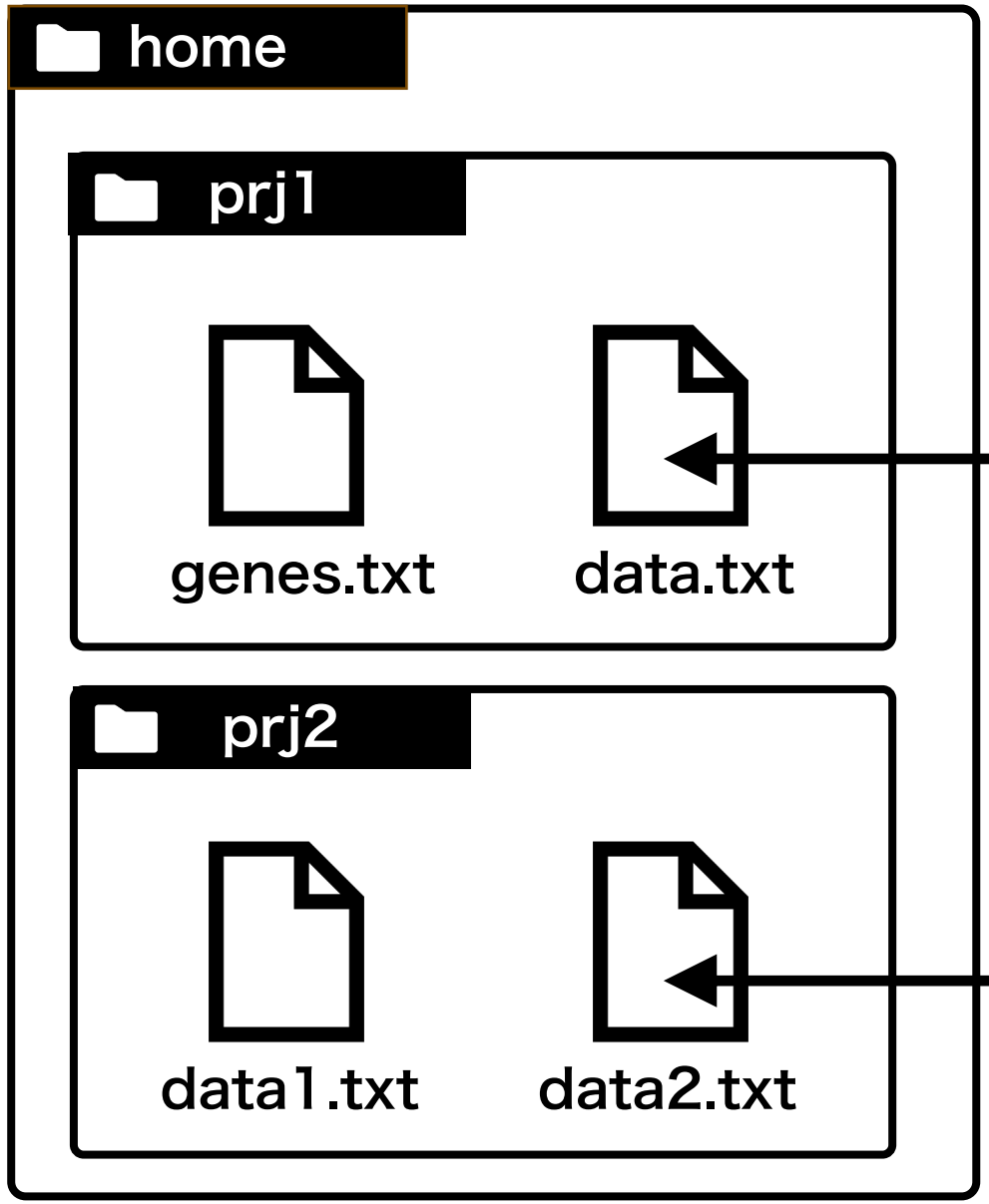
k	V1	f	V2
a	1	a	9
b	1	b	7
c	0	nan	nan
nan	nan	d	8

```
import pandas as pd

d1 = pd.DataFrame([['a', 1],
                   ['b', 1],
                   ['c', 0]],
                  columns=['k', 'V1'])

d2 = pd.DataFrame([['a', 9],
                   ['b', 7],
                   ['d', 8]],
                  columns=['f', 'V2'])


d = pd.merge(d1, d2, how='outer',
             left_on='k', right_on='f')
d
#      k  V1    f  V2
# 0    a   1    a   9
# 1    b   1    b   7
# 2    c   0  nan nan
# 3  nan nan    d   8
```

Pandas

- シリーズ
- データフレーム
- 表データ処理

パス



/home/prj1/data.txt

/home/prj2/data2.txt

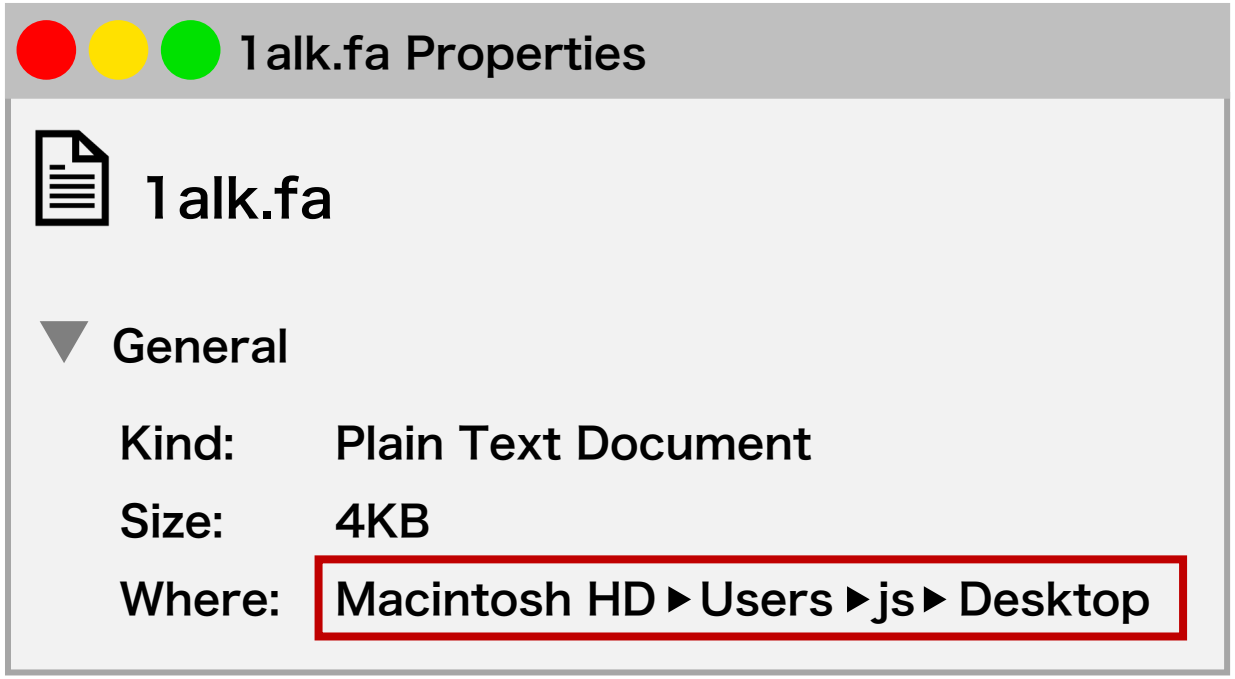
ファイルパスの調べ方

Macintosh

1. パスを調べたいファイルを右クリックし、「Get Info」を選ぶ。

2. 「General」タブの「Where」項目にファイルへのパスが記載されている。

    - ファイルパスの「Macintosh HD」は最上層を表し、パスを書くとき「/」と書く。
    - 小さい三角形はフォルダの包含関係を表すので、パスを書くときは「/」と書く。



/Users/js/Desktop/1alk.fa

ファイルパスの調べ方

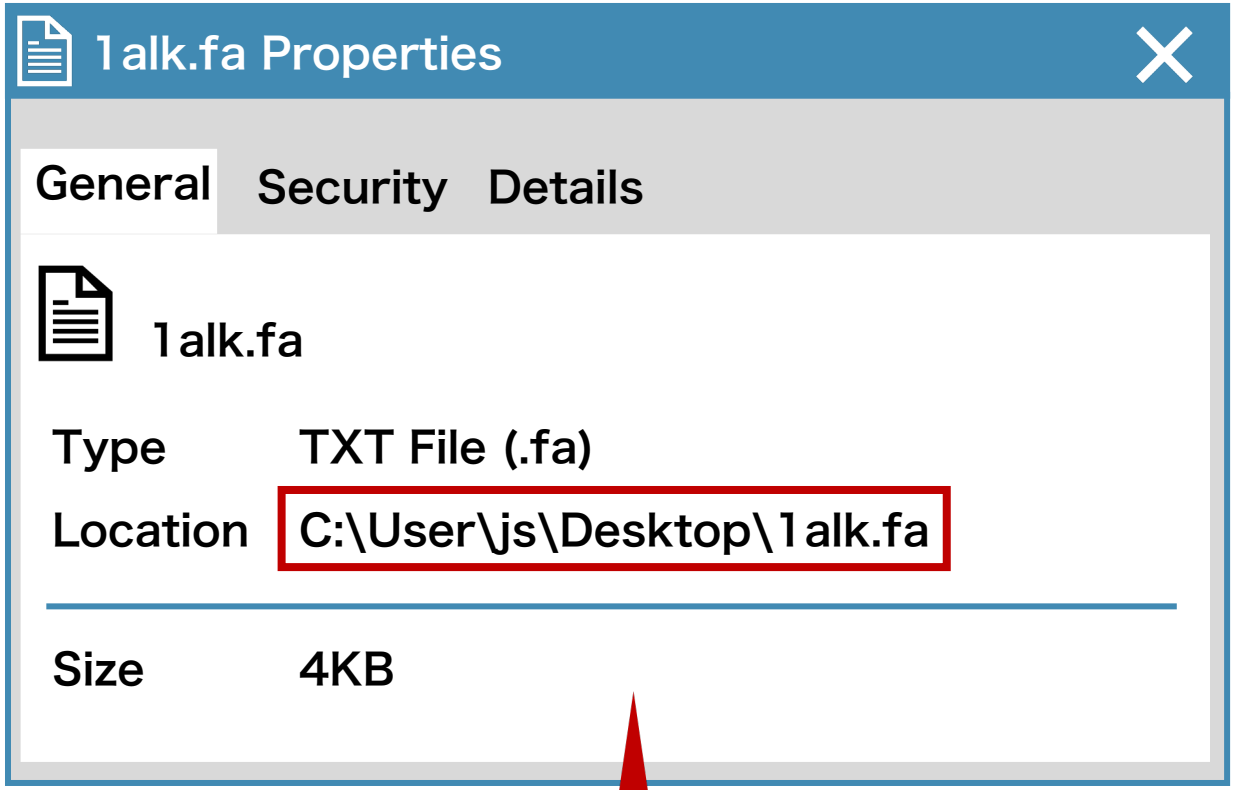
Windows

1. パスを調べたいファイルを右クリックし、「Properties」を選ぶ。

2. 「General」タブの「Location」項目にファイルへのパスが記載されている。

    - 日本語環境の場合、「\」が「¥」として表示される。どちらでも Windows コンピューター内部では 0x5C として認識されている。

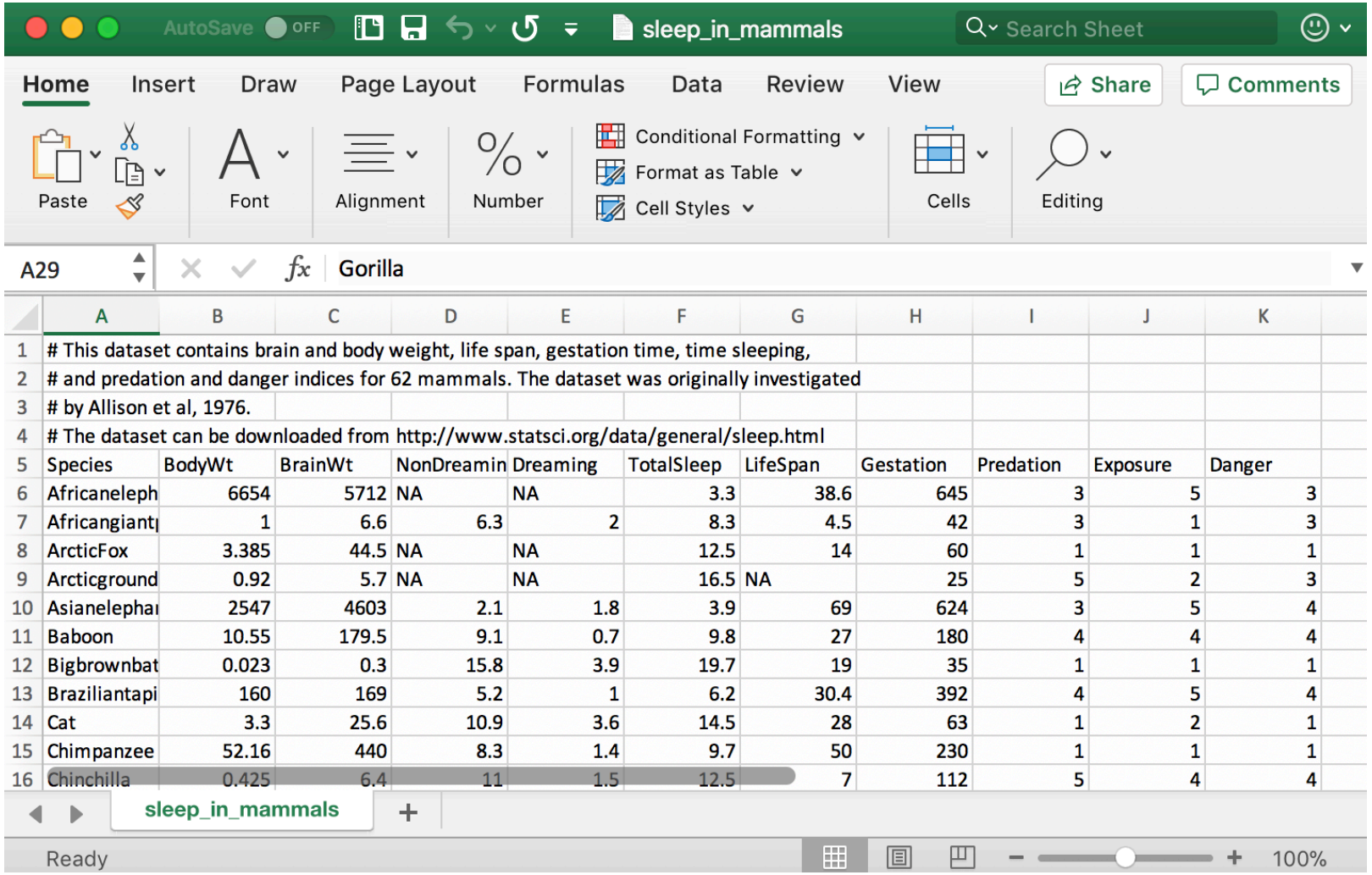
    - Location 欄に書かれているパス中の「\」は、パスとして使用するとき「/」に置き換える。



C:/User/js/Desktop/1alk.fa

表データ

生物学で取り扱うデータは、一般的に、各列には属性、各行にサンプルを書く構造をとっている。このような表形式のデータを記録したり、整理したりする場合は、Excel 等の表計算アプリケーションを使うのが一般的である。しかし、プログラミング言語でデータを解析する場合は、Excel データのままでは非常に不便である。ほとんどの場合、Excel からデータをタブ区切りファイル（TSV）またはカンマ区切りファイル（CSV）として書き出してから解析するのが一般的である。



	A	B	C	D	E	F	G	H	I	J	K
1	# This dataset contains brain and body weight, life span, gestation time, time sleeping,										
2	# and predation and danger indices for 62 mammals. The dataset was originally investigated										
3	# by Allison et al, 1976.										
4	# The dataset can be downloaded from http://www.statsci.org/data/general/sleep.html										
5	Species	BodyWt	BrainWt	NonDreamin	Dreaming	TotalSleep	LifeSpan	Gestation	Predation	Exposure	Danger
6	Africaneleph	6654	5712	NA	NA	3.3	38.6	645	3	5	3
7	Africangiant	1	6.6	6.3	2	8.3	4.5	42	3	1	3
8	ArcticFox	3.385	44.5	NA	NA	12.5	14	60	1	1	1
9	Arcticground	0.92	5.7	NA	NA	16.5	NA	25	5	2	3
10	Asianelephar	2547	4603	2.1	1.8	3.9	69	624	3	5	4
11	Baboon	10.55	179.5	9.1	0.7	9.8	27	180	4	4	4
12	Bigbrownbat	0.023	0.3	15.8	3.9	19.7	19	35	1	1	1
13	Braziliantapi	160	169	5.2	1	6.2	30.4	392	4	5	4
14	Cat	3.3	25.6	10.9	3.6	14.5	28	63	1	2	1
15	Chimpanzee	52.16	440	8.3	1.4	9.7	50	230	1	1	1
16	Chinchilla	0.425	6.4	11	1.5	12.5	7	112	5	4	4

表データ

https://aabbdd.jp/data/sleep_in_mammals.txt

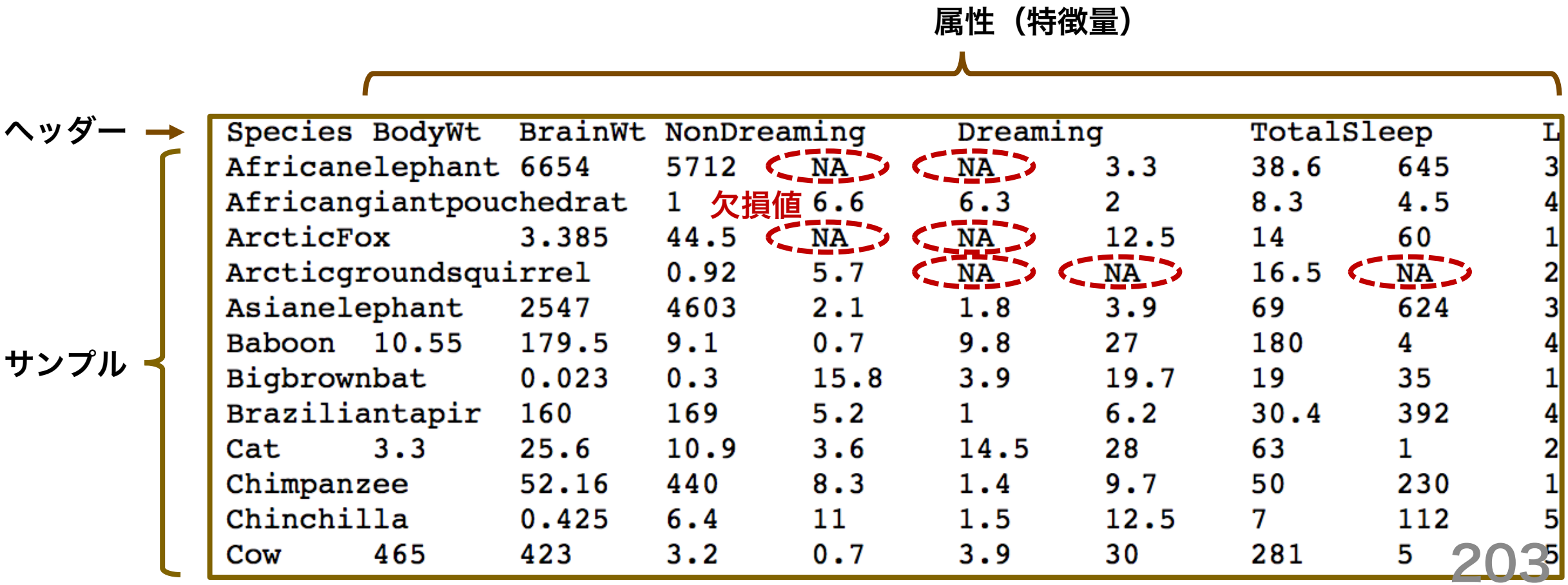
コメント

データ

```
# This dataset contains brain and body weight, life span, gestation time,
# and predation and danger indices for 62 mammals. The dataset was origin
# by Allison et al, 1976.
# The dataset can be downloaded from http://www.statsci.org/data/general/
Species BodyWt  BrainWt NonDreaming     Dreaming         TotalSleep       L
Africanelephant 6654    5712    NA      NA      3.3      38.6    645      3
Africangiantpouchedrat  1       6.6     6.3     2        8.3     4.5      4
ArcticFox       3.385   44.5    NA      NA      12.5     14      60       1
Arcticgroundsquirrel    0.92    5.7     NA      NA       16.5    NA       2
Asianelephant   2547    4603    2.1     1.8     3.9      69      624      3
Baboon  10.55   179.5   9.1     0.7     9.8      27      180     4        4
Bigbrownbat     0.023   0.3     15.8    3.9      19.7    19      35       1
Braziliantapir  160     169     5.2     1       6.2      30.4    392      4
Cat     3.3     25.6    10.9    3.6     14.5     28      63      1        2
Chimpanzee      52.16   440     8.3     1.4     9.7      50      230      1
Chinchilla      0.425   6.4     11      1.5     12.5     7       112      5
Cow     465     423     3.2     0.7     3.9      30      281     5        5
```

表データ

https://aabbdd.jp/data/sleep_in_mammals.txt



Species	BodyWt	BrainWt	NonDreaming	Dreaming		TotalSleep		L
Africanelephant	6654	5712	NA	NA	3.3	38.6	645	3
Africangiantpouchedrat	1	6.6	6.3	2	8.3	4.5	4	
ArcticFox	3.385	44.5	NA	NA	12.5	14	60	1
Arcticgroundsquirrel		0.92	5.7	NA	NA	16.5	NA	2
Asianelephant	2547	4603	2.1	1.8	3.9	69	624	3
Baboon	10.55	179.5	9.1	0.7	9.8	27	180	4
Bigbrownbat		0.023	0.3	15.8	3.9	19.7	19	35
Braziliantapir	160	169	5.2	1	6.2	30.4	392	4
Cat	3.3	25.6	10.9	3.6	14.5	28	63	1
Chimpanzee		52.16	440	8.3	1.4	9.7	50	230
Chinchilla		0.425	6.4	11	1.5	12.5	7	112
Cow	465	423	3.2	0.7	3.9	30	281	5

Pandas によるファイル読み込み

Pandas では read_table 関数（メソッド）を使うことで、CSV または TSV データを読み込むことができる。

```
import pandas as pd

file = 'sleep_in_mammals.txt'

d = pd.read_csv(file)
```

```
pandas/_libs/parsers.pyx in pandas._libs.parsers.TextReader.read()

pandas/_libs/parsers.pyx in pandas._libs.parsers.TextReader._read_low_memory()

pandas/_libs/parsers.pyx in pandas._libs.parsers.TextReader._read_rows()

pandas/_libs/parsers.pyx in pandas._libs.parsers.TextReader._tokenize_rows()

pandas/_libs/parsers.pyx in pandas._libs.parsers.raise_parser_error()

ParserError: Error tokenizing data. C error: Expected 1 fields in line 5, saw 11
```

sleep_in_mammals.txt が read_table が想定しているフォーマットに従っていないため、エラーとなった。

https://aabbdd.jp/data/sleep_in_mammals.txt

Pandas によるファイル読み込み

Pandas では read_table 関数（メソッド）を使うことで、CSV または TSV データを読み込むことができる。ここで読み込もうとしているファイルはタブ区切りで、の最初の数行は # から始まるコメントであり、それに続き 1 行目がヘッダー、それ以降データと続くので、この情報もファイル読み込み関数 read_table に与える必要がある。

```
import pandas as pd
file = 'sleep_in_mammals.txt'
d = pd.read_csv(file,
                comment='#',
                header=0,
                sep='\t')
```

Python では、バックスラッシュは、エスケープ文字と呼ばれている。エスケープ文字の後に続く1文字は特別な意味を持つ。例えば't' は英文字の t を表すが、'\t' はタブを表す。

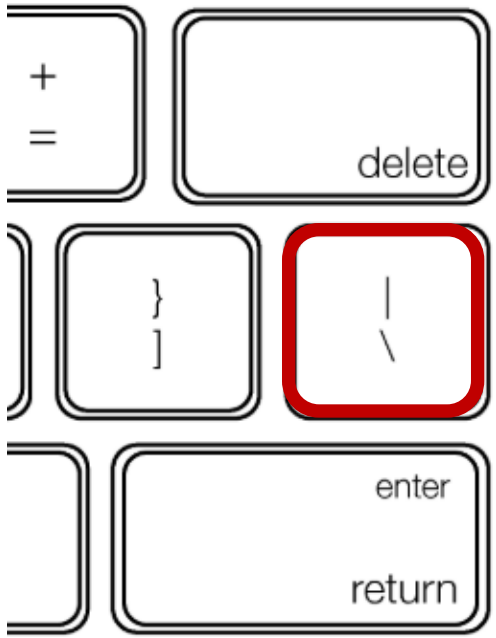
https://aabbdd.jp/data/sleep_in_mammals.txt

バックスラッシュ

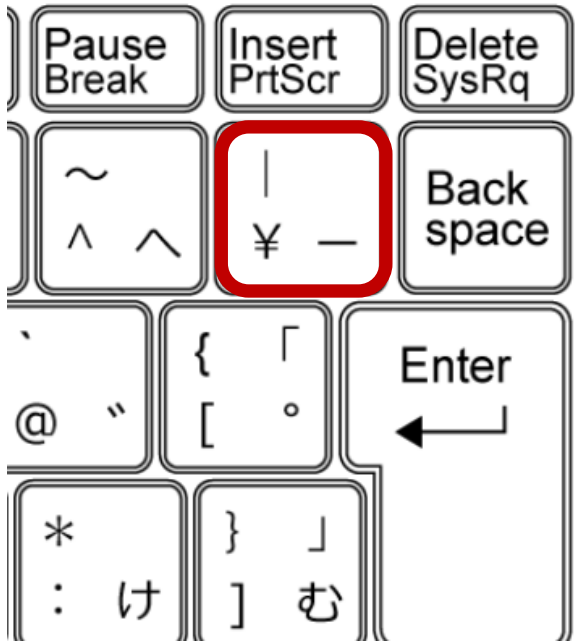
R では、バックスラッシュは、エスケープ文字と呼ばれている。エスケープ文字の後に続く1文字は特別な意味を持つ。例えば't' は英文字の t を表すが、'\t' はタブを表す。また、'd' は英文字 d を表すが、'\d' は 0〜9 までの数字を表す。

バックスラッシュは、日本語環境（日本語キーボード、日本語フォントなどを含む）では円マークとして印字・表示される。バックスラッシュと円マークは、印字・表示が異なるが、コンピューター上では同じもの（0x5C）として扱われる。

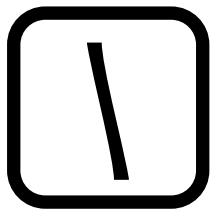
英語（US）配列

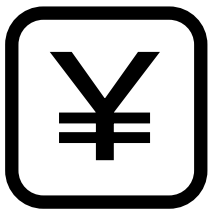


日本語配列



MacOS 日本語環境を使用している場合は \ と￥の両方を入力できる。Option を押しながら \ または￥キーを押すことで、\ と￥の入力の切り替えができる。





Pandas によるファイル読み込み

Pandas では read_table 関数（メソッド）を使うことで、CSV または TSV データを読み込むことができる。ここで読み込もうとしているファイルの最初の数行は # から始まるコメントであり、それからヘッダー、データと続くので、この情報もファイル読み込み関数 read_table に与える必要がある。

```
import pandas as pd

file = 'sleep_in_mammals.txt'

d = pd.read_csv(file, comment='#', header=0, sep='\t')


d.shape

# (62, 11)


d.head()
```

	Species	BodyWt	BrainWt	NonDreaming	Dreaming	TotalSleep	LifeSpan	Gestation	Predation	Exposure	Danger
0	Africanelephant	6654.000	5712.0	NaN	NaN	3.3	38.6	645.0	3	5	3
1	Africangiantpouchedrat	1.000	6.6	6.3	2.0	8.3	4.5	42.0	3	1	3
2	ArcticFox	3.385	44.5	NaN	NaN	12.5	14.0	60.0	1	1	1
3	Arcticgroundsquirrel	0.920	5.7	NaN	NaN	16.5	NaN	25.0	5	2	3
4	Asianelephant	2547.000	4603.0	2.1	1.8	3.9	69.0	624.0	3	5	4

Pandas によるファイル読み込み

Pandas では read_table 関数（メソッド）を使うことで、CSV または TSV データを読み込むことができる。ここで読み込もうとしているファイルの最初の数行は # から始まるコメントであり、それからヘッダー、データと続くので、この情報もファイル読み込み関数 read_table に与える必要がある。行名（インデックス）を指定することもできる。

```
import pandas as pd
file = 'sleep_in_mammals.txt'
d = pd.read_csv(file, comment='#', header=0, sep='\t',
                index_col=0)
d.shape
# (62, 10)


d.head()
```

Species	BodyWt	BrainWt	NonDreaming	Dreaming	TotalSleep	LifeSpan	Gestation	Predation	Exposure	Danger
Africanelephant	6654.000	5712.0	NaN	NaN	3.3	38.6	645.0	3	5	3
Africangiantpouchedrat	1.000	6.6	6.3	2.0	8.3	4.5	42.0	3	1	3
ArcticFox	3.385	44.5	NaN	NaN	12.5	14.0	60.0	1	1	1
Arcticgroundsquirrel	0.920	5.7	NaN	NaN	16.5	NaN	25.0	5	2	3
Asianelephant	2547.000	4603.0	2.1	1.8	3.9	69.0	624.0	3	5	4

行名

データフレーム

Pandas の read_table で読み込んだデータはデータフレームの形で保存される。Pandas のデータフレームには様々な便利なメソッドが用意されている。

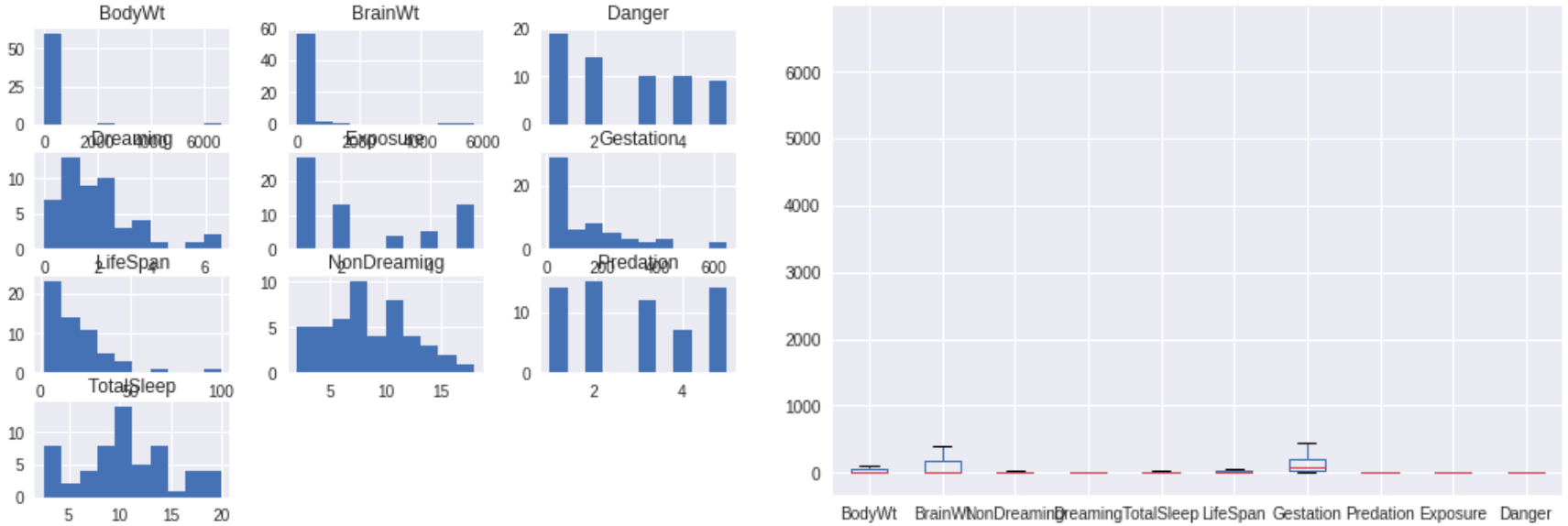
```
import pandas as pd
file = 'sleep_in_mammals.txt'
d = pd.read_csv(file, comment='#', header=0, sep='\t',
                index_col=0)
d.describe()
```

	BodyWt	BrainWt	NonDreaming	Dreaming	TotalSleep	LifeSpan	Gestation	Predation	Exposure	Danger
count	62.000000	62.000000	48.000000	50.000000	58.000000	58.000000	58.000000	62.000000	62.000000	62.000000
mean	198.789984	283.134194	8.672917	1.972000	10.532759	19.877586	142.353448	2.870968	2.419355	2.612903
std	899.158011	930.278942	3.666452	1.442651	4.606760	18.206255	146.805039	1.476414	1.604792	1.441252
min	0.005000	0.140000	2.100000	0.000000	2.600000	2.000000	12.000000	1.000000	1.000000	1.000000
25%	0.600000	4.250000	6.250000	0.900000	8.050000	6.625000	35.750000	2.000000	1.000000	1.000000
50%	3.342500	17.250000	8.350000	1.800000	10.450000	15.100000	79.000000	3.000000	2.000000	2.000000
75%	48.202500	166.000000	11.000000	2.550000	13.200000	27.750000	207.500000	4.000000	4.000000	4.000000
max	6654.000000	5712.000000	17.900000	6.600000	19.900000	100.000000	645.000000	5.000000	5.000000	5.000000

データフレーム

Pandas の read_table で読み込んだデータはデータフレームの形で保存される。Pandas のデータフレームには様々な便利なメソッドが用意されている。

```
import pandas as pd
file = 'sleep_in_mammals.txt'
d = pd.read_csv(file, comment='#', header=0, sep='\t',
                index_col=0)
d.hist()
d.boxplot()
```



※ 一部の属性の値の範囲が大きいので、対数化してからグラフに描くと全体の傾向を掴めやすくなる。
※ matplotlib でグラフを描いた方が一般的であるので、この講習会では Pandas の視覚化機能を取り上げない。

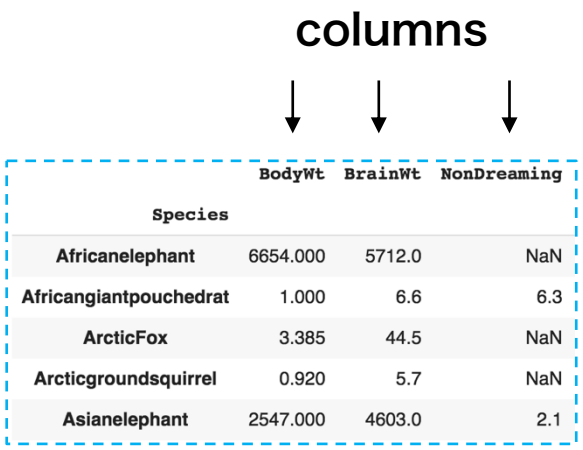
データフレーム

Pandas の read_table で読み込んだデータはデータフレームの形で保存される。Pandas のデータフレームには様々な便利なメソッドが用意されている。

```
import pandas as pd
file = 'sleep_in_mammals.txt'
d = pd.read_csv(file, comment='#', header=0, sep='\t',
                index_col=0)
dir(d)
# if __name__ == '__main__':
# ['BodyWt',
#  'BrainWt',
#  'Danger',
#  'Dreaming',
#  ...
#  'values',
#  'var',
#  'where',
#  'xs']
```

データフレーム

Pandas のデータフレームの列名および行名は、それぞれ columns および index で取得できる。

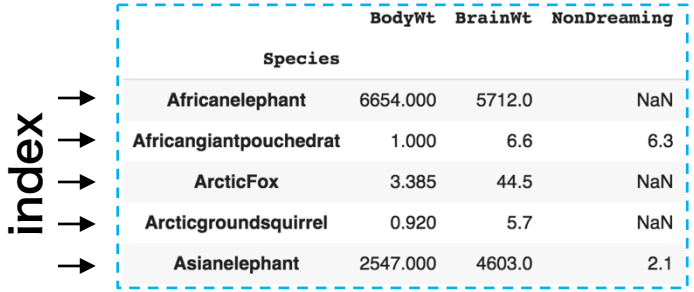


Species	BodyWt	BrainWt	NonDreaming
Africanelephant	6654.000	5712.0	NaN
Africangiantpouchedrat	1.000	6.6	6.3
ArcticFox	3.385	44.5	NaN
Arcticgroundsquirrel	0.920	5.7	NaN
Asianelephant	2547.000	4603.0	2.1

```
import pandas as pd
file = 'sleep_in_mammals.txt'
d = pd.read_csv(file, comment='#', header=0, sep='\t',
                index_col=0)


d.columns
# Index(['BodyWt', 'BrainWt', 'NonDreaming', 'Dreaming',
'TotalSleep', 'LifeSpan', 'Gestation', 'Predation', 'Exposure',
'Danger'], dtype='object')



d.columns.values
# array(['BodyWt', 'BrainWt', 'NonDreaming', 'Dreaming',
'TotalSleep', 'LifeSpan', 'Gestation', 'Predation', 'Exposure',
'Danger'], dtype=object)
```

データフレーム

Pandas のデータフレームの列名および行名は、それぞれ columns および index で取得できる。



Species	BodyWt	BrainWt	NonDreaming
Africanelephant	6654.000	5712.0	NaN
Africangiantpouchedrat	1.000	6.6	6.3
ArcticFox	3.385	44.5	NaN
Arcticgroundsquirrel	0.920	5.7	NaN
Asianelephant	2547.000	4603.0	2.1

```
import pandas as pd
file = 'sleep_in_mammals.txt'
d = pd.read_csv(file, comment='#', header=0, sep='\t',
                index_col=0)

d.index
# Index(['Africanelephant', 'Africangiantpouchedrat',
'ArcticFox', 'Arcticgroundsquirrel', ... 'Vervet',
'Wateropossum', 'Yellow-belliedmarmot'], dtype='object',
name='Species')

d.index.values
# array(['Africanelephant', 'Africangiantpouchedrat',
'ArcticFox', 'Arcticgroundsquirrel', ... 'Vervet',
'Wateropossum', 'Yellow-belliedmarmot'], dtype=object)
```

データフレーム

Pandas のデータフレームから特定の列あるいは行のデータを取り出すとき iloc あるいは loc メソッドを使う。位置番号で要素を取り出すときは iloc を使う。名前で要素を取り出すときは loc を使う。

```
import pandas as pd
file = 'sleep_in_mammals.txt'
d = pd.read_csv(file, comment='#', header=0, sep='\t',
                index_col=0)
d.iloc[0:3, 0:5]
```

Species	BodyWt	BrainWt	NonDreaming	Dreaming	TotalSleep
Africanelephant	6654.000	5712.0	NaN	NaN	3.3
Africangiantpouchedrat	1.000	6.6	6.3	2.0	8.3
ArcticFox	3.385	44.5	NaN	NaN	12.5

```
d.iloc[0:3, :]
```

Species	BodyWt	BrainWt	NonDreaming	Dreaming	TotalSleep	LifeSpan	Gestation	Predation	Exposure	Danger
Africanelephant	6654.000	5712.0	NaN	NaN	3.3	38.6	645.0	3	5	3
Africangiantpouchedrat	1.000	6.6	6.3	2.0	8.3	4.5	42.0	3	1	3
ArcticFox	3.385	44.5	NaN	NaN	12.5	14.0	60.0	1	1	1

データフレーム

Pandas のデータフレームから特定の列あるいは行のデータを取り出すとき iloc あるいは loc メソッドを使う。位置番号で要素を取り出すときは iloc を使う。名前で要素を取り出すときは loc を使う。

```
import pandas as pd
file = 'sleep_in_mammals.txt'
d = pd.read_csv(file, comment='#', header=0, sep='\t',
                index_col=0)

species = ['Cat', 'Rat', 'Cow', 'Pig']
features = ['BodyWt', 'BrainWt', 'TotalSleep', 'LifeSpan']
d.loc[species, features]
```

Species	BodyWt	BrainWt	TotalSleep	LifeSpan
Cat	3.30	25.6	14.5	28.0
Rat	0.28	1.9	13.2	4.7
Cow	465.00	423.0	3.9	30.0
Pig	192.00	180.0	8.4	27.0

Pandas ファイルへの書き込み

Pandas では、データをCSV または TSV ファイルに書き出すとき、to_csv 関数（メソッド）を使う。この際に、区切り文字、行名の有無、列名の有無を指定することができる。また、欠損値を特定の文字に変換することもできる。

```
import pandas as pd
file = 'sleep_in_mammals.txt'
d = pd.read_csv(file, comment='#', header=0, sep='\t',
                index_col=0)


d.to_csv('o.txt', sep=',', header=False, index=False)
```

```
6654.0,5712.0,,,3.3,38.6,645.0,3,5,3
1.0,6.6,6.3,2.0,8.3,4.5,42.0,3,1,3
3.385,44.5,,,12.5,14.0,60.0,1,1,1
0.92,5.7,,,16.5,,25.0,5,2,3
2547.0,4603.0,2.1,1.8,3.9,69.0,624.0,3,5,4
10.55,179.5,9.1,0.7,9.8,27.0,180.0,4,4,4
0.023,0.3,15.8,3.9,19.7,19.0,35.0,1,1,1
160.0,169.0,5.2,1.0,6.2,30.4,392.0,4,5,4
3.3,25.6,10.9,3.6,14.5,28.0,63.0,1,2,1
52.16,440.0,8.3,1.4,9.7,50.0,230.0,1,1,1
```

Pandas ファイルへの書き込み

Pandas では、データをCSV または TSV ファイルに書き出すとき、to_csv 関数（メソッド）を使う。この際に、区切り文字、行名の有無、列名の有無を指定することができる。また、欠損値を特定の文字に変換することもできる。

```
import pandas as pd
file = 'sleep_in_mammals.txt'
d = pd.read_csv(file, comment='#', header=0, sep='\t',
                index_col=0)


d.to_csv('o.txt', sep=',', header=True, index=True)
```

```
Species,BodyWt,BrainWt,NonDreaming,Dreaming,TotalSleep,LifeSpan,Gestation,Predation,Exposure
Africanelephant,6654.0,5712.0,,,3.3,38.6,645.0,3,5,3
Africangiantpouchedrat,1.0,6.6,6.3,2.0,8.3,4.5,42.0,3,1,3
ArcticFox,3.385,44.5,,,12.5,14.0,60.0,1,1,1
Arcticgroundsquirrel,0.92,5.7,,,16.5,,25.0,5,2,3
Asianelephant,2547.0,4603.0,2.1,1.8,3.9,69.0,624.0,3,5,4
Baboon,10.55,179.5,9.1,0.7,9.8,27.0,180.0,4,4,4
Bigbrownbat,0.023,0.3,15.8,3.9,19.7,19.0,35.0,1,1,1
Braziliantapir,160.0,169.0,5.2,1.0,6.2,30.4,392.0,4,5,4
Cat,3.3,25.6,10.9,3.6,14.5,28.0,63.0,1,2,1
```

Pandas ファイルへの書き込み

Pandas では、データをCSV または TSV ファイルに書き出すとき、to_csv 関数（メソッド）を使う。この際に、区切り文字、行名の有無、列名の有無を指定することができる。また、欠損値を特定の文字に変換することもできる。

```
import pandas as pd
file = 'sleep_in_mammals.txt'
d = pd.read_csv(file, comment='#', header=0, sep='\t',
                index_col=0)


d.to_csv('o.txt', sep='\t', header=True, index=True)
```

Species	BodyWt	BrainWt	NonDreaming		Dreaming	TotalSleep	LifeSpan		Gestati
Africanelephant	6654.0	5712.0			3.3	38.6	645.0	3	5	3
Africangiantpouchedrat	1.0	6.6	6.3	2.0	8.3	4.5	42.0	3	1	
ArcticFox	3.385	44.5			12.5	14.0	60.0	1	1	1
Arcticgroundsquirrel		0.92	5.7			16.5		25.0	5	2
Asianelephant	2547.0	4603.0	2.1	1.8	3.9	69.0	624.0	3	5	4
Baboon	10.55	179.5	9.1	0.7	9.8	27.0	180.0	4	4	4
Bigbrownbat		0.023	0.3	15.8	3.9	19.7	19.0	35.0	1	1
Braziliantapir	160.0	169.0	5.2	1.0	6.2	30.4	392.0	4	5	4
Cat	3.3	25.6	10.9	3.6	14.5	28.0	63.0	1	2	1

Pandas ファイルへの書き込み

Pandas では、データをCSV または TSV ファイルに書き出すとき、to_csv 関数（メソッド）を使う。この際に、区切り文字、行名の有無、列名の有無を指定することができる。また、欠損値を特定の文字に変換することもできる。

```
import pandas as pd
file = 'sleep_in_mammals.txt'
d = pd.read_csv(file, comment='#', header=0, sep='\t',
                index_col=0)

d.to_csv('o.txt', sep='\t', header=True, index=True,
         na_rep='NA')
```

Species	BodyWt	BrainWt	NonDreaming	Dreaming		TotalSleep		LifeSpan		Gestati
Africanelephant	6654.0	5712.0	NA	NA	3.3	38.6	645.0	3	5	3
Africangiantpouchedrat	1.0	6.6	6.3	2.0	8.3	4.5	42.0	3	1	
ArcticFox	3.385	44.5	NA	NA	12.5	14.0	60.0	1	1	
Arcticgroundsquirrel		0.92	5.7	NA	NA	16.5	NA	25.0	5	2
Asianelephant	2547.0	4603.0	2.1	1.8	3.9	69.0	624.0	3	5	4
Baboon	10.55	179.5	9.1	0.7	9.8	27.0	180.0	4	4	4
Bigbrownbat		0.023	0.3	15.8	3.9	19.7	19.0	35.0	1	1
Braziliantapir	160.0	169.0	5.2	1.0	6.2	30.4	392.0	4	5	4
Cat	3.3	25.6	10.9	3.6	14.5	28.0	63.0	1	2	1

空白がNAに置換される

可視化

- matplotlib 基本
- 基本グラフ
- プロット領域の分割

pandas

matplotlib をベースとしている視覚化ライブラリーである。pandas データ構造をグラフ化できる。ただし、複雑なグラフや細かい部分の調整が難しい。

ggplot

R の ggplot2 とほぼ同じような使い方で、ほぼ同じような仕上がりとなる。The grammar of graphics と呼ばれる文法に従って記述する必要がある。

matplotlib

matplotlib は初期から存在する視覚化ライブラリーである。使用者が多いために、情報量も多い。複雑なグラフや細かい調整などが可能である。

plotly

ウェブベースのインタラクティブなグラフを作成できる。matplotlib に比べて勉強しやすいと言われている。

seaborn

matplotlib をベースとしている。matplotlib を補完する位置付けである。最近にリリースされたライブラリーである。複雑なグラフも簡単に作成できる。

Bokeh

ウェブベースのインタラクティブなグラフを作成できる。The grammar of graphics と呼ばれる文法に従って記述する必要がある。

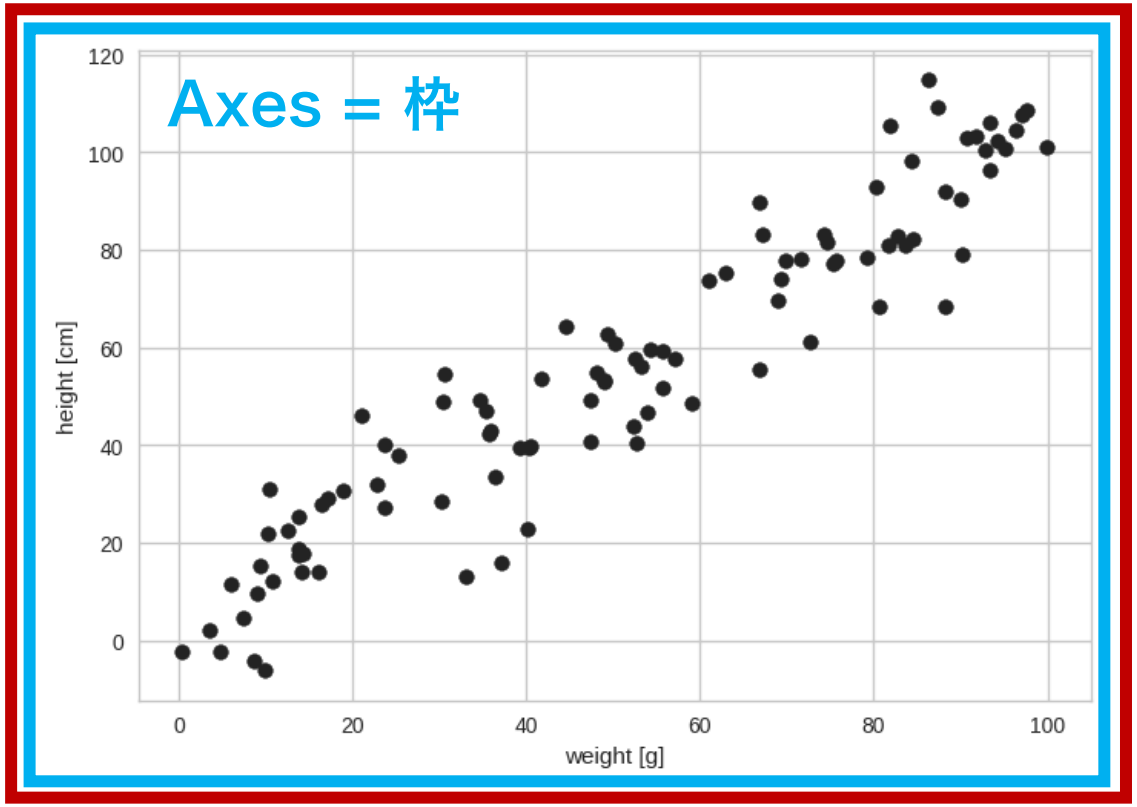
可視化

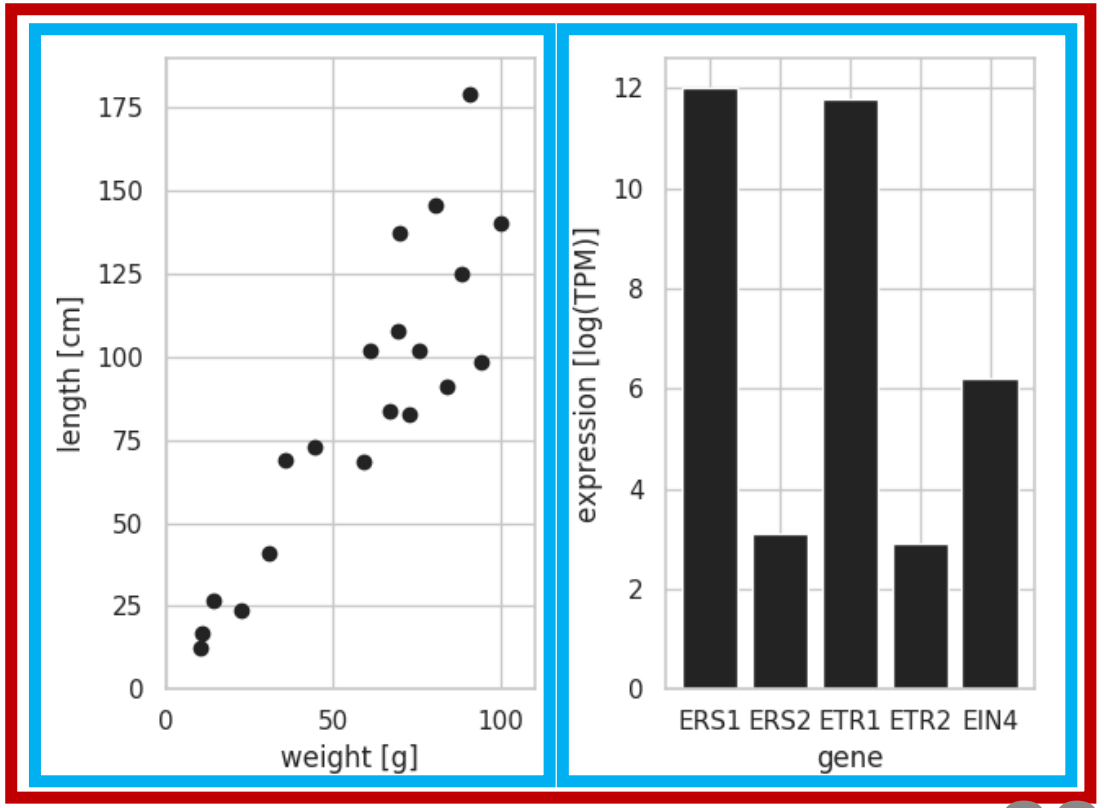
- matplotlib 基本
- 基本グラフ
- プロット領域の分割

matplotlib グラフ作成

Pyplot = 落書き帳

Figure = ページ

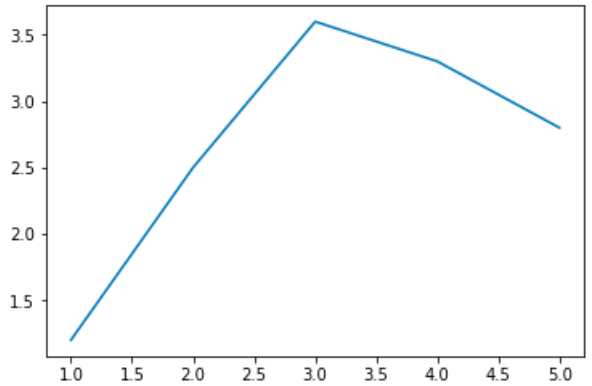




matplotlib グラフ作成

matplotlib を利用したグラフ作成は、pyplot モジュール中のメソッドを使用する。

```
import numpy as np
import matplotlib.pyplot as plt
x = np.array([1.0, 2.0, 3.0, 4.0, 5.0])
y = np.array([1.2, 2.5, 3.4, 3.3, 2.8])
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.plot(x, y)
fig.show()
```



matplotlib グラフ作成

matplotlib を利用したグラフ作成は、pyplot モジュール中のメソッドを使用する。

1. matplotlib の機能を呼び出す。pyplot 描画領域が用意される。

```
import numpy as np
import matplotlib.pyplot as plt
x = np.array([1.0, 2.0, 3.0, 4.0, 5.0])
y = np.array([1.2, 2.5, 3.4, 3.3, 2.8])
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.plot(x, y)
fig.show()
```

matplotlib グラフ作成

matplotlib を利用したグラフ作成は、pyplot モジュール中のメソッドを使用する。

1. matplotlib の機能を呼び出す。pyplot 描画領域が用意される。
2. Figure クラスのオブジェクトを用意する。

```
import numpy as np
import matplotlib.pyplot as plt
x = np.array([1.0, 2.0, 3.0, 4.0, 5.0])
y = np.array([1.2, 2.5, 3.4, 3.3, 2.8])
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.plot(x, y)
fig.show()
```

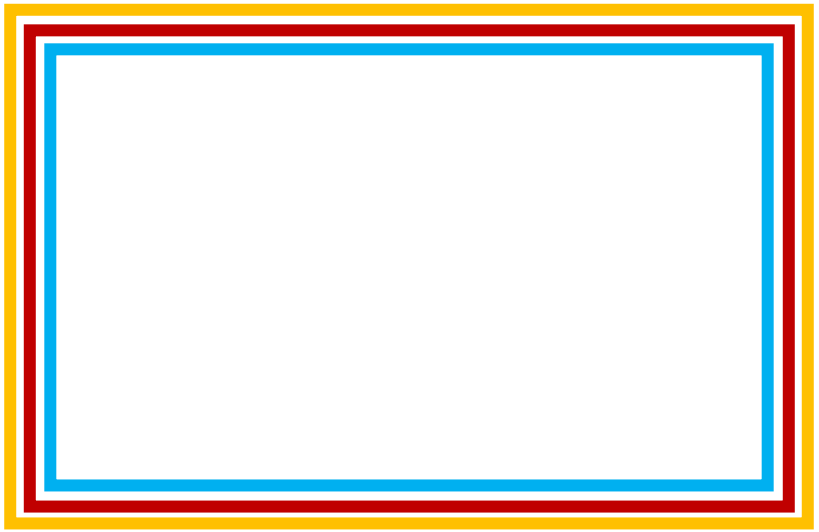
matplotlib グラフ作成

matplotlib を利用したグラフ作成は、pyplot モジュール中のメソッドを使用する。

1. matplotlib の機能を呼び出す。pyplot 描画領域が用意される。
2. Figure クラスのオブジェクトを用意する。
3. Figure 領域を1行1列に分割し、分割領域の1番目の領域でAxesクラスのオブジェクトを作成する。

▶

```
import numpy as np
import matplotlib.pyplot as plt
x = np.array([1.0, 2.0, 3.0, 4.0, 5.0])
y = np.array([1.2, 2.5, 3.4, 3.3, 2.8])
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.plot(x, y)
fig.show()
```

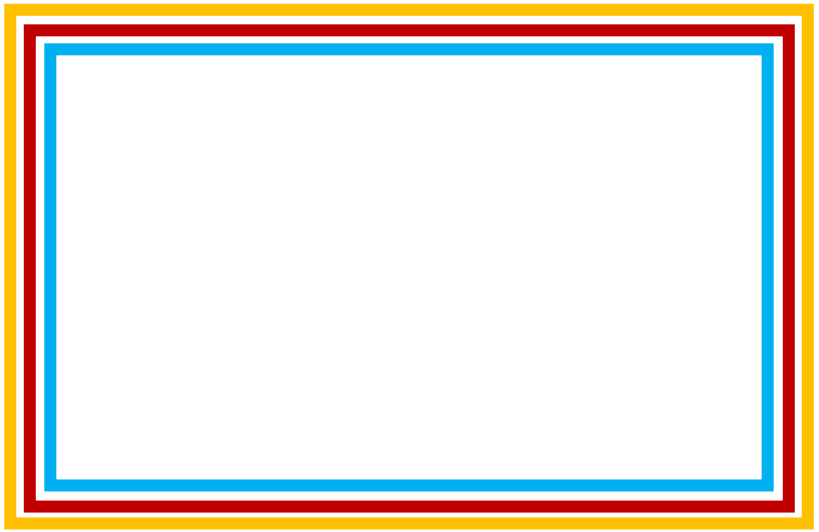


matplotlib グラフ作成

matplotlib を利用したグラフ作成は、pyplot モジュール中のメソッドを使用する。

1. matplotlib の機能を呼び出す。pyplot 描画領域が用意される。
2. Figure クラスのオブジェクトを用意する。
3. Figure 領域を1行1列に分割し、分割領域の1番目の領域でAxesクラスのオブジェクトを作成する。
4. 1番目のAxesオブジェクトに線グラフを描く。

```
import numpy as np
import matplotlib.pyplot as plt
x = np.array([1.0, 2.0, 3.0, 4.0, 5.0])
y = np.array([1.2, 2.5, 3.4, 3.3, 2.8])
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.plot(x, y)
fig.show()
```

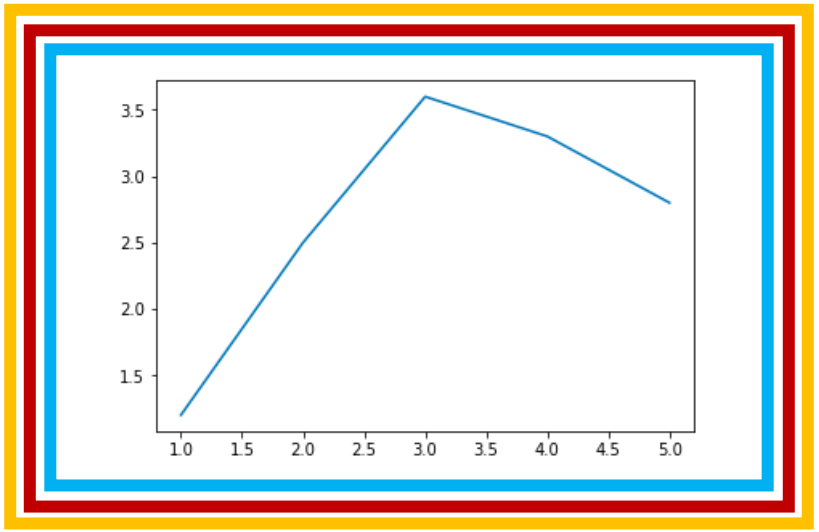


matplotlib グラフ作成

matplotlib を利用したグラフ作成は、pyplot モジュール中のメソッドを使用する。

1. matplotlib の機能を呼び出す。pyplot 描画領域が用意される。
2. Figure クラスのオブジェクトを用意する。
3. Figure 領域を1行1列に分割し、分割領域の1番目の領域でAxesクラスのオブジェクトを作成する。
4. 1番目のAxesオブジェクトに線グラフを描く。
5. これまでに描いたグラフをディスプレイ上に表示させる。

```
import numpy as np
import matplotlib.pyplot as plt
x = np.array([1.0, 2.0, 3.0, 4.0, 5.0])
y = np.array([1.2, 2.5, 3.4, 3.3, 2.8])
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.plot(x, y)
fig.show()
```

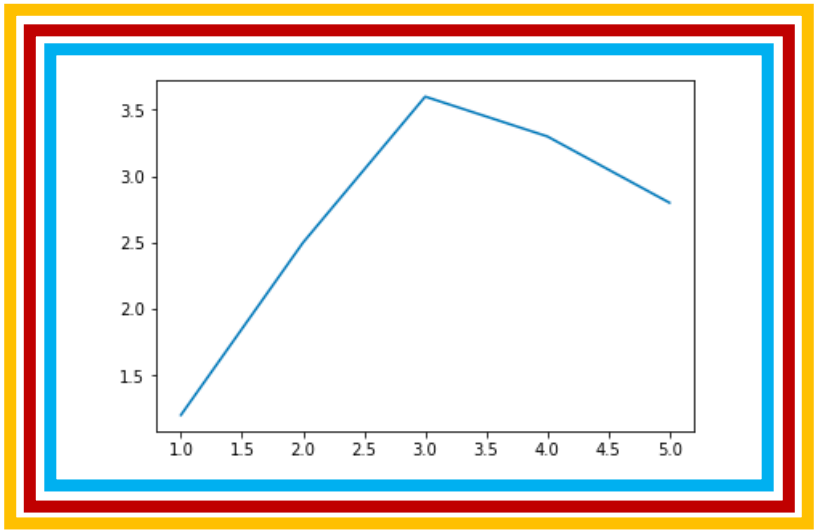


matplotlib グラフ保存

matplotlib を利用したグラフ作成は、pyplot モジュール中のメソッドを使用する。

1. matplotlib の機能を呼び出す。pyplot 描画領域が用意される。
2. Figure クラスのオブジェクトを用意する。
3. Figure 領域を1行1列に分割し、分割領域の1番目の領域でAxesクラスのオブジェクトを作成する。
4. 1番目のAxesオブジェクトに線グラフを描く。
5. これまでに描いたグラフをファイルにに保存する。

```
import numpy as np
import matplotlib.pyplot as plt
x = np.array([1.0, 2.0, 3.0, 4.0, 5.0])
y = np.array([1.2, 2.5, 3.4, 3.3, 2.8])
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.plot(x, y)
fig.savefig('fig1.png', format='png')
```



matplotlib グラフ保存

グラフを画像ファイルへ書き出すとき、show メソッドの代わりに savefig メソッドを使用する。画像ファイルのフォーマットは format オプションで指定する。PNG の他に PDF、PS、EPS、SVG を指定できる。

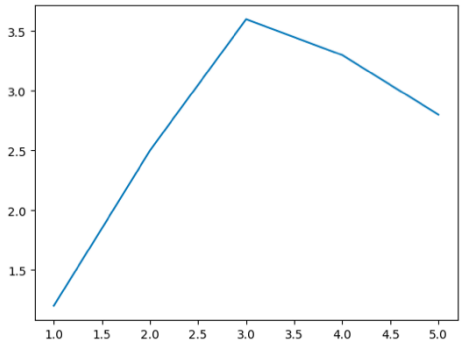
グラフのメモリ軸などに使う文字のフォントサイズなどを調整する場合は、set_xlabel などのメソッドを使う。また、画像ファイルのサイズや解像度などを調整したい場合は、描画デバイスを呼び出す際に行う。

```
import numpy as np
import matplotlib.pyplot as plt


x = np.array([1.0, 2.0, 3.0, 4.0, 5.0])
y = np.array([1.2, 2.5, 3.4, 3.3, 2.8])

fig = plt.figure(figsize=[8, 6], dpi=300)
ax = fig.add_subplot(1, 1, 1)
ax.plot(x, y)


ax.set_xlabel("xlabel", fontsize=18)
ax.set_ylabel("ylabel", fontsize=18)
ax.tick_params(labelsize=18)


fig.savefig('fig1.png', format='png')
```

可視化

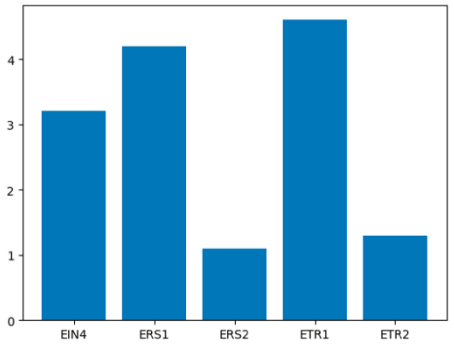
- matplotlib 基本
- 基本グラフ
- プロット領域の分割

基本グラフ



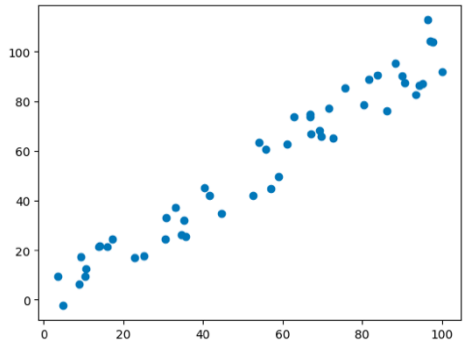
line chart

データの時系列的な変化傾向を視覚化するとき目的で使われる。



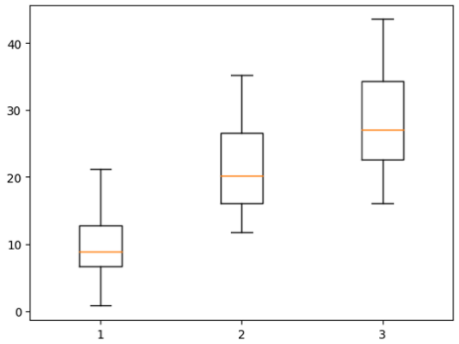
bar chart

カテゴリ毎に分類されるデータを視覚化する目的で使われる。



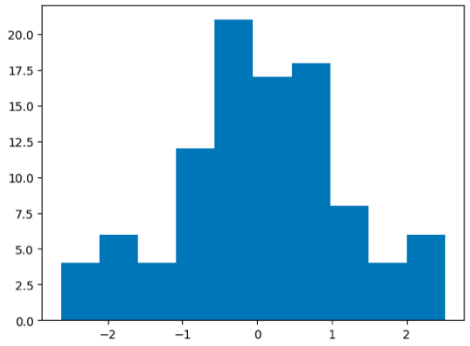
scatter chart

2 変量の連続値データ同士の相関や分布などを視覚する目的で使われる。



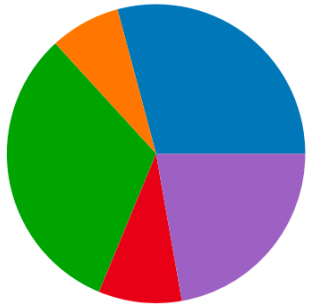
boxplot

カテゴリ毎に分類されるデータの分布などを視覚化する目的で使われる。



histogram

1 変量の連続値データの分布などを視覚化する目的で使われる。



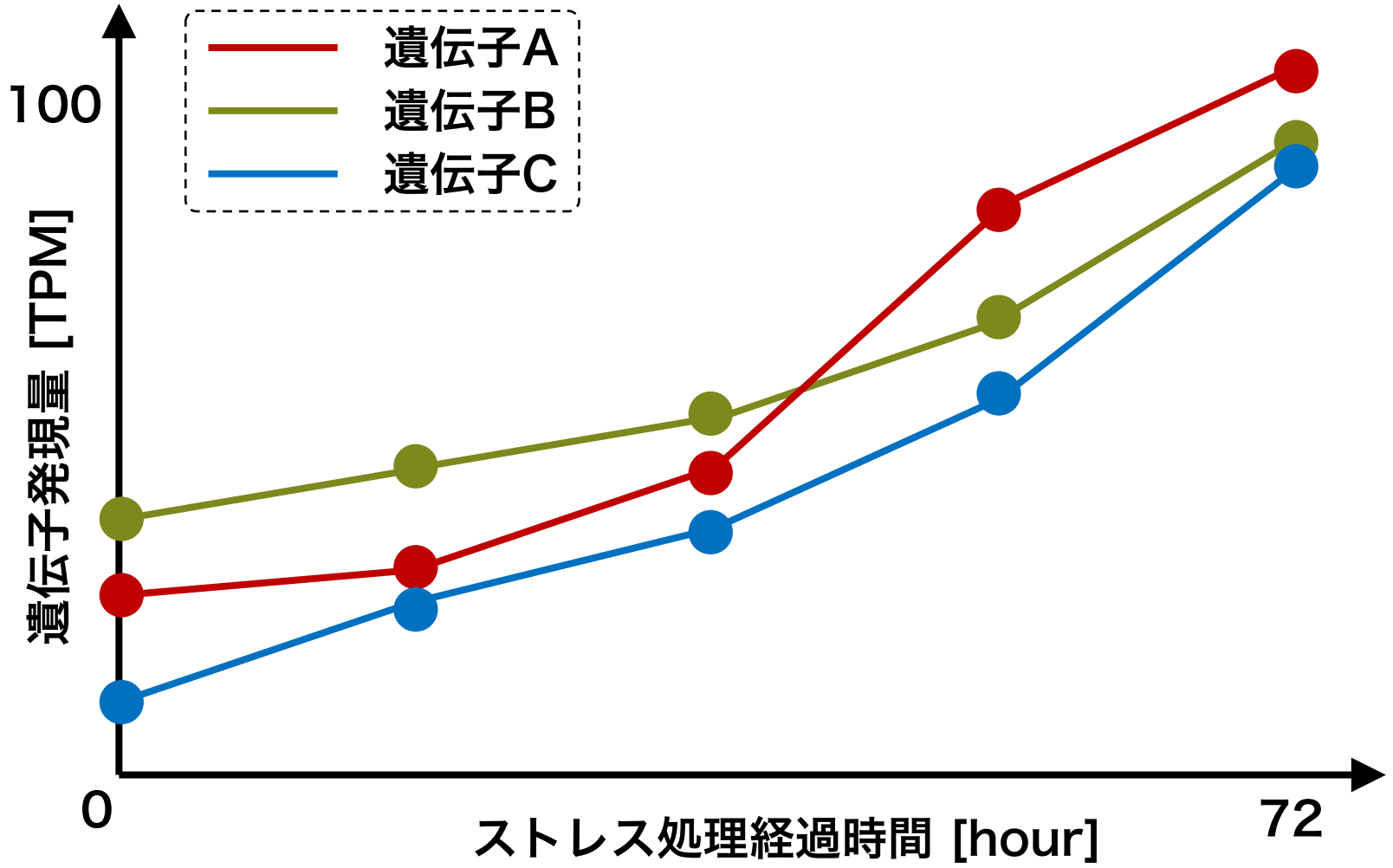
pie chart

割合データを視覚する目的で使われる。

基本グラフ

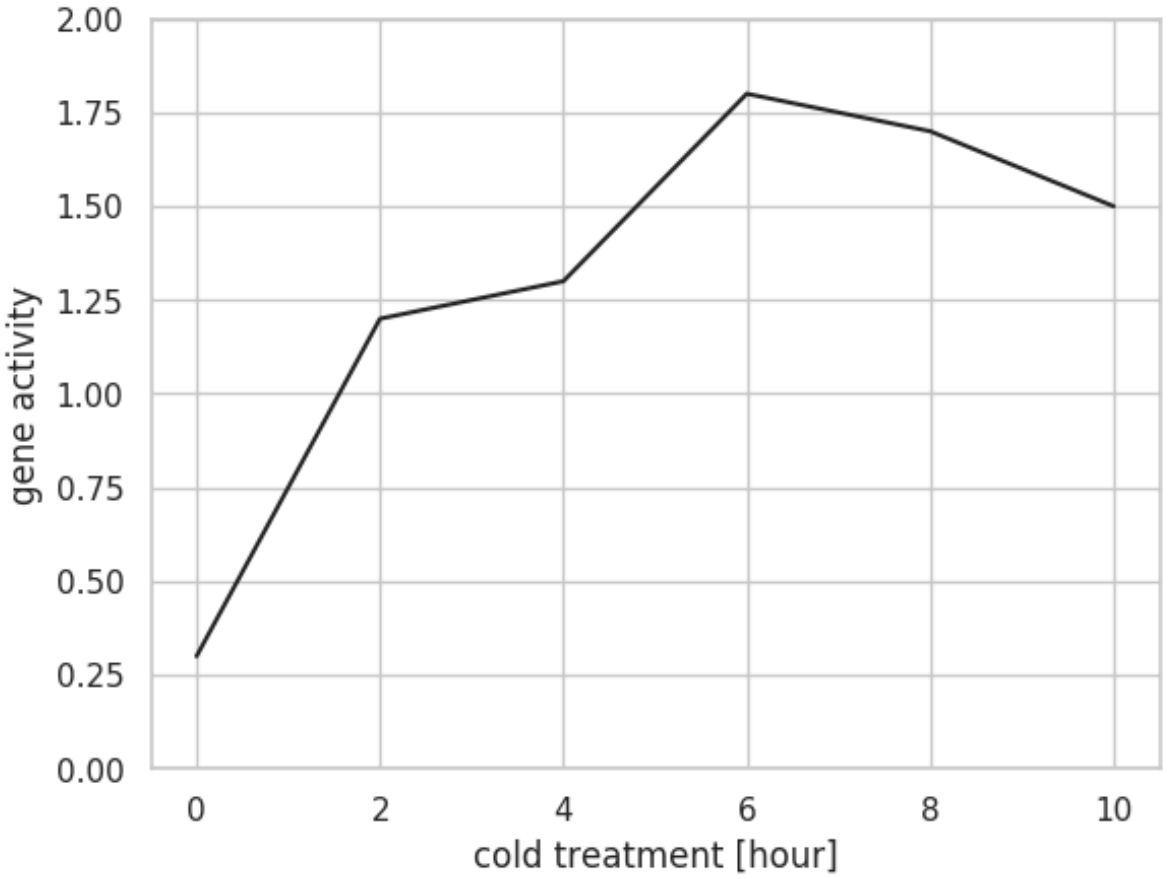
- 線グラフ
- 散布図
- 棒グラフ
- ヒストグラム
- ボックスプロット
- 円グラフ

線グラフ



- 線グラフは基本的にデータの系列的な変化を見るためのグラフ
- 縦軸および横軸は連続量
- 原点 (0, 0) が省略されているグラフを十分に注意すること
- 観測値を強調するために、観測値には点を明示することもある

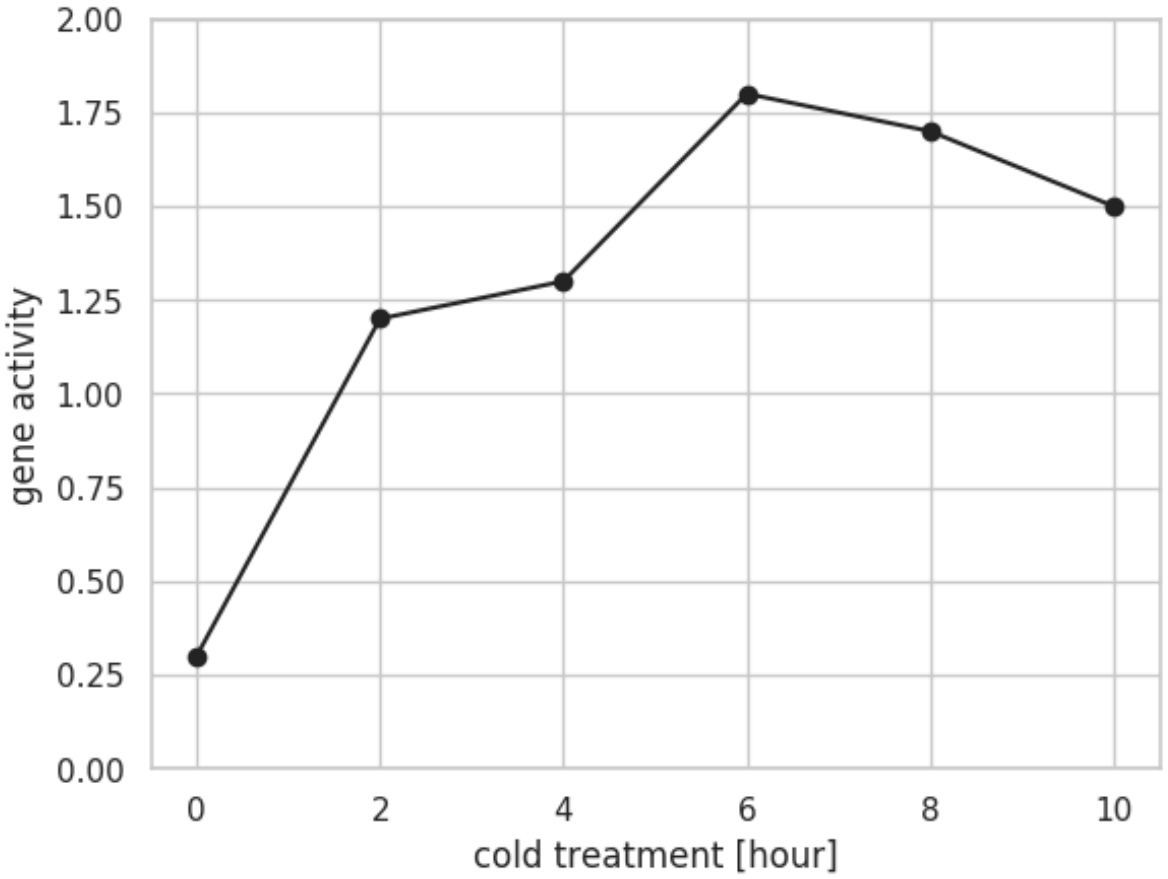
線グラフ




```
import matplotlib.pyplot as plt
import numpy as np

x = np.array([0, 2, 4, 6, 8, 10])
y = np.array([0.3, 1.2, 1.3, 1.8, 1.7, 1.5])

fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.plot(x, y)
ax.set_xlabel('cold treatment [hour]')
ax.set_ylabel('gene activity')
ax.set_ylim(0, 2)
fig.show()
```

set_xlabel、set_ylabel、set_ylim などの機能を、関数の名前と出力グラフと見比べながら推定してみよう。

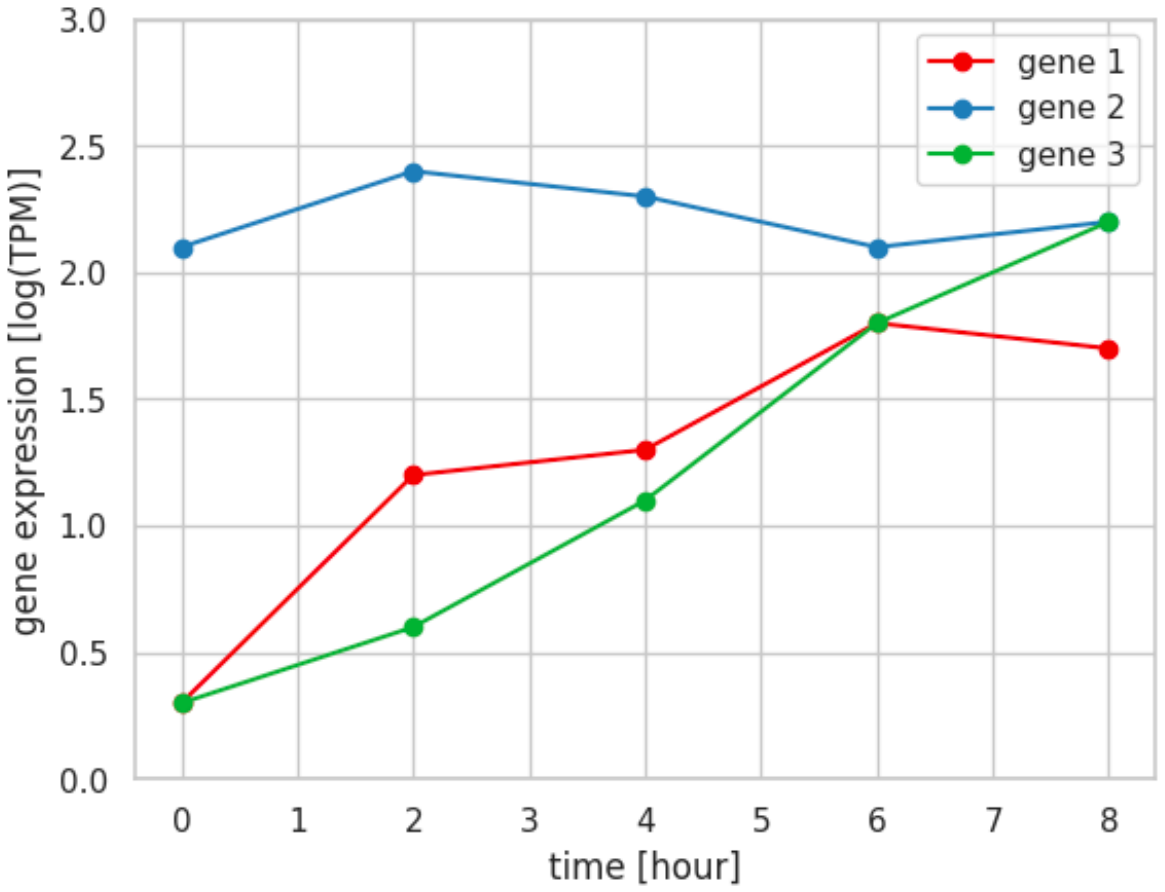
線グラフ




```
import matplotlib.pyplot as plt
import numpy as np

x = np.array([0, 2, 4, 6, 8, 10])
y = np.array([0.3, 1.2, 1.3, 1.8, 1.7, 1.5])

fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.plot(x, y, marker='o')
ax.set_xlabel('cold treatment [hour]')
ax.set_ylabel('gene activity')
ax.set_ylim(0, 2)
fig.show()
```

marker の値を変えることで、様々なマーカーがプロットされる。matplotlib で利用できるマーカー一覧は次のページで確認できる。 https://matplotlib.org/api/markers_api.html

線グラフ



plot メソッドを複数回使うことで、複数の線グラフを描くことができる。グラフを描く際に色が指定されていない場合は、自動的に配色される。

```
import matplotlib.pyplot as plt
import numpy as np
x = np.array([0, 2, 4, 6, 8])
gene_1 = np.array([0.3, 1.2, 1.3, 1.8, 1.7])
gene_2 = np.array([2.1, 2.4, 2.3, 2.1, 2.2])
gene_3 = np.array([0.3, 0.6, 1.1, 1.8, 2.2])
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.plot(x, gene_1, label='gene 1',
        marker='o')
ax.plot(x, gene_2, label='gene 2',
        marker='o')
ax.plot(x, gene_3, label='gene 3',
        marker='o')
ax.legend()
# ax.set_title('Gene expression')
ax.set_xlabel('time [hour]')
ax.set_ylabel('gene expression [log(TPM)]')
fig.show()
```

確認問題

直線 y = x のグラフを描け。たたし、x 軸の範囲を 0～10 とし、y 軸の範囲を 0～10 とする。

○ 解答例

確認問題 解答

直線 y = x のグラフを描け。たたし、x 軸の範囲を 0～10 とし、y 軸の範囲を 0～10 とする。

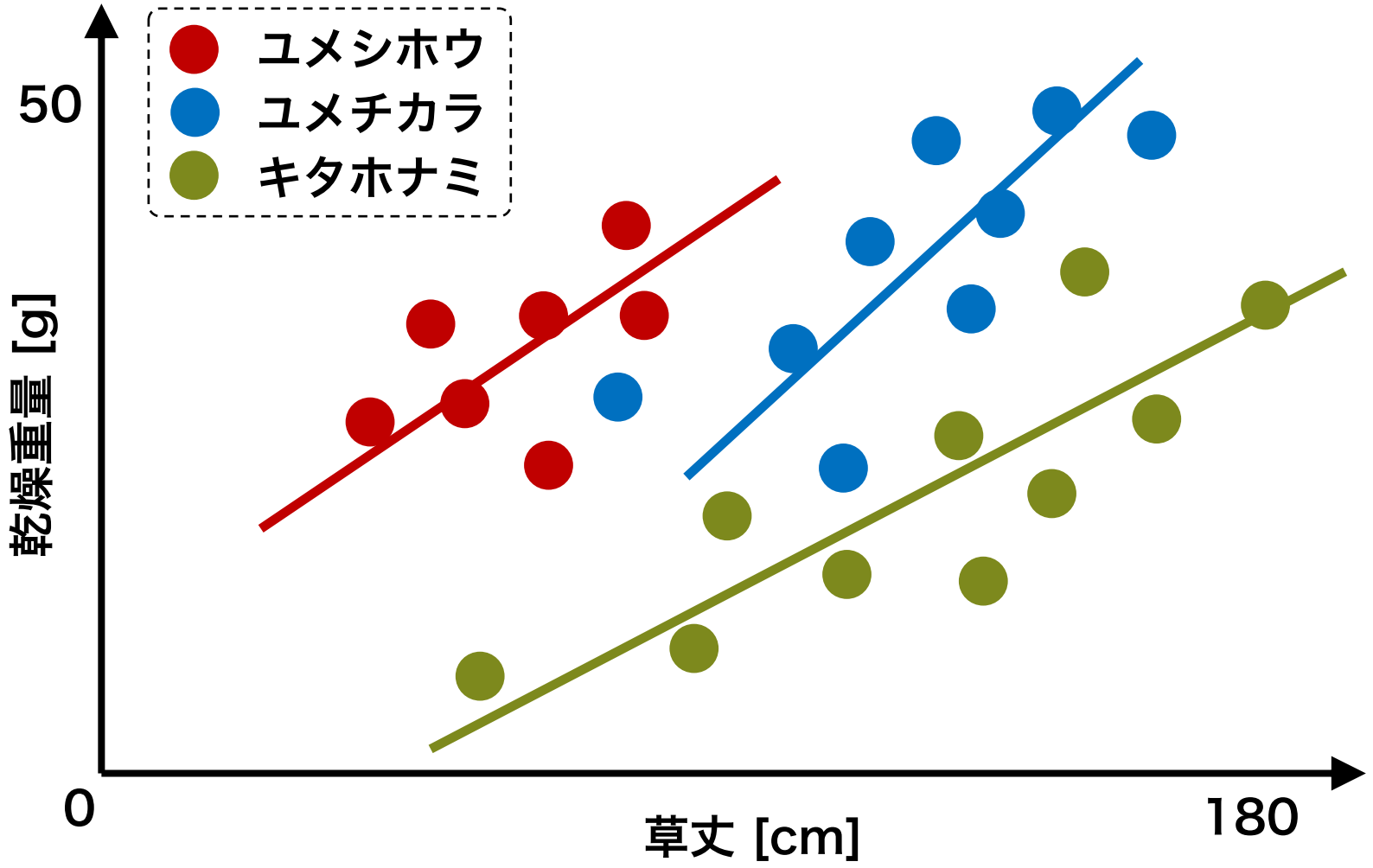
解答例


```
import matplotlib.pyplot as plt
import numpy as np

x = np.array([0, 100])
y = np.array([0, 100])

fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.plot(x, y)
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_xlim(0, 10)
ax.set_ylim(0, 10)
fig.show()
```

基本グラフ

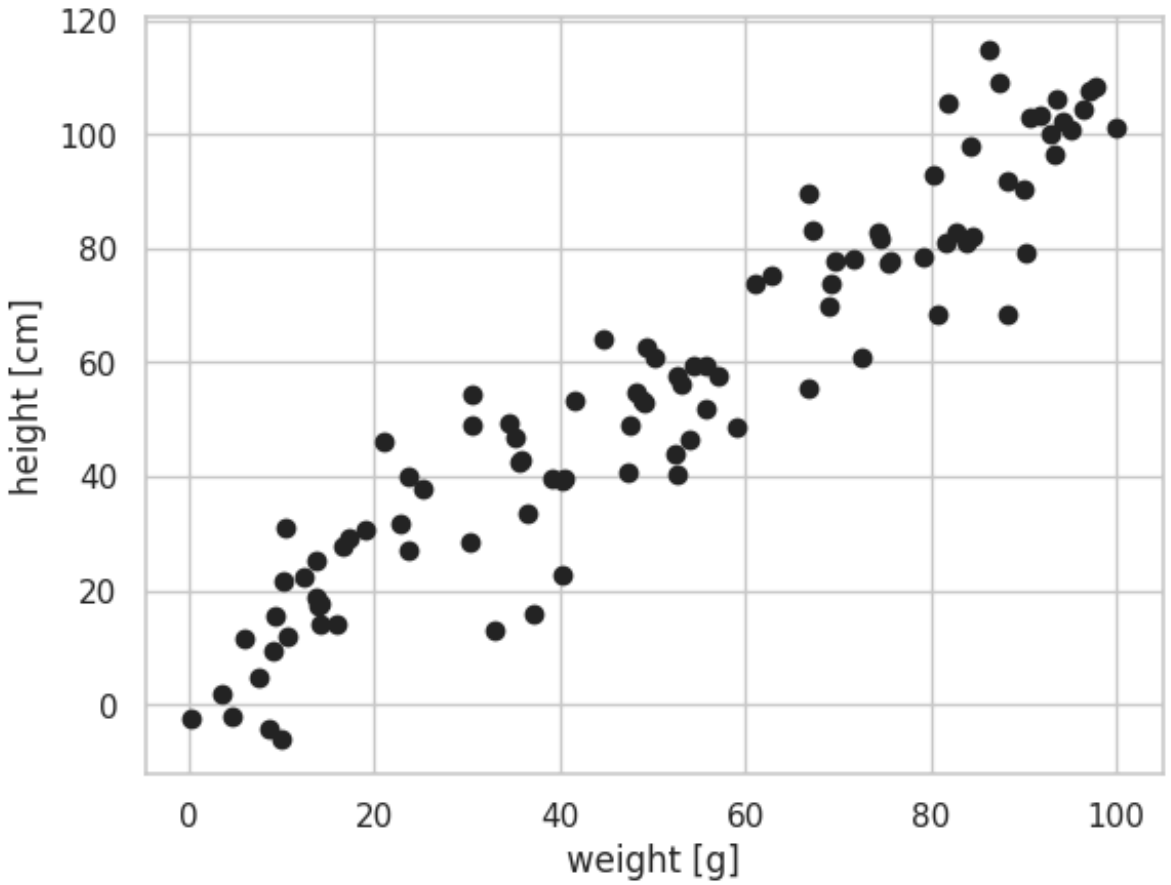
- 線グラフ
- 散布図
- 棒グラフ
- ヒストグラム
- ボックスプロット
- 円グラフ

散布図



- 連続量からなる多変量データ同士の相関や分布などを見るためのグラフ
- 縦軸および横軸は連続量
- 原点 (0, 0) が省略されているグラフを十分に注意すること
- 回帰直線との併用もよく見られる

散布図



plot 関数は線グラフ、scatter 関数は点グラフなどのように、関数を変えることで様々なグラフを描けるようになる。

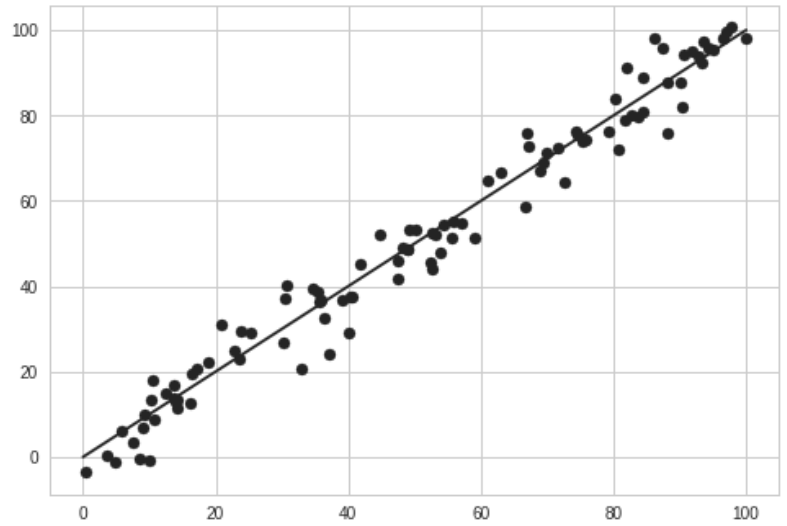
```
import matplotlib.pyplot as plt
import numpy as np

np.random.seed(2018)
x = np.random.uniform(0, 100, 100)
y = x + np.random.normal(5, 10, 100)

fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.scatter(x, y)
ax.set_xlabel('weight [g]')
ax.set_ylabel('height [cm]')

fig.show()
```

確認問題

次のように生成された乱数 x と y からなる散布図を描き、その上に直線 y = x のグラフを描け。

```
import matplotlib.pyplot as plt
import numpy as np

np.random.seed(2018)
x = np.random.uniform(0, 100, 100)
y = x + np.random.normal(0, 5, 100)
```

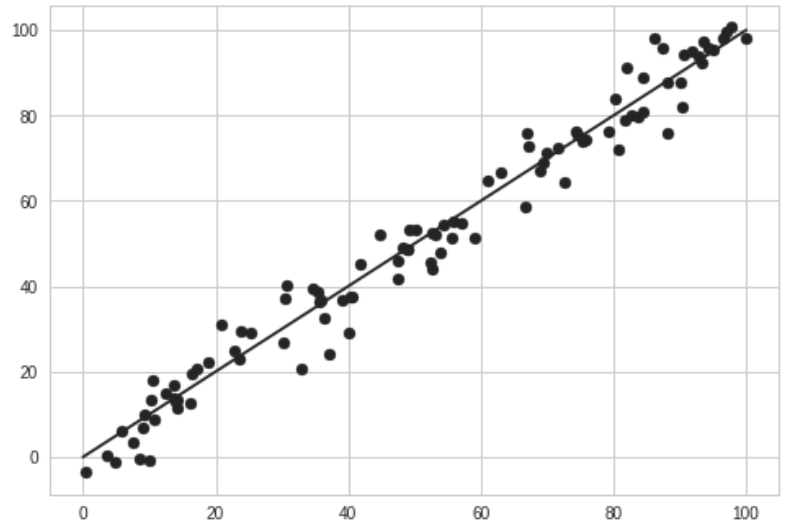
○ 解答例



確認問題 解答

次のように生成された乱数 x と y からなる散布図を描き、その上に直線 y = x のグラフを描け。

○ 解答例

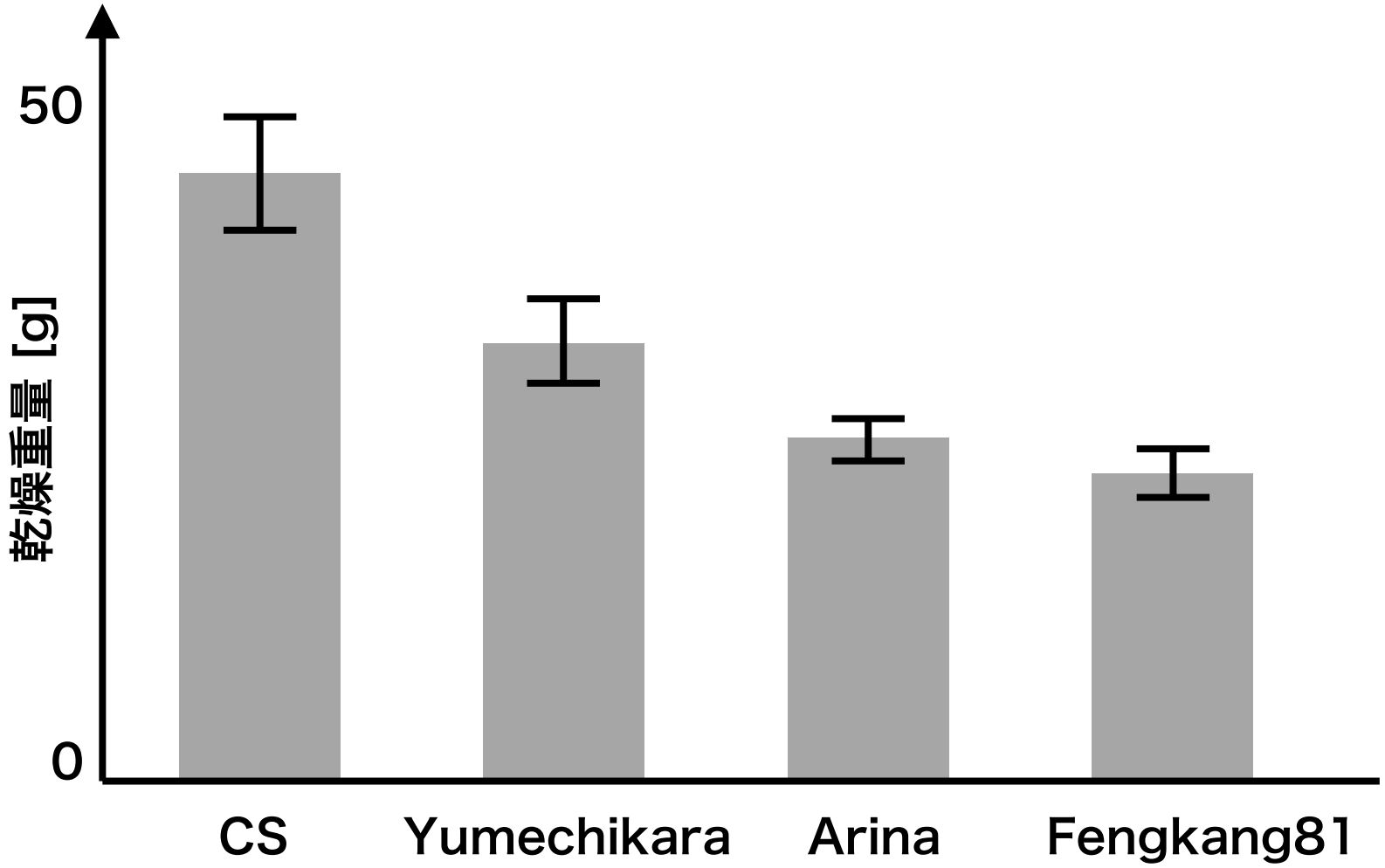



```
import matplotlib.pyplot as plt
import numpy as np

np.random.seed(2018)
x = np.random.uniform(0, 100, 100)
y = x + np.random.normal(0, 5, 100)

x2 = np.array([0, 100])
y2 = np.array([0, 100])

fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.scatter(x, y)
ax.plot(x2, y2)
fig.show()
```

基本グラフ

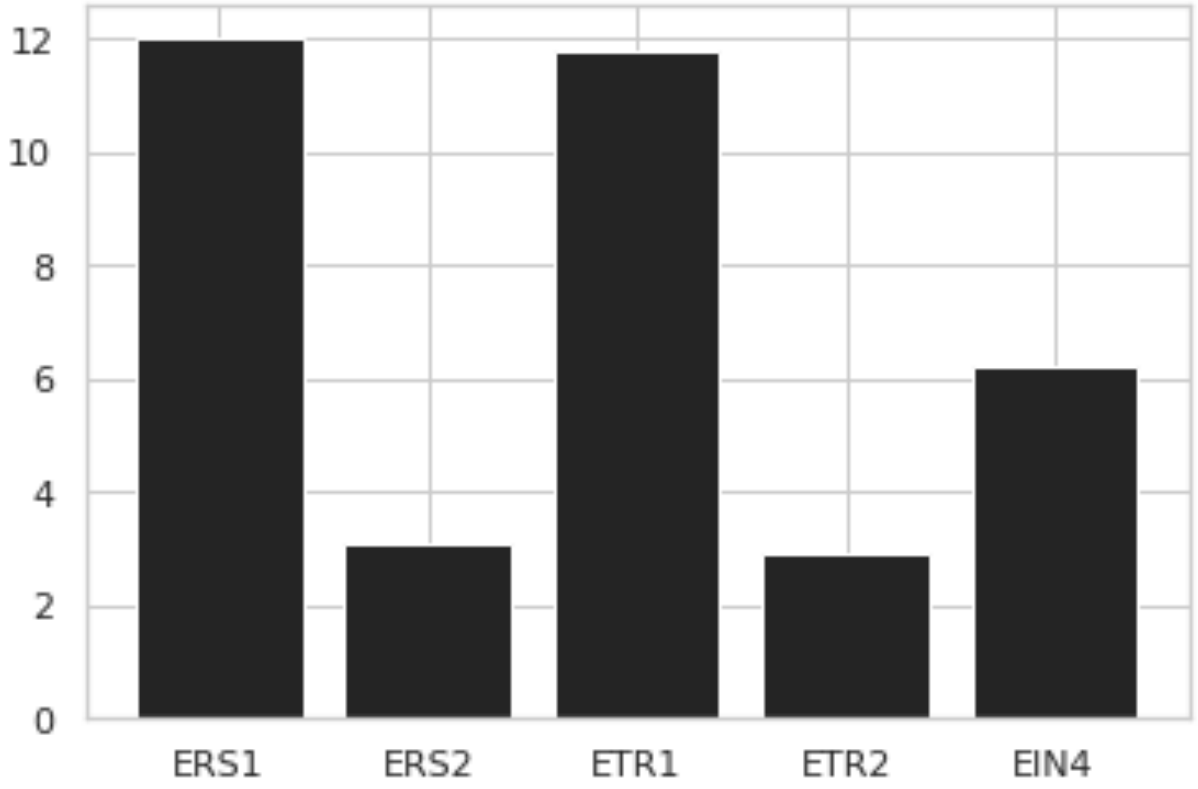
- 線グラフ
- 散布図
- 棒グラフ
- ヒストグラム
- ボックスプロット
- 円グラフ

棒グラフ



- 離散データを視覚化するためのグラフ
- 横軸が連続量、縦軸が離散量、あるいはその逆
- 原点が省略されているグラフを十分に注意すること
- エラーバーとともに用いられることがある

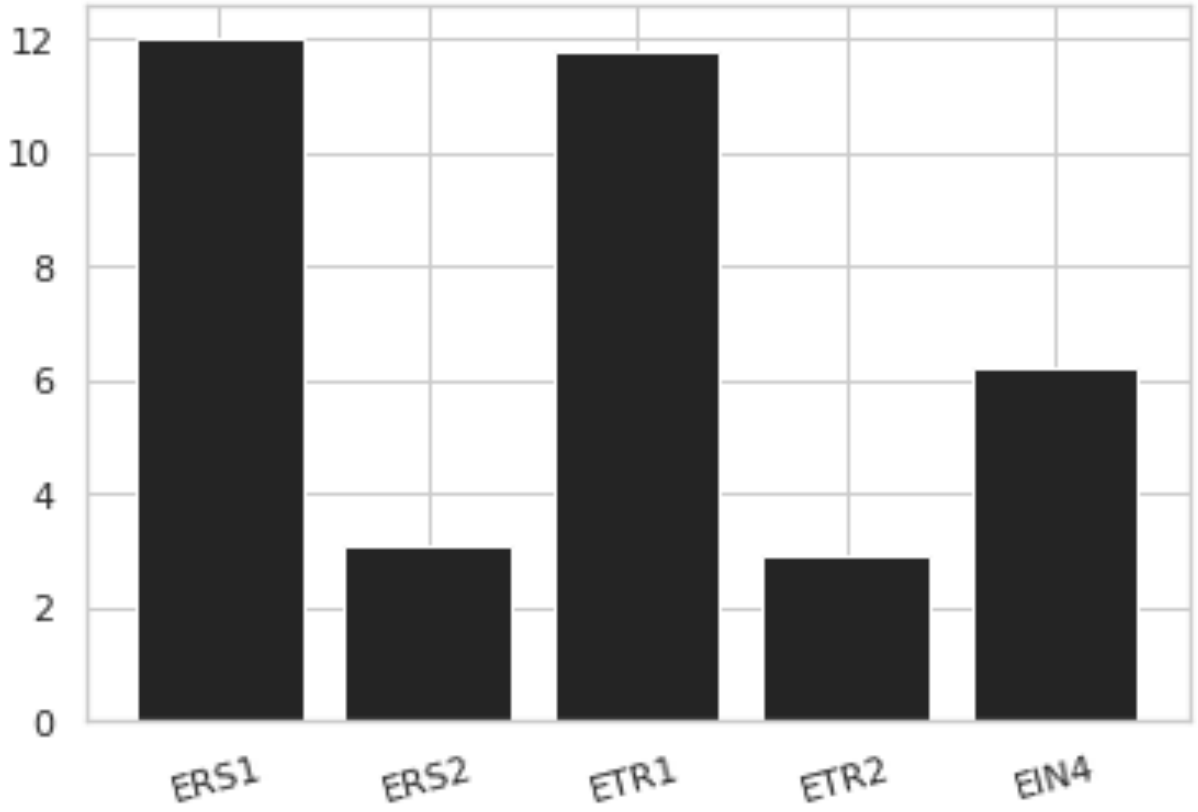
棒グラフ




```
import matplotlib.pyplot as plt
import numpy as np

x = np.array(['ERS1', 'ERS2', 'ETR1',
              'ETR2', 'EIN4'])
y = np.array([12.0, 3.1, 11.8, 2.9, 6.2])

fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.bar(x, y)

fig.show()
```

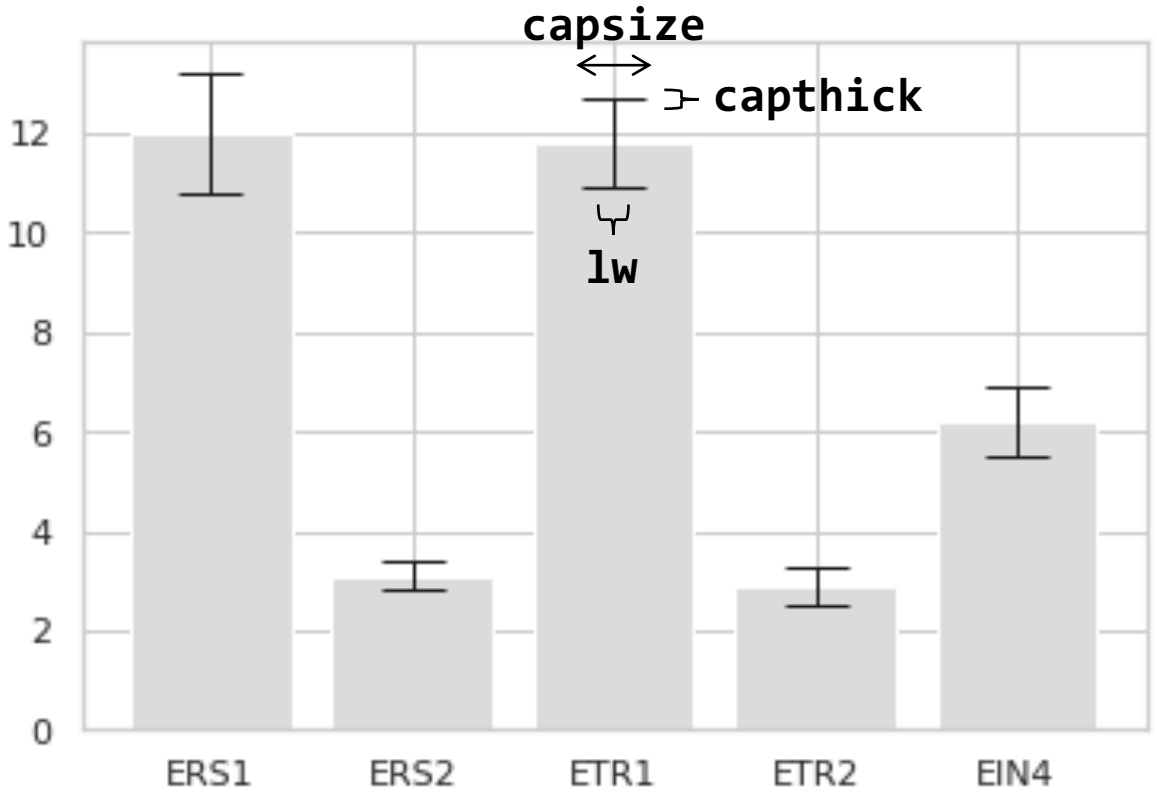
matplotlib のバージョンが古いと、横軸ラベルがアルファベット順に並べ替えられて描かれる場合がある。

棒グラフ




```
import matplotlib.pyplot as plt
import numpy as np

x = np.array(['ERS1', 'ERS2', 'ETR1',
              'ETR2', 'EIN4'])
y = np.array([12.0, 3.1, 11.8, 2.9, 6.2])

fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.bar(x, y)
ax.set_xticklabels(x, rotation=15)

fig.show()
```

棒グラフ



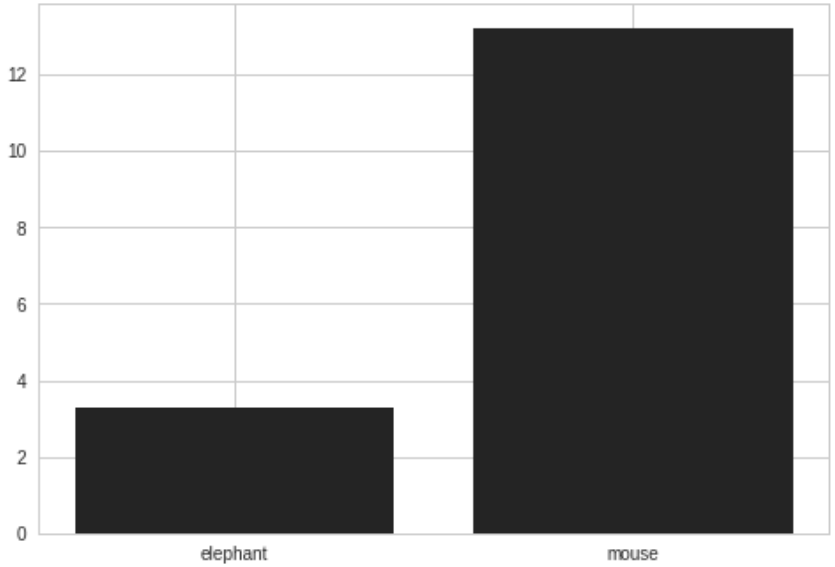

```
import matplotlib.pyplot as plt
import numpy as np


x = np.array(['ERS1', 'ERS2', 'ETR1',
              'ETR2', 'EIN4'])
y = np.array([12.0, 3.1, 11.8, 2.9, 6.2])
e = np.array([1.2, 0.3, 0.9, 0.4, 0.7])


fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
error_bar_set = dict(lw = 1, capthick = 1,
                     capsize = 10)
ax.bar(x, y, yerr = e,
       error_kw=error_bar_set)
fig.show()
```

確認問題

象の睡眠時間は3.3時間、ネズミの睡眠時間は13.2時間。両者を比較するための棒グラフを描け。

```
import matplotlib.pyplot as plt
import numpy as np

x = ['elephant', 'mouse']
y = [3.3, 13.2]
```

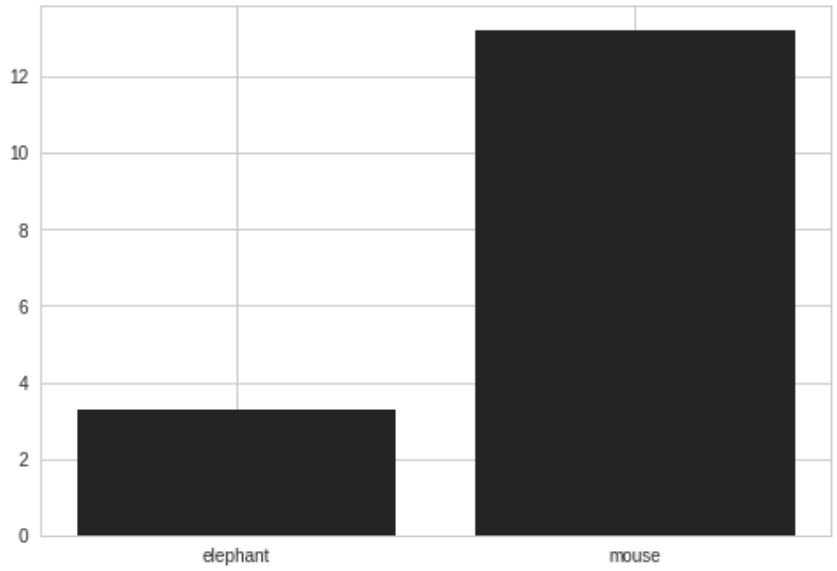
○ 解答例



確認問題 解答

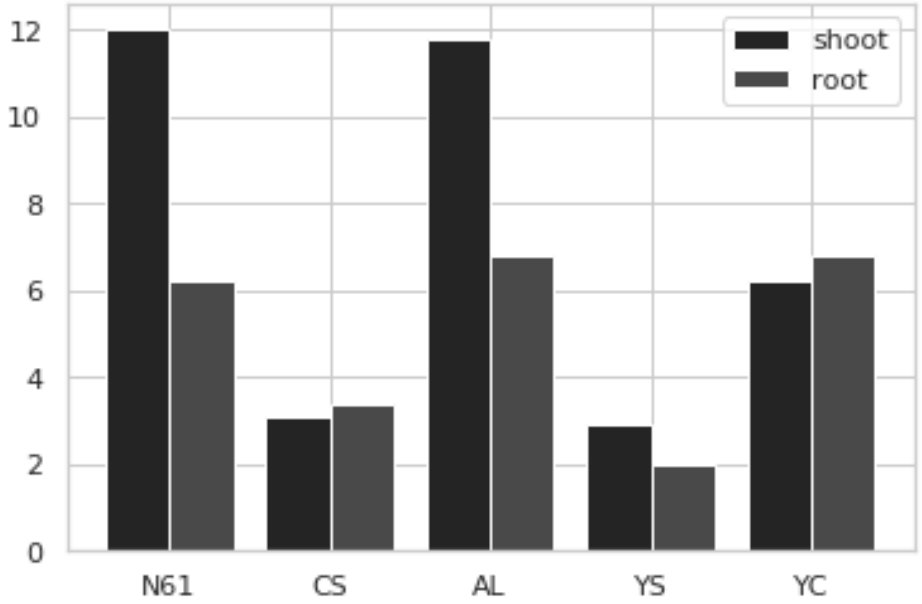
象の睡眠時間は3.3時間、ネズミの睡眠時間は13.2時間。両者を比較するための棒グラフを描け。

○ 解答例




```
import matplotlib.pyplot as plt
import numpy as np

x = ['elephant', 'mouse']
y = [3.3, 13.2]

fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.bar(x, y)

fig.show()
```

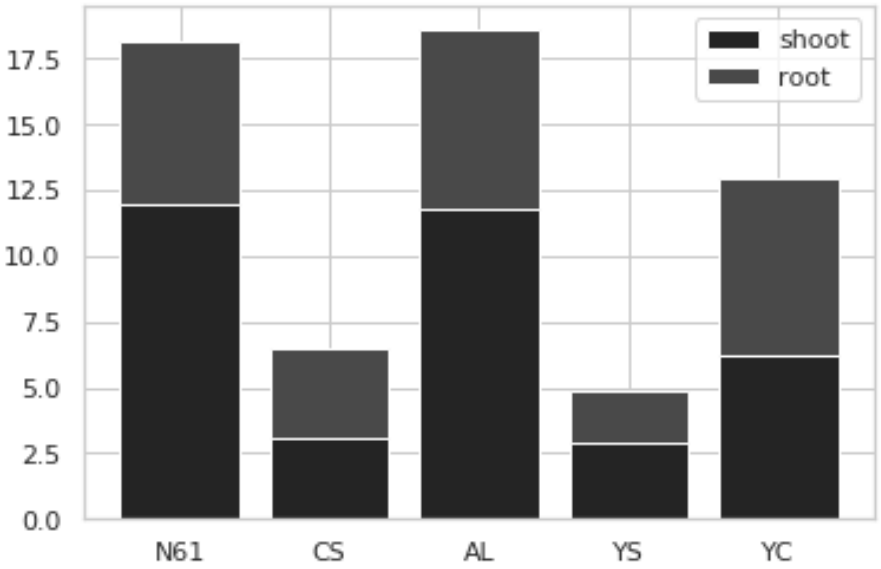
棒グラフ



横並びの棒グラフを描くとき、棒の x 座標および棒の幅をあらかじめ指定する必要がある。

```
import matplotlib.pyplot as plt
import numpy as np

xlabel = np.array(['N61', 'CS', 'AL', 'YS', 'YC'])
x = np.arange(len(xlabel))
y_shoot = np.array([12.0, 3.1, 11.8, 2.9, 6.2])
y_root = np.array([6.2, 3.4, 6.8, 2.0, 6.8])
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.bar(x - 0.2, y_shoot, width=0.4, label='shoot')
ax.bar(x + 0.2, y_root, width=0.4, label='root')
ax.legend()
ax.set_xticks(x)
ax.set_xticklabels(xlabel)
fig.show()
```

棒グラフ



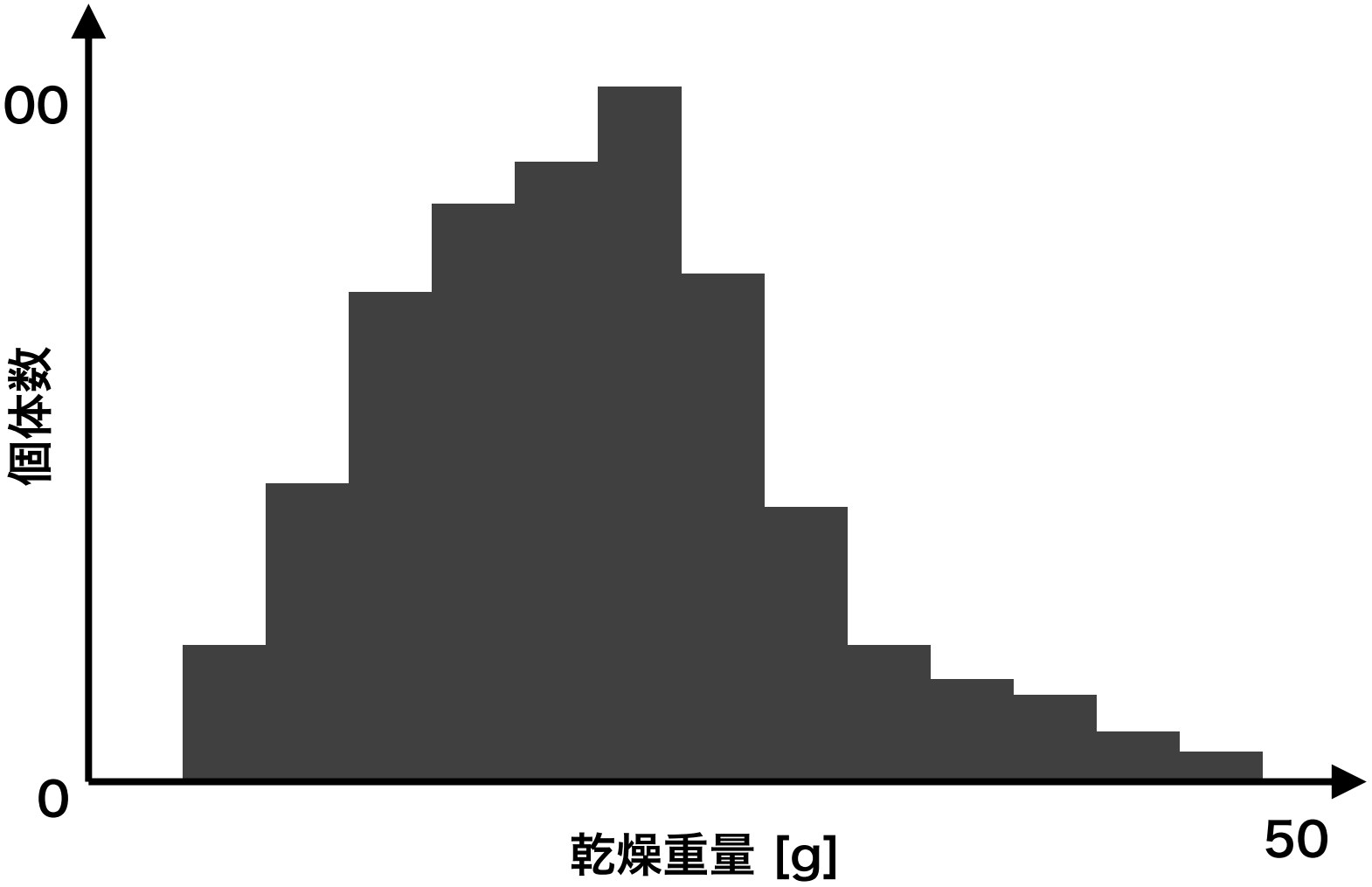
積み重ね棒グラフの場合は、上の方の棒グラフを描くとき、そのスタート地点を調整する必要がある。

```
import matplotlib.pyplot as plt
import numpy as np

xlabel = np.array(['N61', 'CS', 'AL', 'YS', 'YC'])
x = np.arange(len(xlabel))
y_shoot = np.array([12.0, 3.1, 11.8, 2.9, 6.2])
y_root = np.array([6.2, 3.4, 6.8, 2.0, 6.8])
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.bar(x, y_shoot, label='shoot')
ax.bar(x, y_root, label='root', bottom=y_shoot)
ax.legend()
ax.set_xticks(x)
ax.set_xticklabels(xlabel)
fig.show()
```

基本グラフ

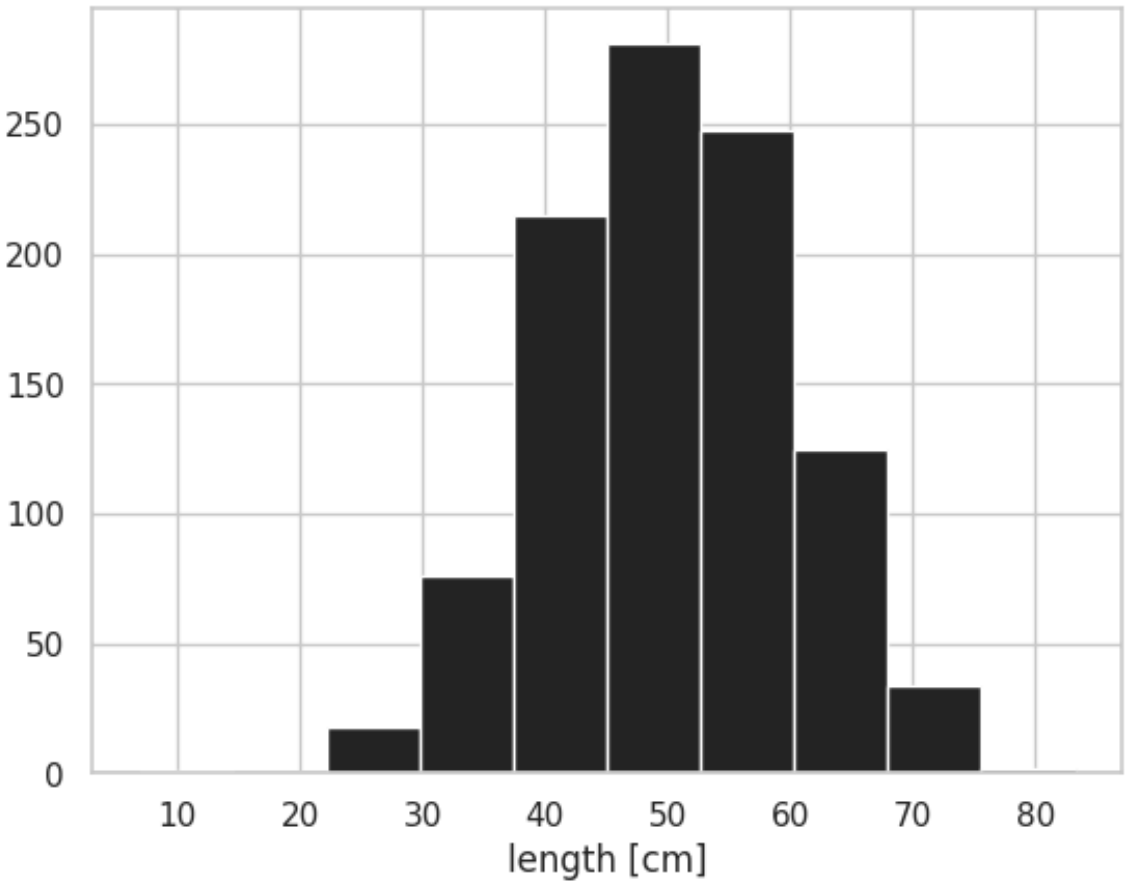
- 線グラフ
- 散布図
- 棒グラフ
- ヒストグラム
- ボックスプロット
- 円グラフ

ヒストグラム



- 1変量の連続値データを視覚化するためのグラフ
- 横軸が連続量であり、縦軸は頻度・個数または確率である
- 横幅は恣意的（経験的）に決められることもあれば、スタージェスの公式などで決めることもある

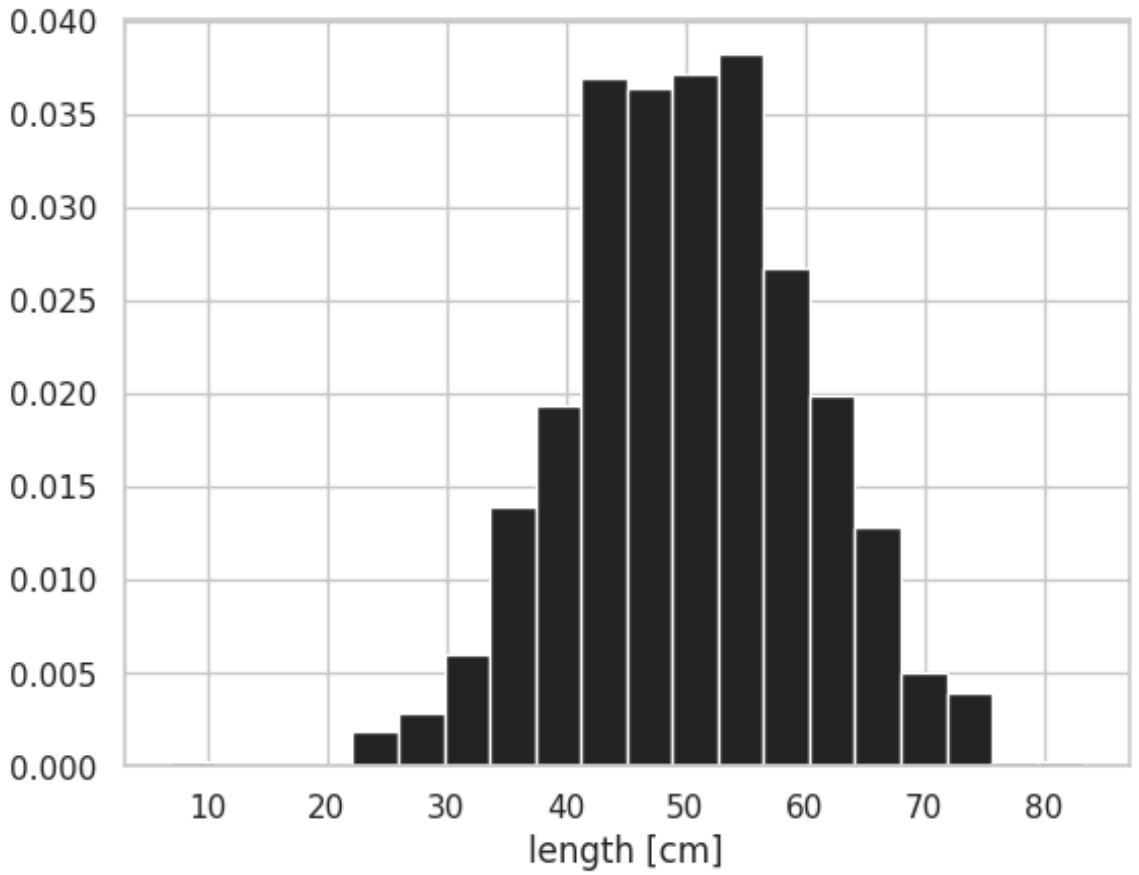
ヒストグラム



matplotlib デフォルトでは、ヒストグラムの横幅はスタージェスまたはフリードマン・ダイアコニスの公式により算出される。

```
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(2018)

x = np.random.normal(50, 10, 1000)

fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.hist(x)

ax.set_xlabel('length [cm]')

fig.show()
```

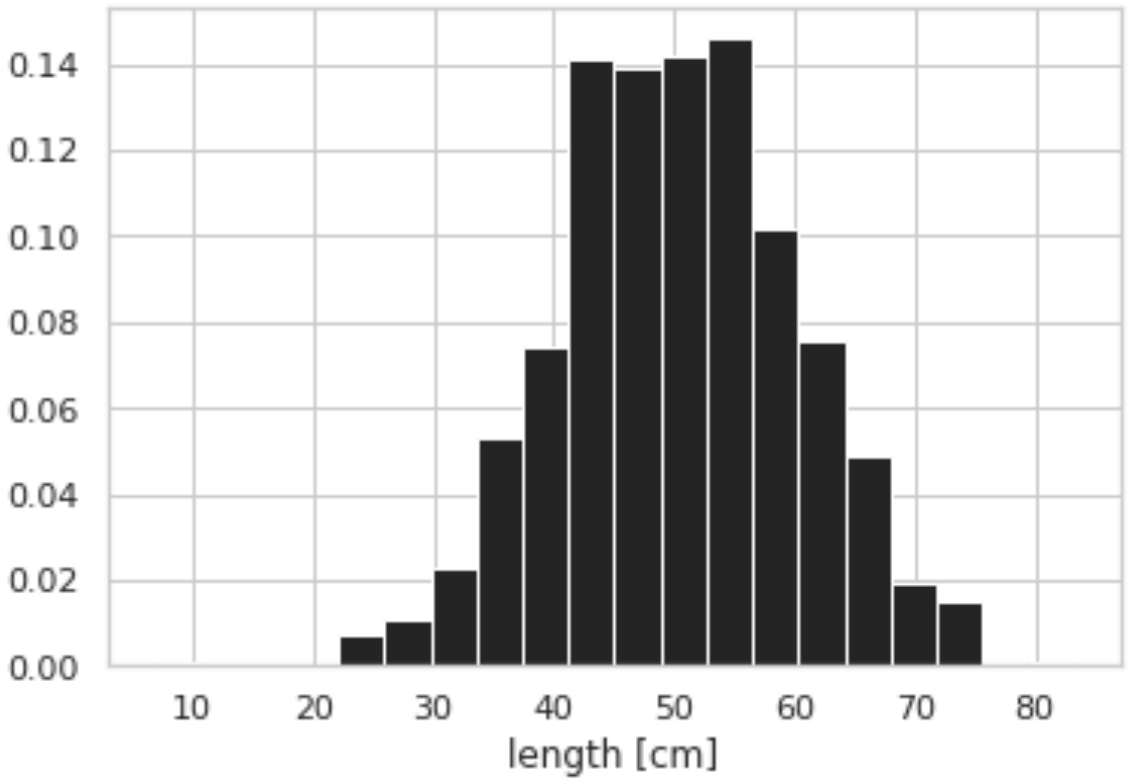
ヒストグラム



density=True を指定したとき、すべてのビンの面積を足すと 1 になる。棒の高さを足して 1 になるわけではないことに注意。

```
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(2018)

x = np.random.normal(50, 10, 1000)

fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.hist(x, bins=20, density=True)

ax.set_xlabel('length [cm]')

fig.show()
```

ヒストグラム



ビンの高さの合計値が 1 となるように、データに重みをおかけてヒストグラム描く。

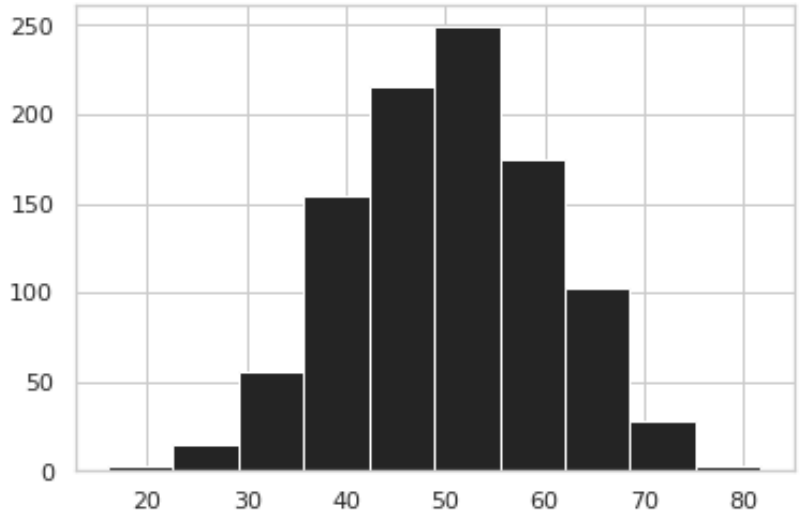
```
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(2018)

x = np.random.normal(50, 10, 1000)
w = np.ones_like(x)/float(len(x))

fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.hist(x, bins=20, weights=w)
ax.set_xlabel('length [cm]')

fig.show()
```

確認問題

平均 0、分散 1 の正規分布に従う乱数を 100 個生成し、そのヒストグラムを描け。

```
import numpy as np
import matplotlib.pyplot as plt


x = np.random.normal(50, 10, 1000)
```

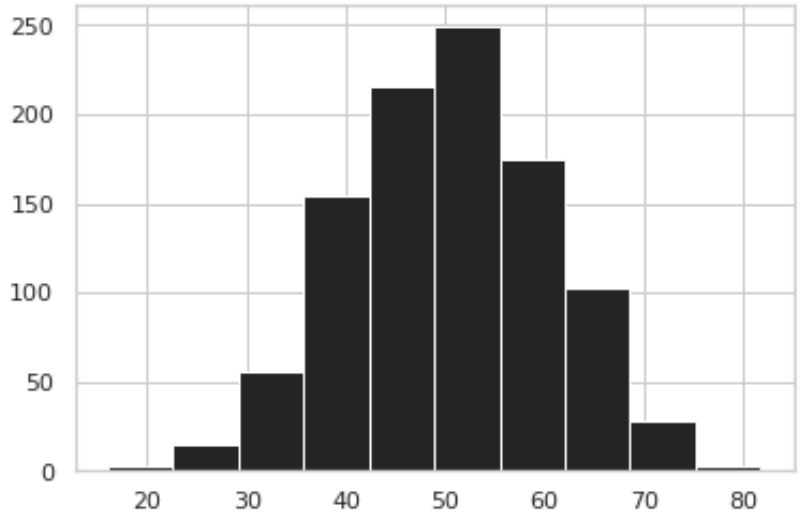
○ 解答例



確認問題 解答

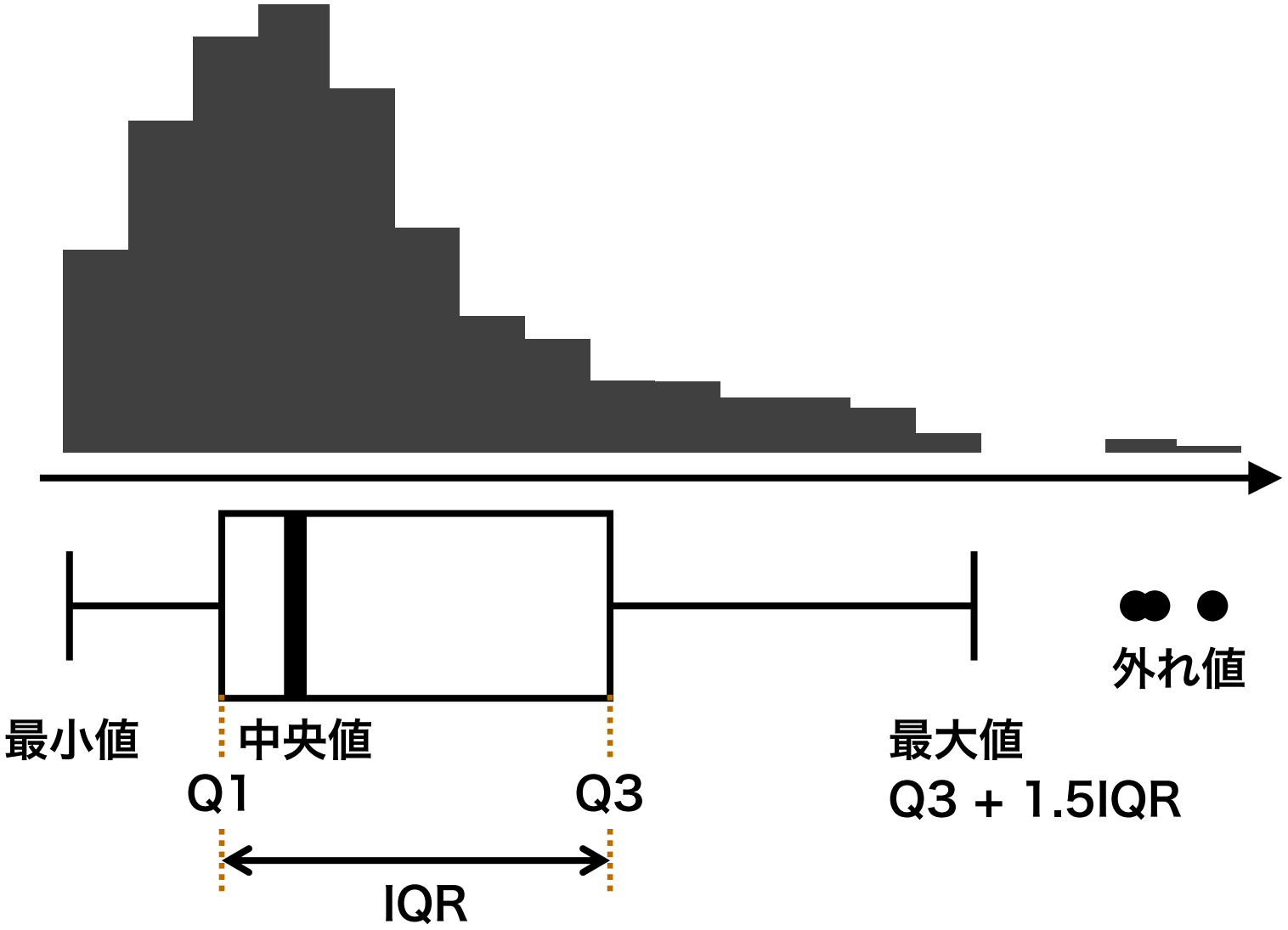
平均 0、分散 1 の正規分布に従う乱数を 100 個生成し、そのヒストグラムを描け。

○ 解答例




```
import numpy as np
import matplotlib.pyplot as plt

x = np.random.normal(50, 10, 1000)

fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.hist(x)

fig.show()
```

ボックスプロット

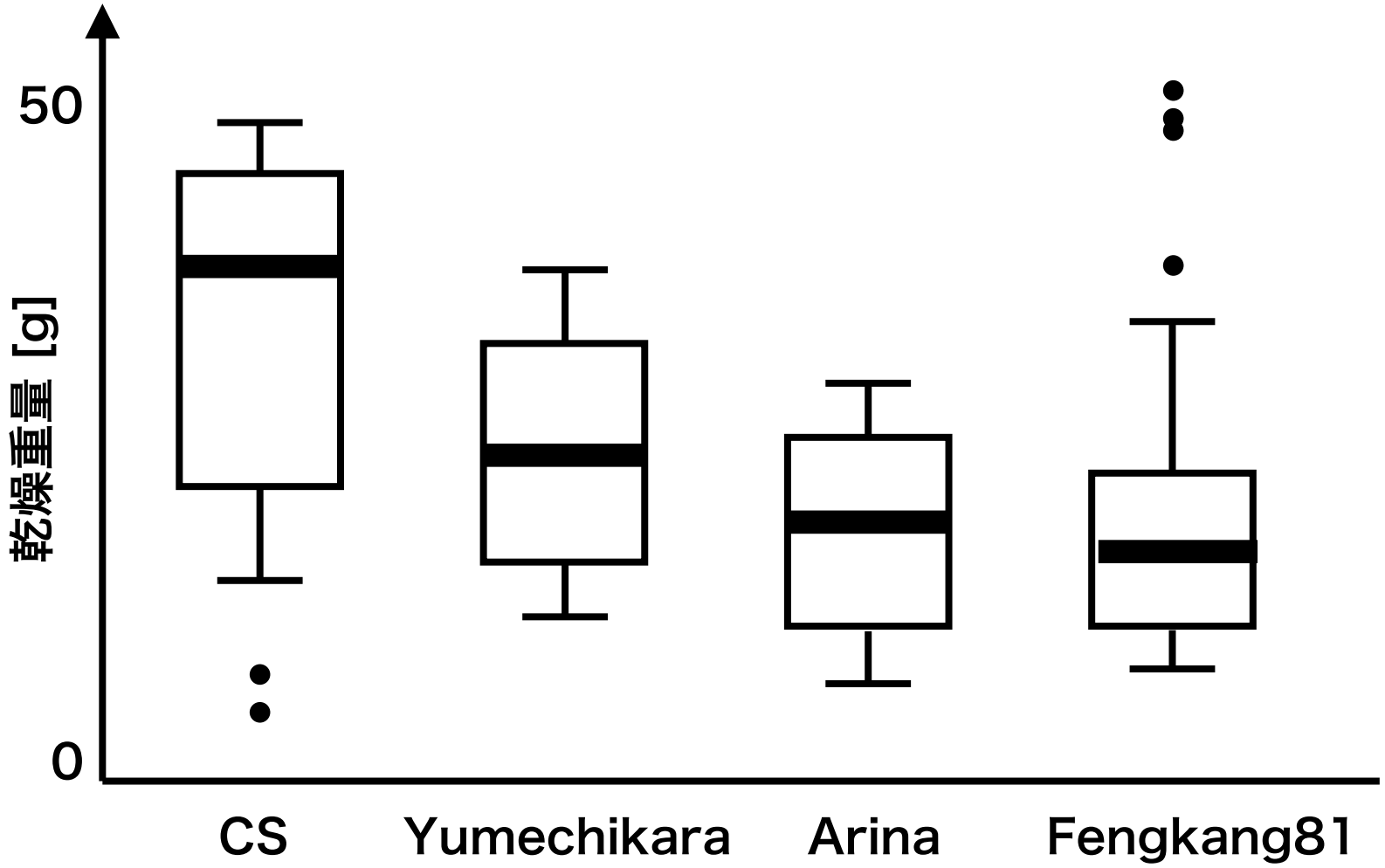


- 1変量の連続値データを視覚化するためのグラフ
- 最大値、最小値、第1四分位数（Q1）、中央値、第3四分位数（Q3）などを簡単に確認できる
- Q3±1.5(Q3-Q1) 範囲外にあるデータは外れ値と定義される

基本グラフ

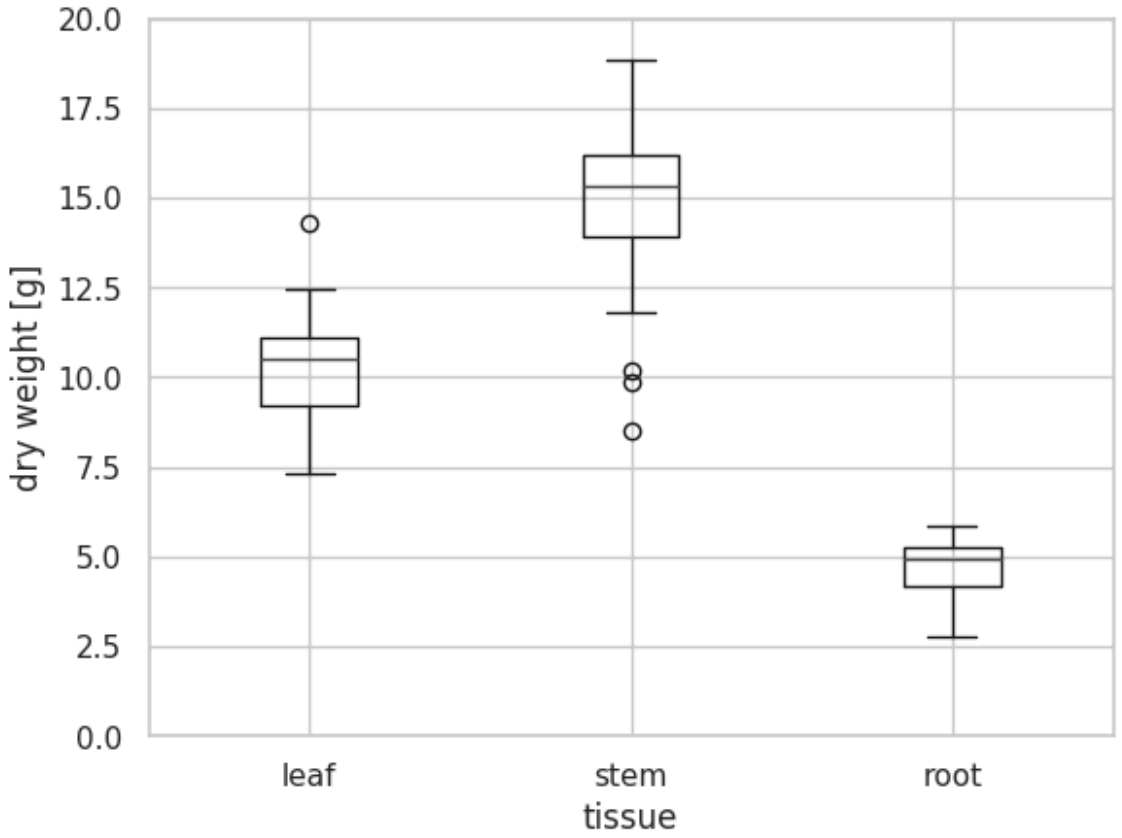
- 線グラフ
- 散布図
- 棒グラフ
- ヒストグラム
- ボックスプロット
- 円グラフ

ボックスプロット



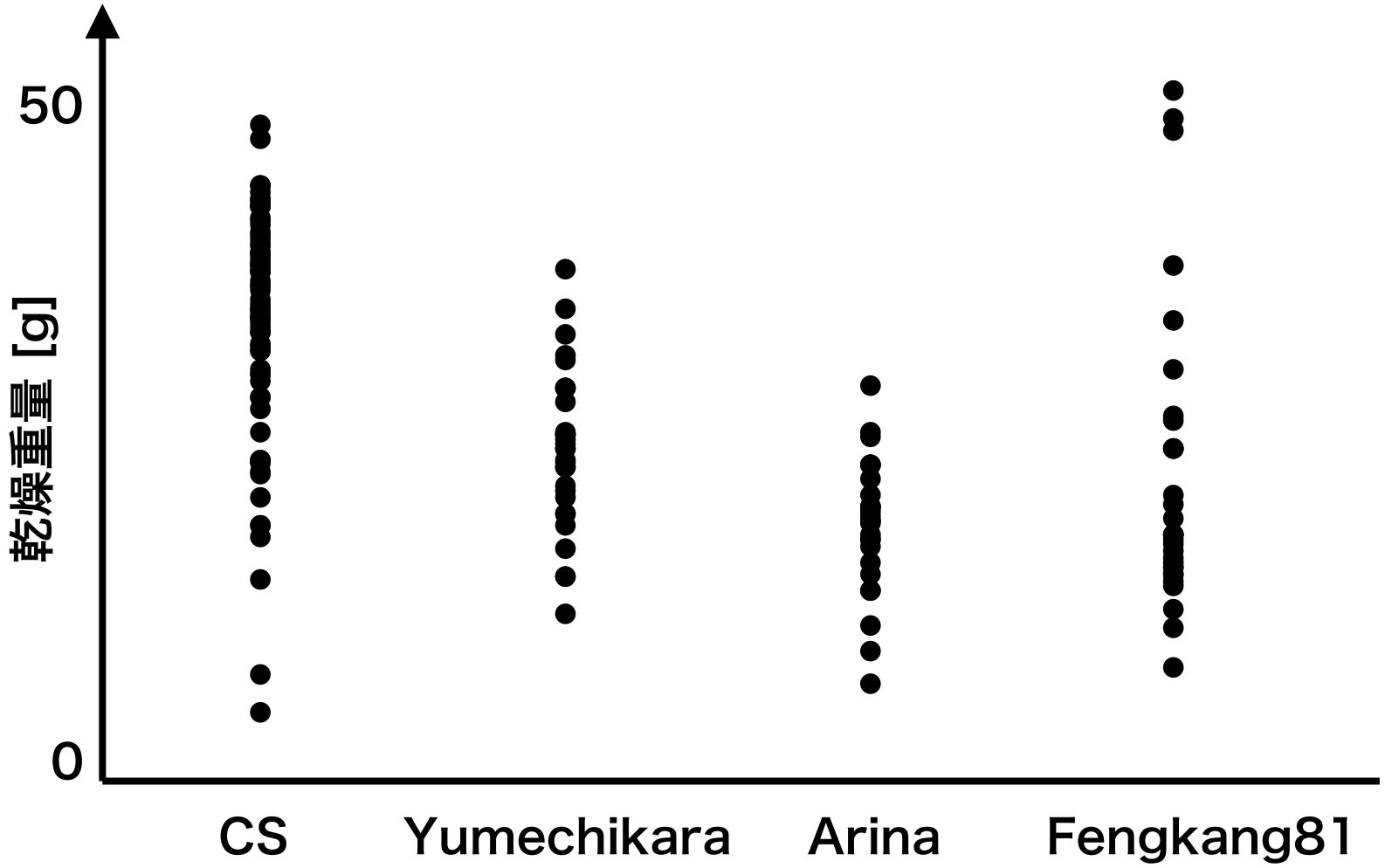
- 1変量の連続値データを視覚化するためのグラフ
- 最大値、最小値、第1四分位数（Q1）、中央値、第3四分位数（Q3）などを簡単に確認できる
- Q3±1.5(Q3-Q1) 範囲外にあるデータは外れ値と定義される
- 複数カテゴリのデータの分布を確認するときに便利

ボックスプロット



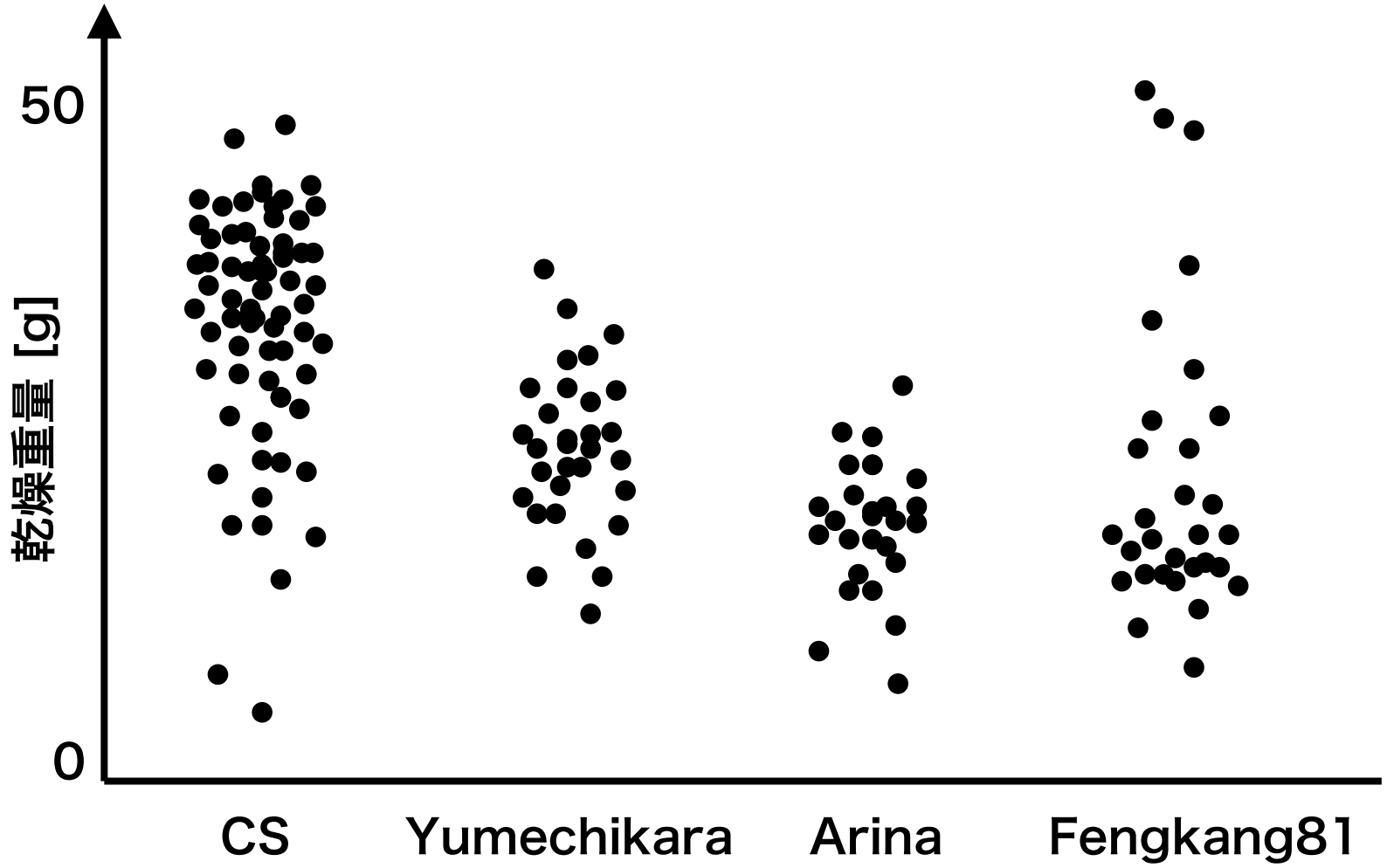

```
import numpy as  np
import matplotlib.pyplot as plt
np.random.seed(2018)
x1 = np.random.normal(10, 2, 20)
x2 = np.random.normal(15, 3, 20)
x3 = np.random.normal(5, 1, 20)
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.boxplot([x1, x2, x3],
           labels=['leaf', 'stem', 'root'])
ax.set_xlabel('tissue')
ax.set_ylabel('dry weight [g]')
ax.set_ylim(0, 20)
fig.show()
```

jitter



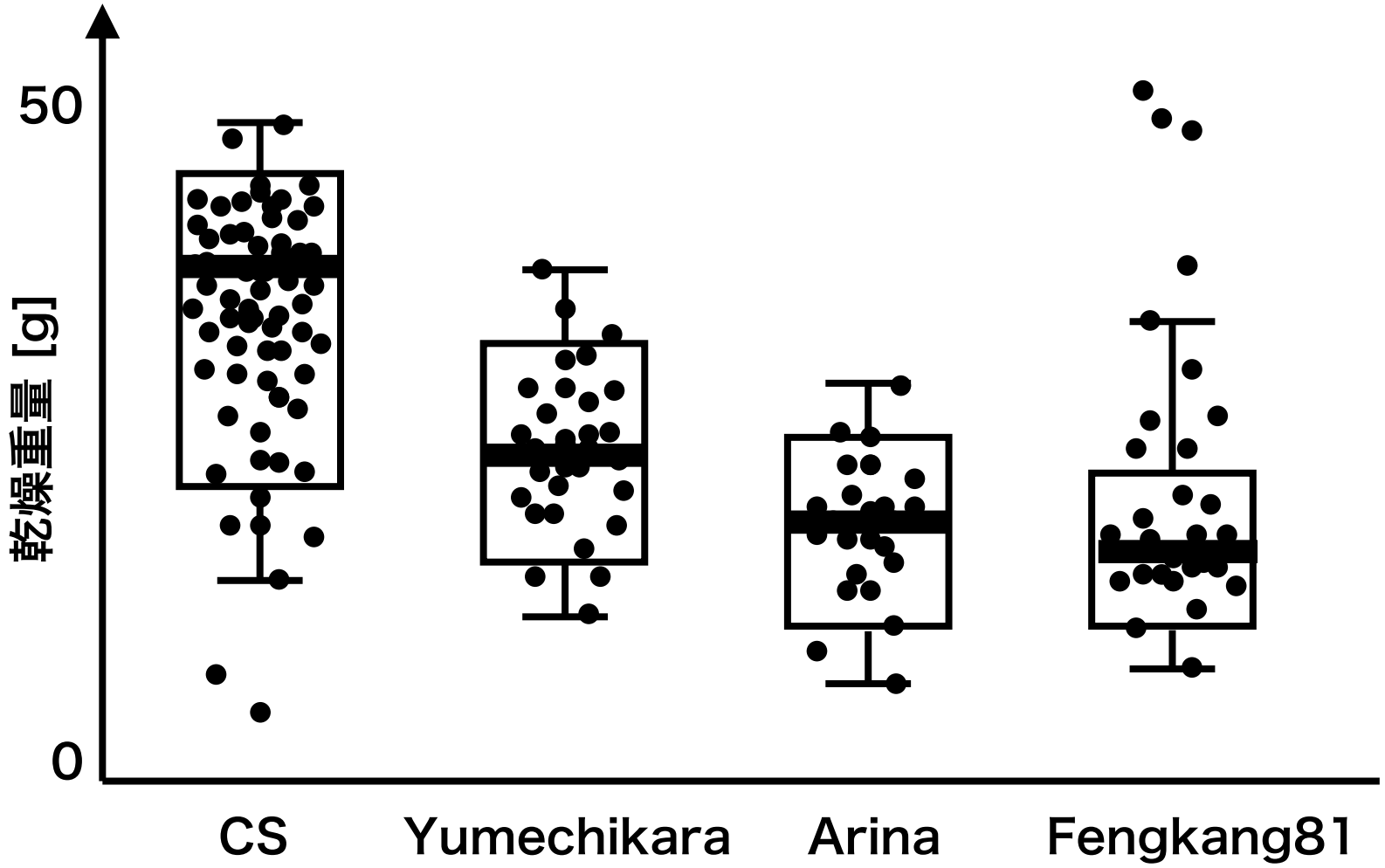
- 複数カテゴリの1変量連続値データの分布を確認ためのグラフ
- 実際の値をプロットすると、データが多いところが重なるため、点を左右にランダムにずらしてプロットする

jitter



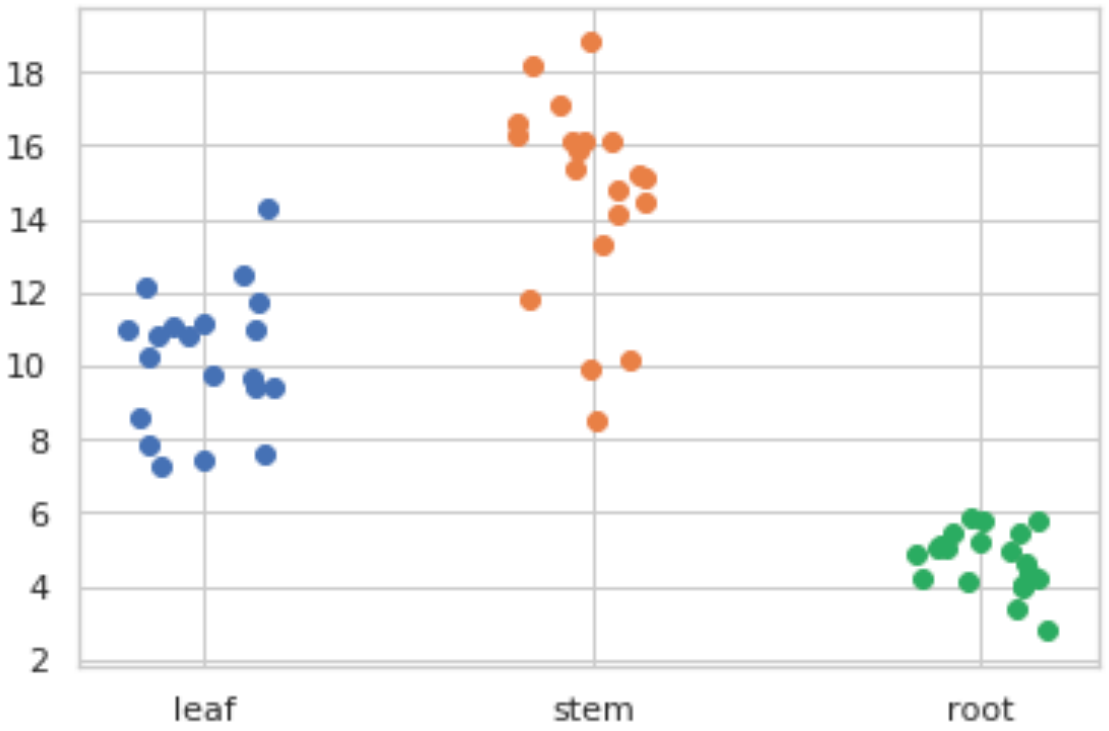
- 複数カテゴリの1変量連続値データの分布を確認ためのグラフ
- 実際の値をプロットすると、データが多いところが重なるため、点を左右にランダムにずらしてプロットする

jitter



- 複数カテゴリの1変量連続値データの分布を確認ためのグラフ
- 実際の値をプロットすると、データが多いところが重なるため、点を左右にランダムにずらしてプロットする
- ボックスプロットと合わせて使われる場合もある

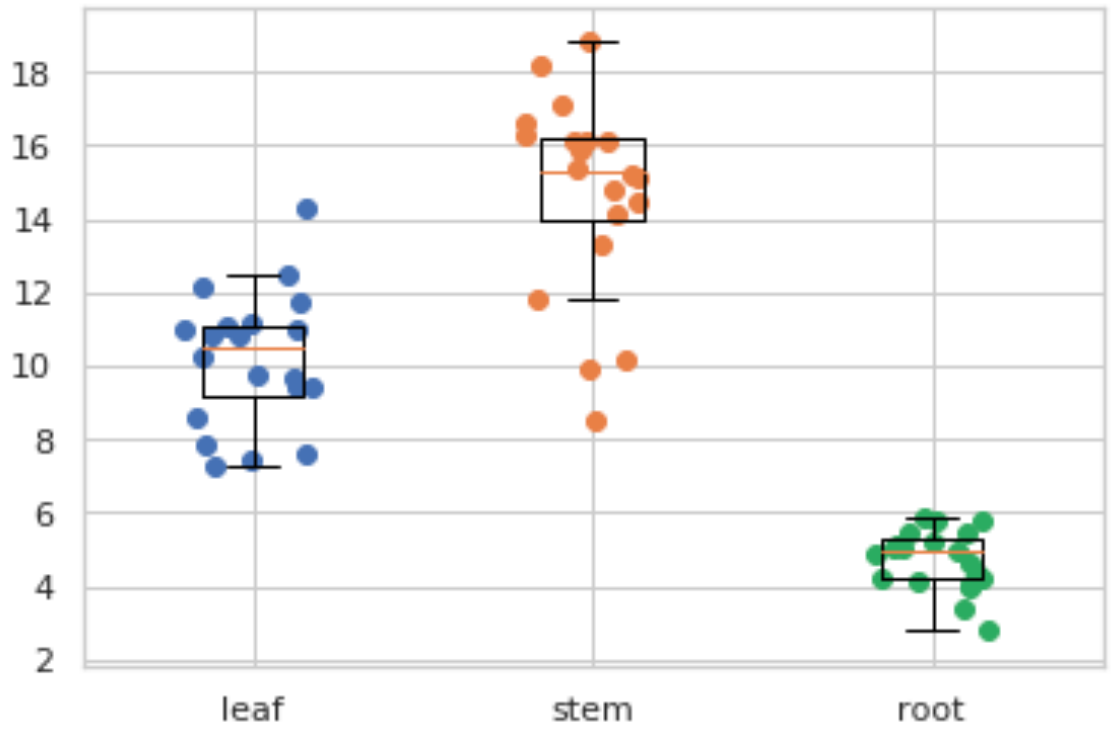
jitter




```
import numpy as np
import matplotlib.pyplot as plt
np.random.seed(2018)
y1 = np.random.normal(10, 2, 20)
y2 = np.random.normal(15, 3, 20)
y3 = np.random.normal(5, 1, 20)

x1 = 1 + np.random.uniform(-0.2, 0.2, len(y1))
x2 = 2 + np.random.uniform(-0.2, 0.2, len(y2))
x3 = 3 + np.random.uniform(-0.2, 0.2, len(y3))

fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)

ax.scatter(x1, y1)
ax.scatter(x2, y2)
ax.scatter(x3, y3)

ax.set_xticks([1, 2, 3])
ax.set_xticklabels(['leaf', 'stem', 'root'])
fig.show()
```

y1、y2、y3 のデータの y 座標をデータの値のそのままにして、x 座標をそれぞれ 1、2、3 から少し左右にずらした値にする。

jitter + ボックスプロット



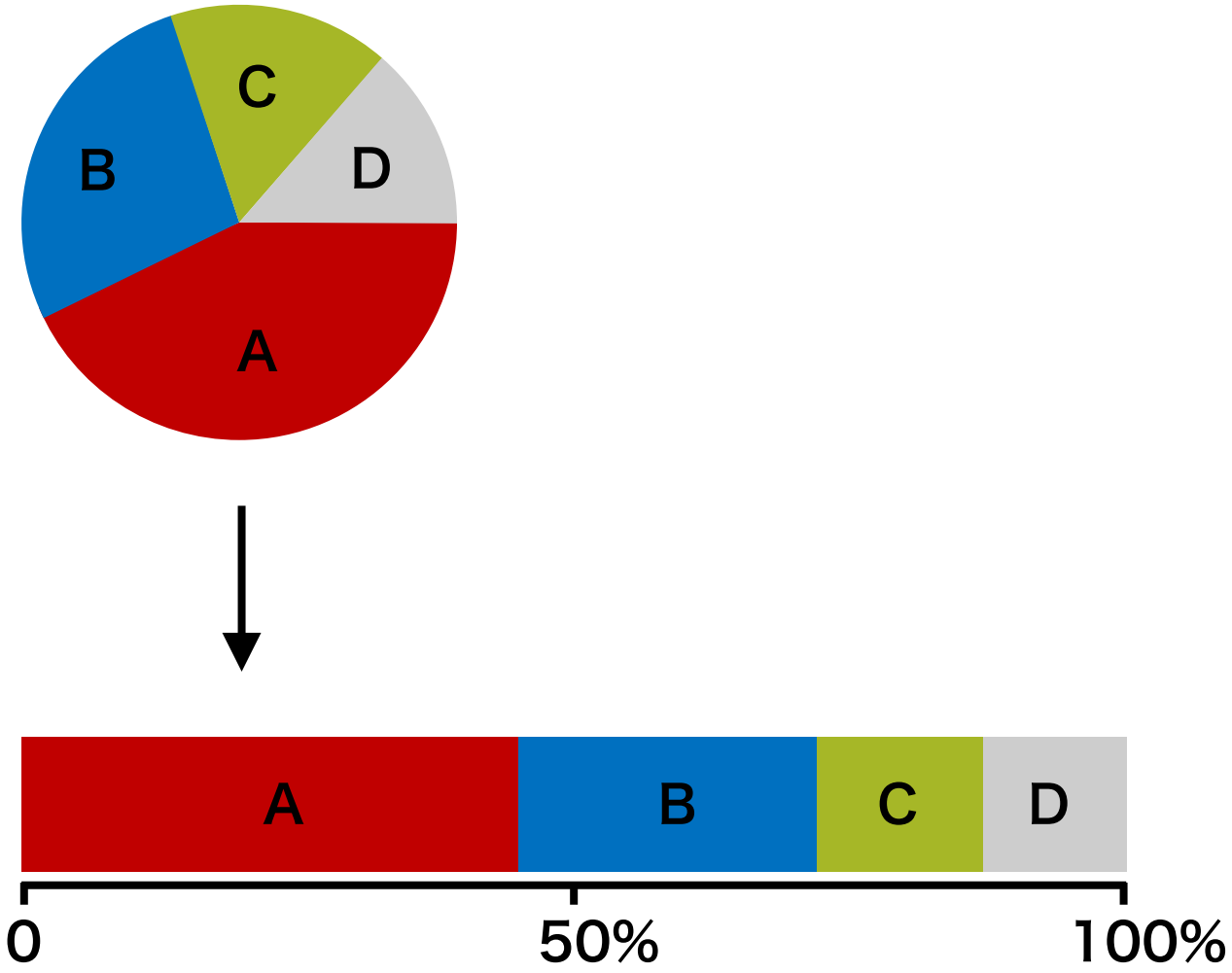

```
import numpy as np
import matplotlib.pyplot as plt
np.random.seed(2018)
y1 = np.random.normal(10, 2, 20)
y2 = np.random.normal(15, 3, 20)
y3 = np.random.normal(5, 1, 20)

x1 = 1 + np.random.uniform(-0.2, 0.2, len(y1))
x2 = 2 + np.random.uniform(-0.2, 0.2, len(y2))
x3 = 3 + np.random.uniform(-0.2, 0.2, len(y3))

fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)

ax.scatter(x1, y1)
ax.scatter(x2, y2)
ax.scatter(x3, y3)

ax.boxplot([y1, y2, y3],
           labels=['leaf', 'stem', 'root'],
           showfliers=False)
fig.show()
```

showfliers=False を指定することで、boxplot メソッドは外れ値を描かなくなる。

基本グラフ

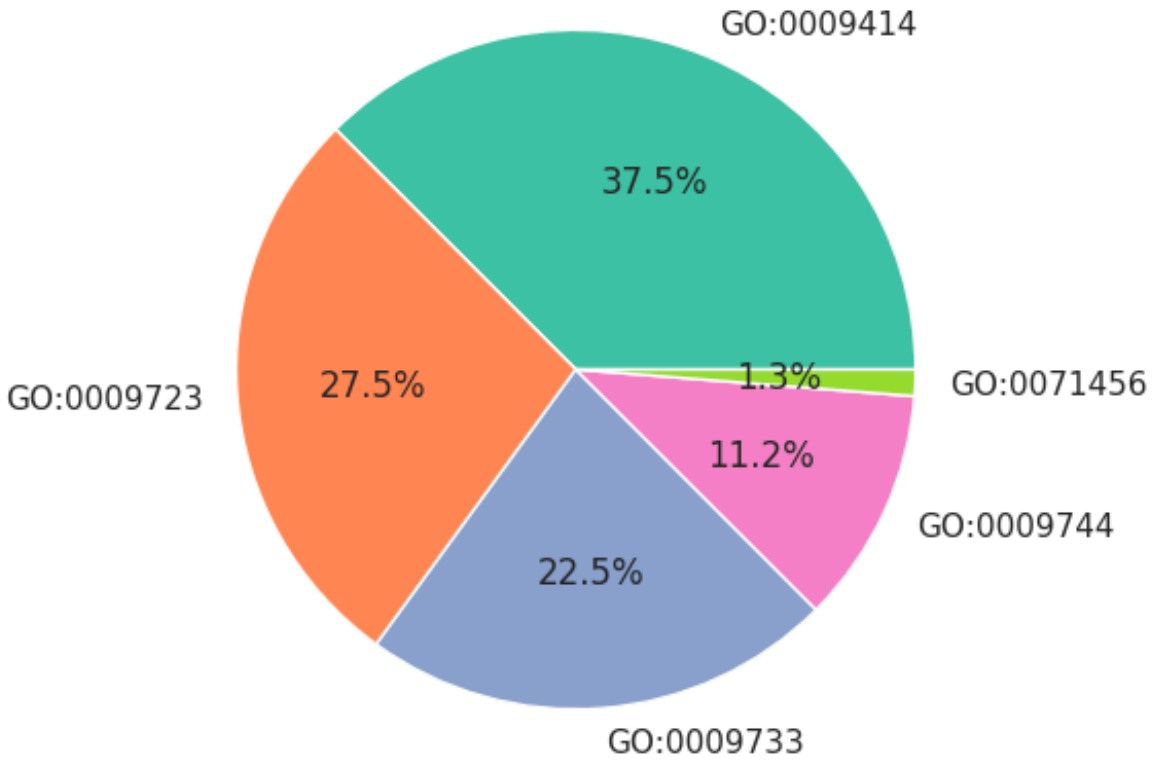
- 線グラフ
- 散布図
- 棒グラフ
- ヒストグラム
- ボックスプロット
- 円グラフ

円グラフ



- 円グラフは誤解されやすい、円グラフを用いる前に帯状グラフで代用できないかを確認すること
- 3D円グラフは基本的に人を騙すようのグラフであることに注意

円グラフ

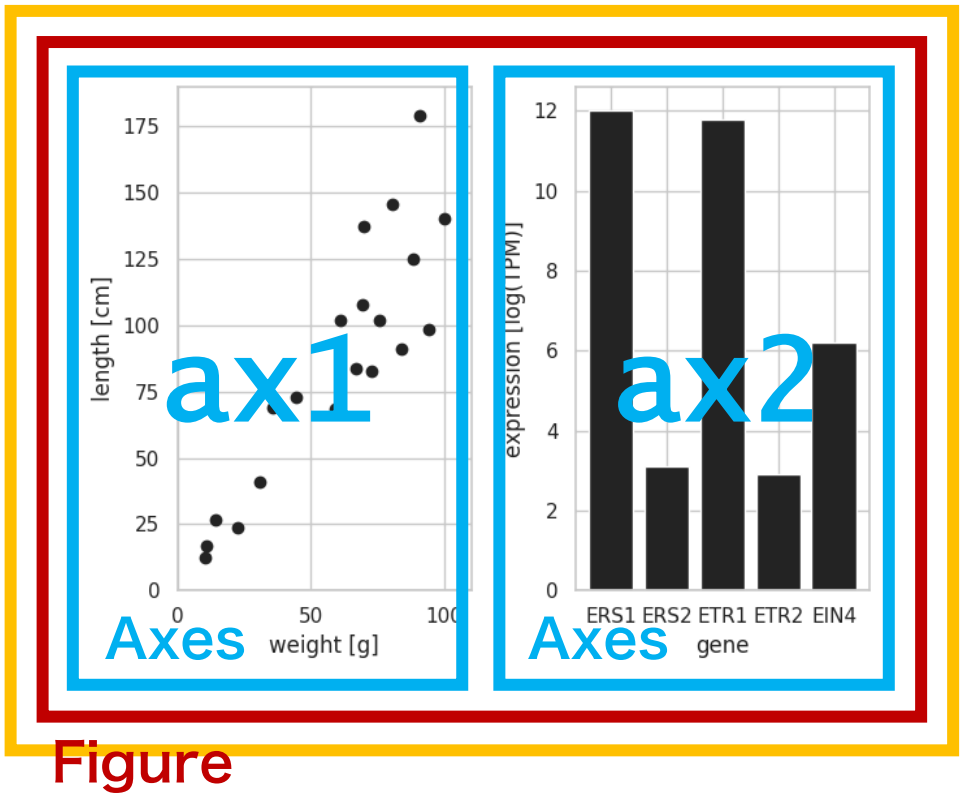



```
import matplotlib.pyplot as plt
import numpy as np

x = ['GO:0009414', 'GO:0009723',
     'GO:0009733', 'GO:0009744',
     'GO:0071456']
y = np.array([300, 220, 180, 90, 10])

fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.pie(y, labels=x, autopct="%1.1f%%")
ax.axis('equal')

fig.show()
```

可視化

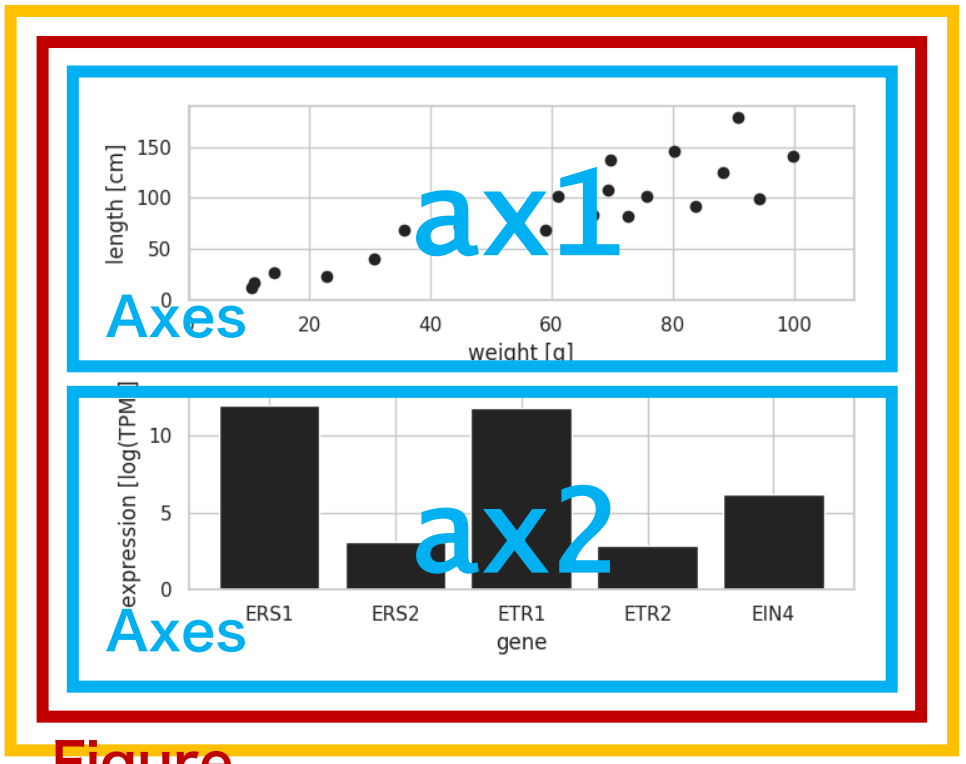
- matplotlib 基本
- 基本グラフ
- プロット領域の分割

画面分割



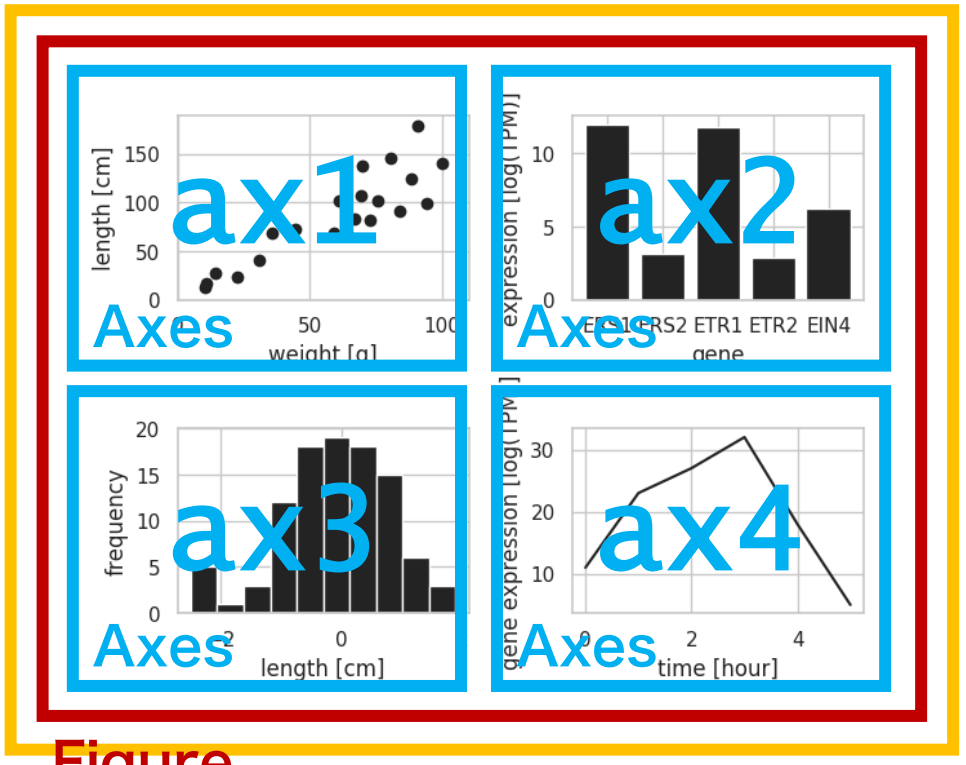

```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
fig = plt.figure()
x1 = np.random.uniform(0, 100, 20)
y1 = x1 * np.random.uniform(1, 2, 20)
ax1 = fig.add_subplot(1, 2, 1)
ax1.scatter(x1, y1)
x2 = np.array(['ERS1', 'ERS2', 'ETR1', 'ETR2', 'EIN4'])
y2 = np.array([12.0, 3.1, 11.8, 2.9, 6.2])
x2_position = np.arange(len(x2))
ax2 = fig.add_subplot(1, 2, 2)
ax2.bar(x2_position, y2, tick_label=x2)
fig.show()
```

画面分割




```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
fig = plt.figure()
x1 = np.random.uniform(0, 100, 20)
y1 = x1 * np.random.uniform(1, 2, 20)
ax1 = fig.add_subplot(2, 1, 1)
ax1.scatter(x1, y1)
x2 = np.array(['ERS1', 'ERS2', 'ETR1', 'ETR2', 'EIN4'])
y2 = np.array([12.0, 3.1, 11.8, 2.9, 6.2])
x2_position = np.arange(len(x2))
ax2 = fig.add_subplot(2, 1, 2)
ax2.bar(x2_position, y2, tick_label=x2)
fig.show()
```

画面分割




```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt


fig = plt.figure()
ax1 = fig.add_subplot(2, 2, 1)
ax1.scatter(x1, y1)
ax2 = fig.add_subplot(2, 2, 2)
ax2.bar(x2, y2)
ax3 = fig.add_subplot(2, 2, 3)
ax3.hist(x3)
ax4 = fig.add_subplot(2, 2, 4)
ax4.plot(x4, y4)
fig.show()
```

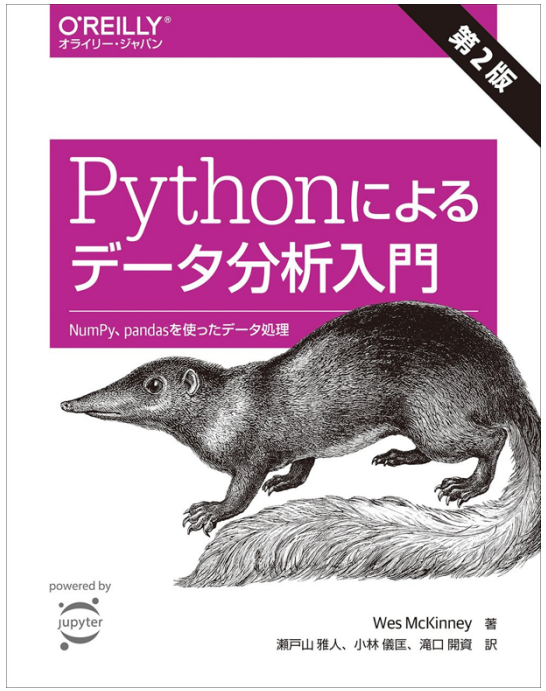
参考図書＆ウェブサイト

Dive Into Python 3 日本語版
http://diveintopython3-ja.rdy.jp/

powerful Python data analysis toolkit
https://pandas.pydata.org/pandas-docs/stable/index.html

matplotlib gallery
https://matplotlib.org/gallery.html

seaborn gallery
https://seaborn.pydata.org/examples/index.html



機械学習のための特徴量エンジニアリング – その原理とPython による実践 –

オライリー・ジャパン

機械学習概略

- 人工知能と機械学習
- 開発ワークフロー
- ニューラルネットワーク

機械学習概略

- 人工知能と機械学習
- 開発ワークフロー
- ニューラルネットワーク

環境問題	• 種の多様性保全 • 外来種侵入の防止 • 平均気温の上昇抑制 • グリーンエネルギー参画	虫取り網　統計モデリング 温度センサー　紫外線センサー クーロンの法則　ドローン エネルギー保存の法則
食料問題	• 環境変動に頑健な品種改良 • 食料廃棄量の削減 • 農耕地の効率的な利用 • 農家支援	車　飛行機　ベイズ理論 自動運転ロボット　衛星
医療問題	• 医療業務の効率化 • 診断治療の効率化 • 感染病の国内侵入の防止 • オーダーメイド医療	待ち行列理論　CRISPR/Cas9 de Bruijn Graph HiSeq　メンデルの法則 PCR　ボイヤー・ムーア法

社会問題 に対し

規律・秩序を制定し、

問題解決の最善方策を探索する。

社会問題

人工知能

機械学習

人工知能

- 1947 A. M. Tuning が人工知能の概念を提唱。
- 1950 チューリングテストを利用して機械が知性を持つかどうかを検証。
- 1956 Dartmouth 会議で初めて Artificial Intelligence が使われた。

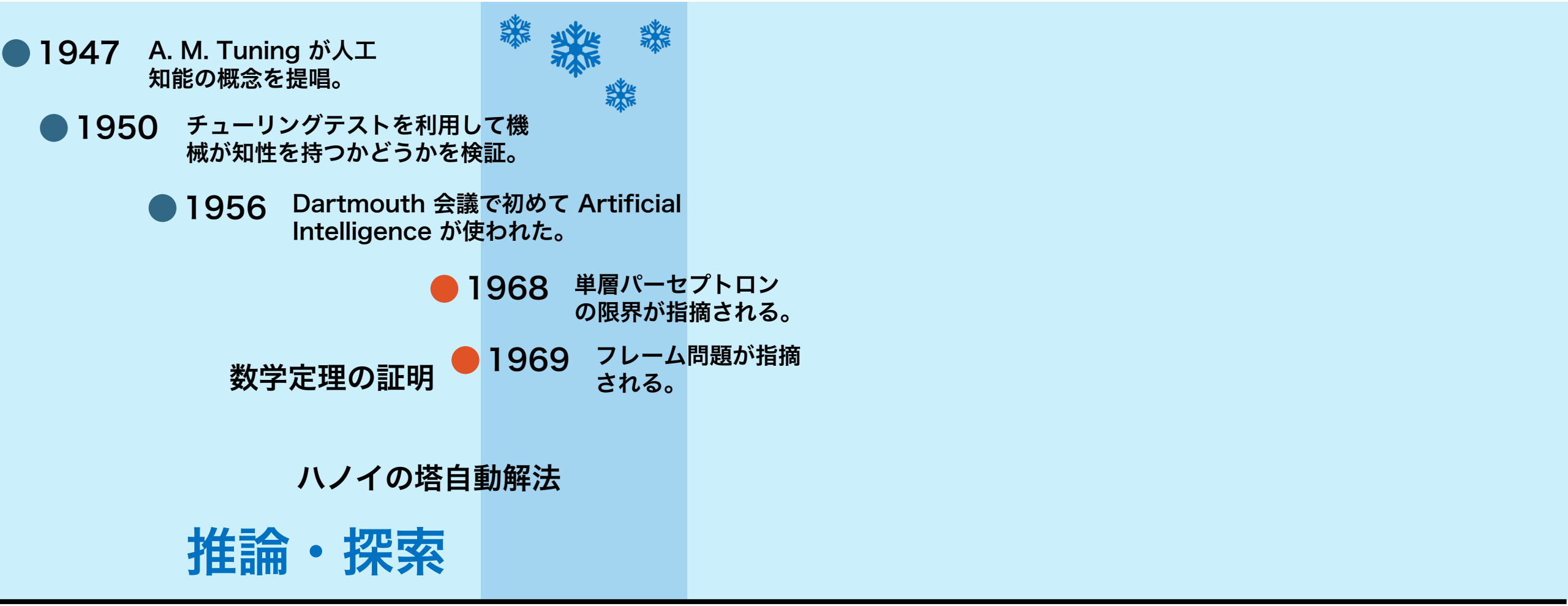
数学定理の証明

ハノイの塔自動解法

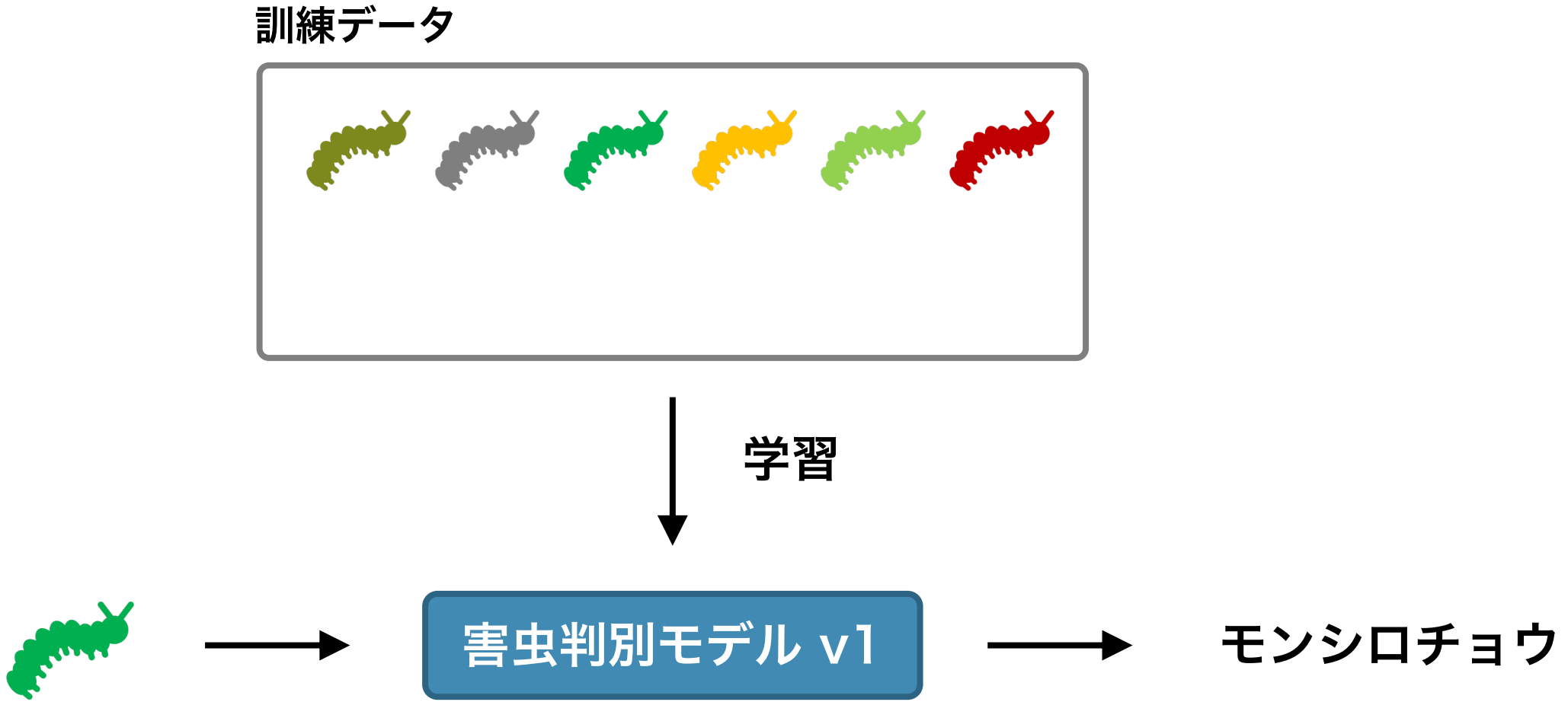
推論・探索

1950 1960 1970 1980 1990 2000 2010

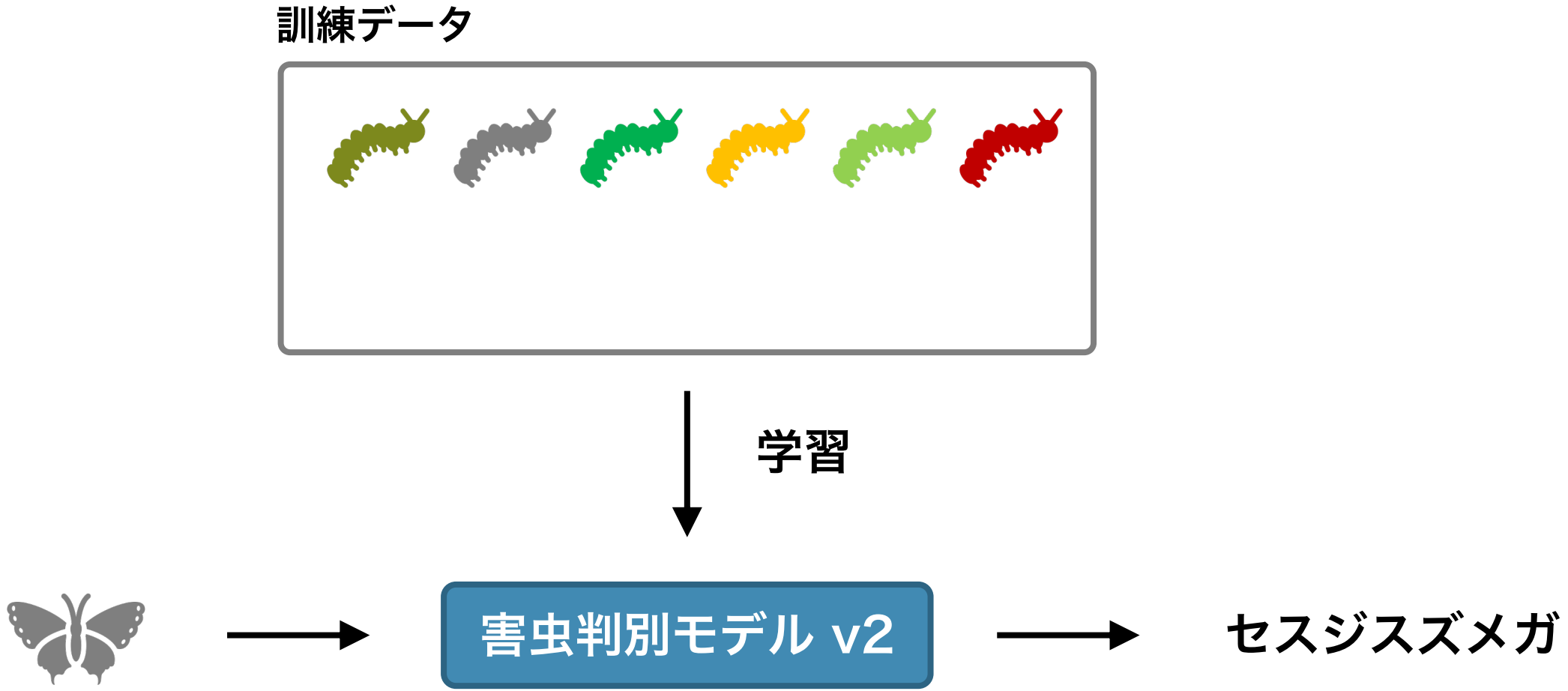
人工知能


1947 A. M. Tuning が人工知能の概念を提唱。
1950 チューリングテストを利用して機械が知性を持つかどうかを検証。
1956 Dartmouth 会議で初めて Artificial Intelligence が使われた。
1968 単層パーセプトロンの限界が指摘される。
1969 フレーム問題が指摘される。
数学定理の証明
ハノイの塔自動解法
推論・探索
1950
1960
1970
1980
1990
2000
2010

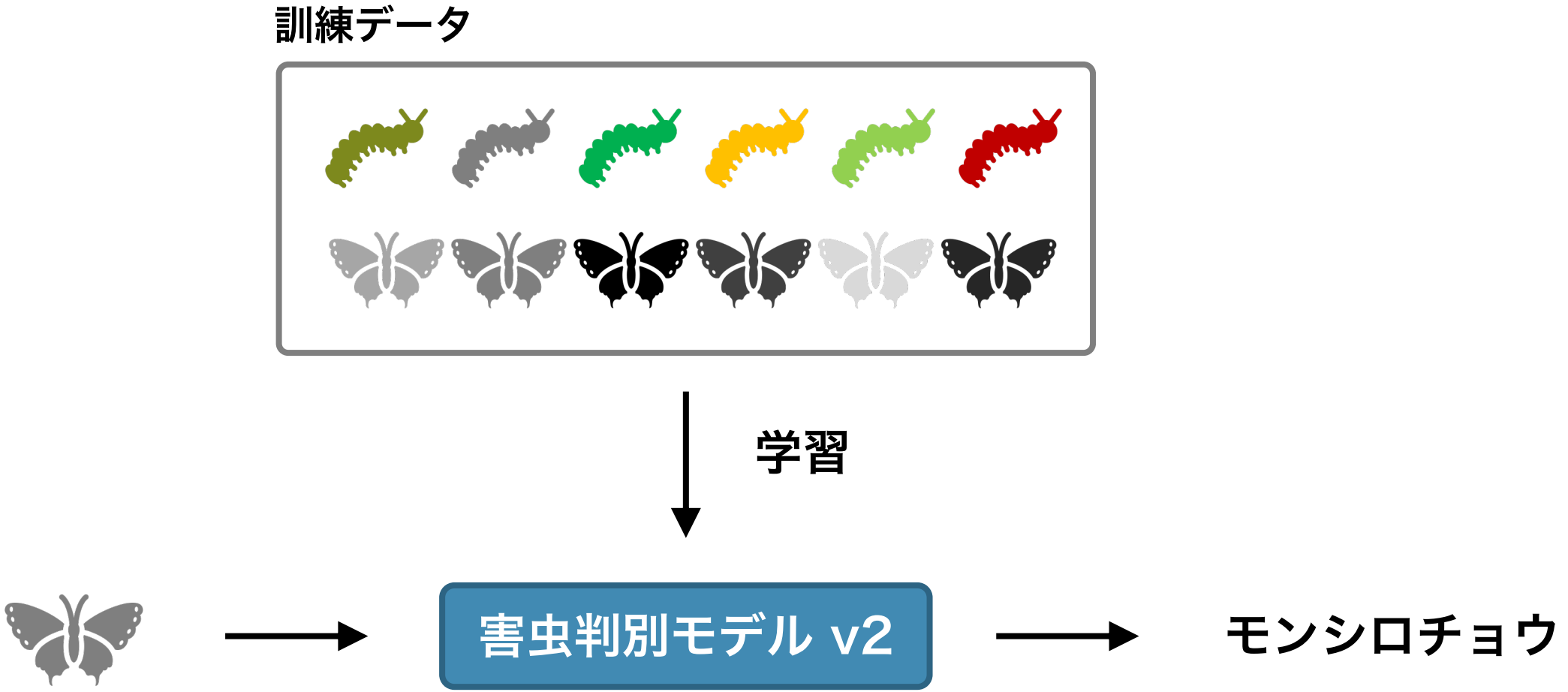
フレーム問題



フレーム問題

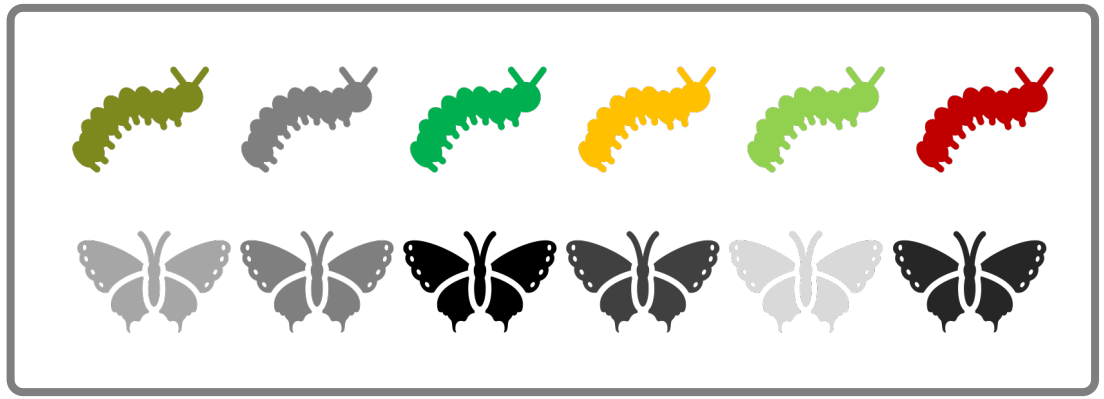


フレーム問題



フレーム問題

訓練データ



学習

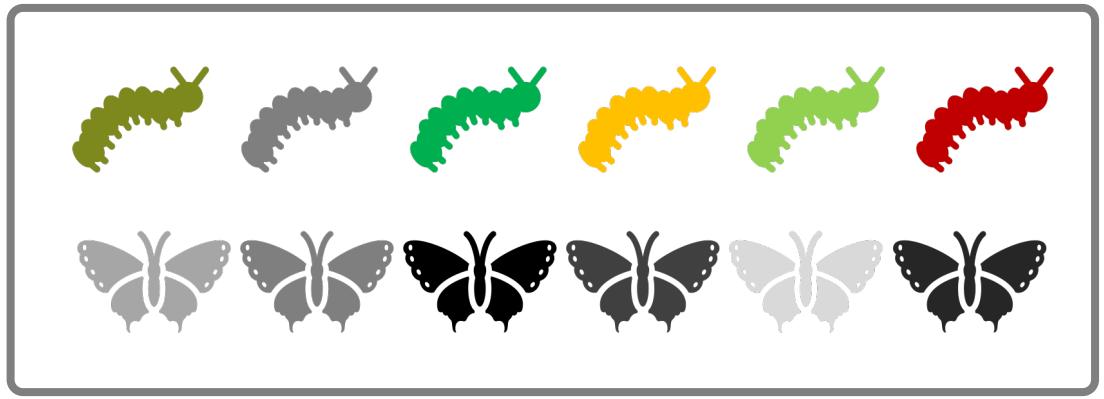
新規外来種

害虫判別モデル v2

オオゴマダラ

フレーム問題

訓練データ



+ 外来種データ
+ 蛹データ
+ 卵データ

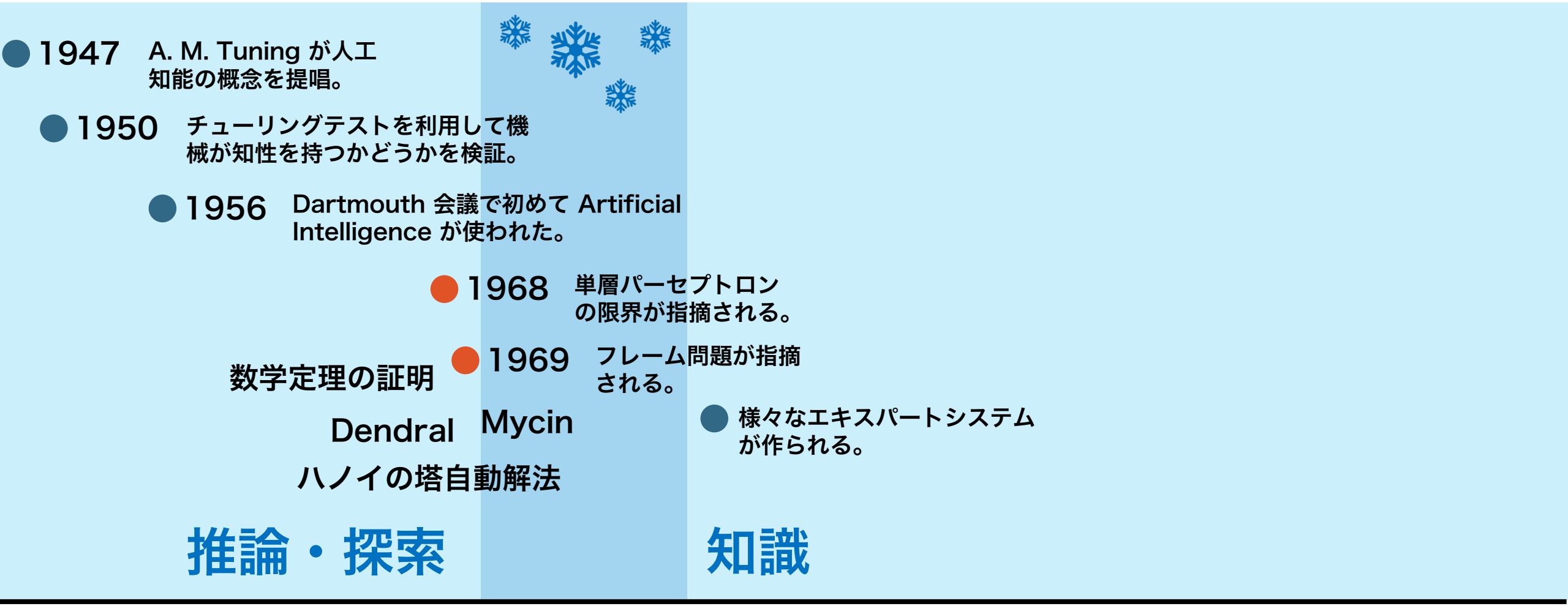
学習

新規外来種

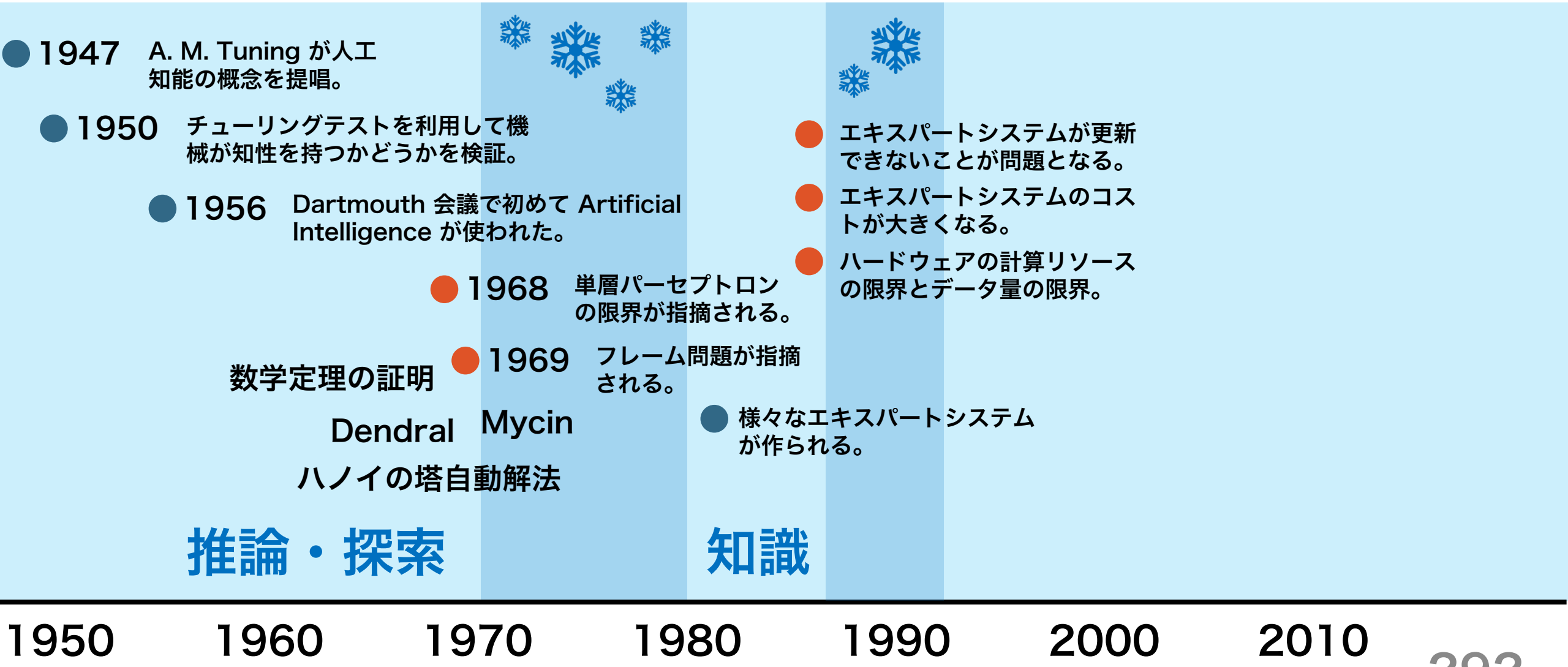
害虫判別モデル v2

オオゴマダラ

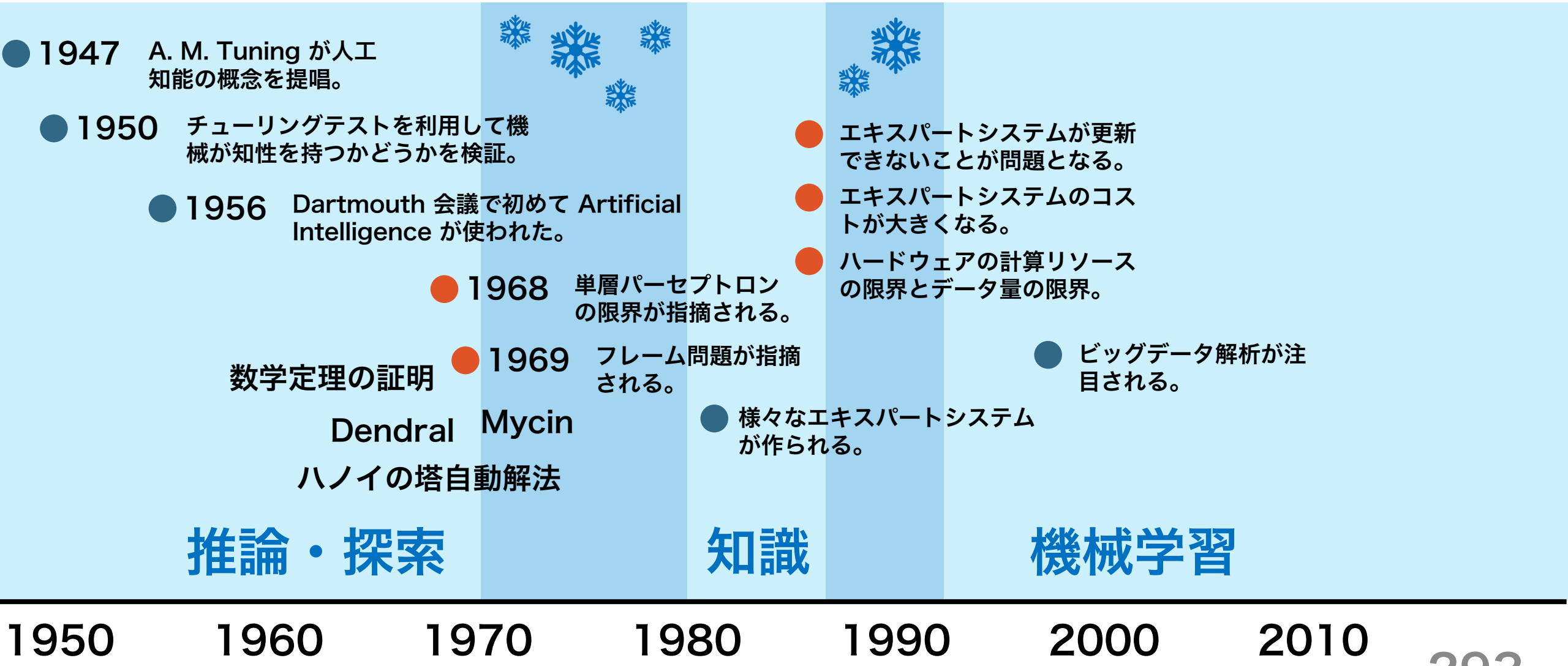
人工知能


1947 A. M. Tuning が人工知能の概念を提唱。
1950 チューリングテストを利用して機械が知性を持つかどうかを検証。
1956 Dartmouth 会議で初めて Artificial Intelligence が使われた。
1968 単層パーセプトロンの限界が指摘される。
1969 フレーム問題が指摘される。
数学定理の証明
Dendral
Mycin
ハノイの塔自動解法
様々なエキスパートシステムが作られる。
推論・探索
知識
1950
1960
1970
1980
1990
2000
2010

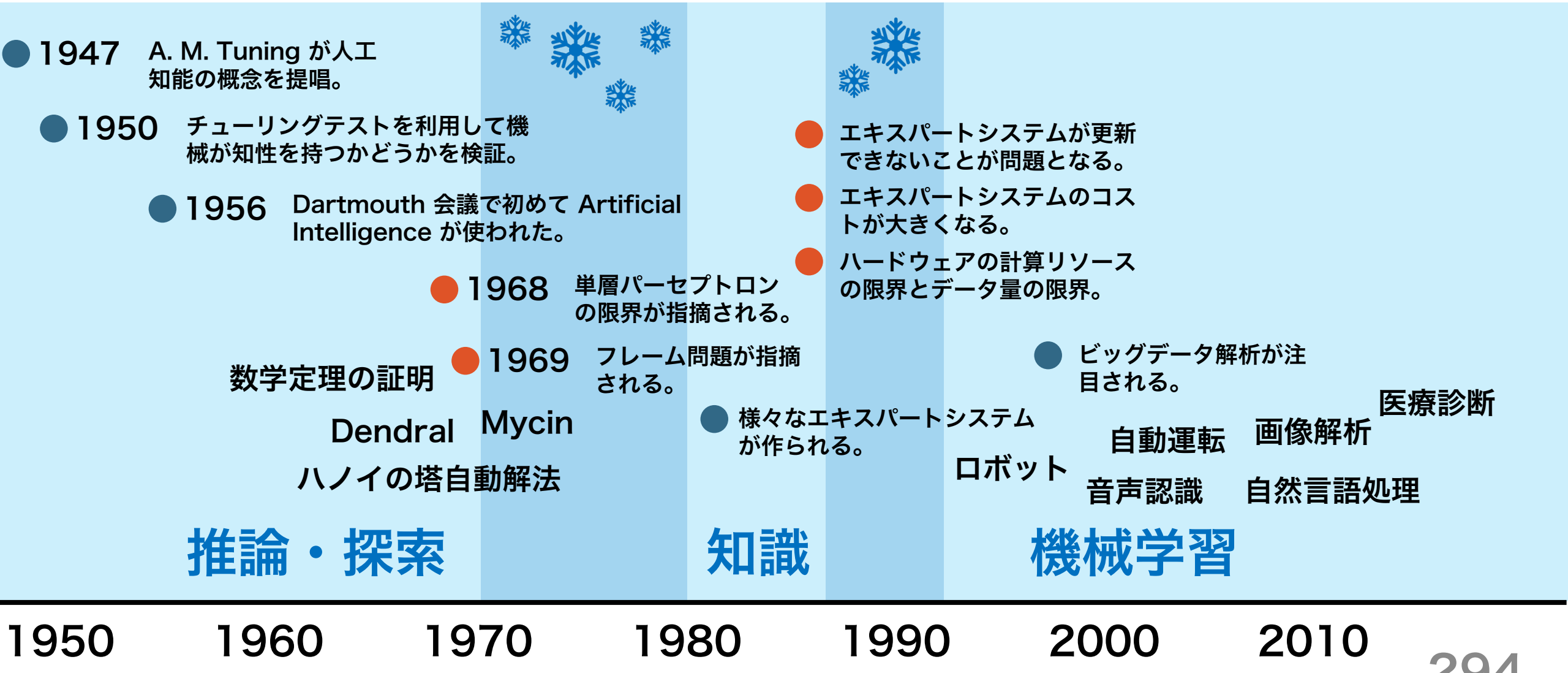
人工知能


1947 A. M. Tuning が人工知能の概念を提唱。
1950 チューリングテストを利用して機械が知性を持つかどうかを検証。
1956 Dartmouth 会議で初めて Artificial Intelligence が使われた。
1968 単層パーセプトロンの限界が指摘される。
1969 フレーム問題が指摘される。
数学定理の証明
Dendral
Mycin
ハノイの塔自動解法
様々なエキスパートシステムが作られる。
エキスパートシステムが更新できないことが問題となる。
エキスパートシステムのコストが大きくなる。
ハードウェアの計算リソースの限界とデータ量の限界。
推論・探索
知識
1950
1960
1970
1980
1990
2000
2010

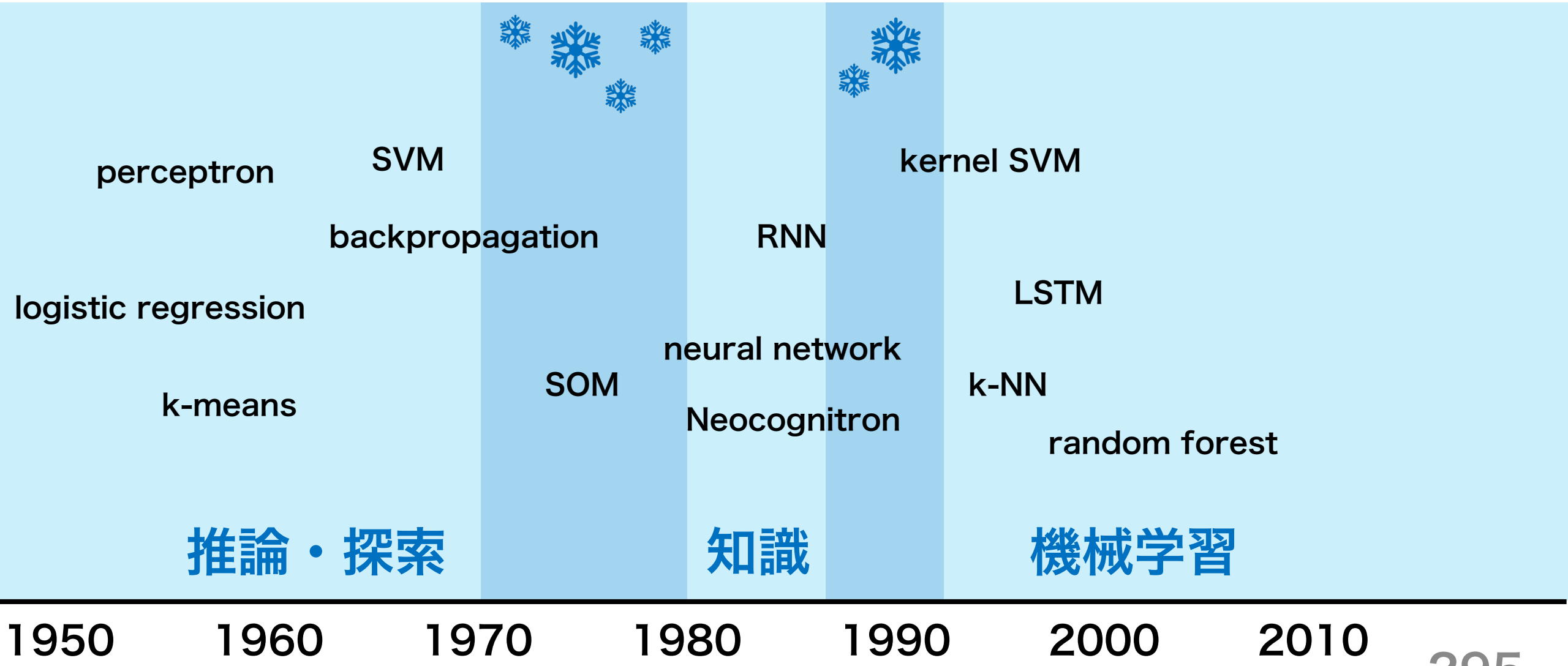
人工知能


1947 A. M. Tuning が人工知能の概念を提唱。
1950 チューリングテストを利用して機械が知性を持つかどうかを検証。
1956 Dartmouth 会議で初めて Artificial Intelligence が使われた。
1968 単層パーセプトロンの限界が指摘される。
1969 フレーム問題が指摘される。
数学定理の証明
Dendral
Mycin
ハノイの塔自動解法
様々なエキスパートシステムが作られる。
エキスパートシステムが更新できないことが問題となる。
エキスパートシステムのコストが大きくなる。
ハードウェアの計算リソースの限界とデータ量の限界。
ビッグデータ解析が注目される。
推論・探索
知識
機械学習
1950
1960
1970
1980
1990
2000
2010

人工知能


1947 A. M. Tuning が人工知能の概念を提唱。
1950 チューリングテストを利用して機械が知性を持つかどうかを検証。
1956 Dartmouth 会議で初めて Artificial Intelligence が使われた。
1968 単層パーセプトロンの限界が指摘される。
1969 フレーム問題が指摘される。
数学定理の証明
Dendral
Mycin
ハノイの塔自動解法
様々なエキスパートシステムが作られる。
エキスパートシステムが更新できないことが問題となる。
エキスパートシステムのコストが大きくなる。
ハードウェアの計算リソースの限界とデータ量の限界。
ビッグデータ解析が注目される。
ロボット
自動運転
画像解析
医療診断
音声認識
自然言語処理
推論・探索
知識
機械学習
1950
1960
1970
1980
1990
2000
2010

機械学習アルゴリズム


perceptron
SVM
backpropagation
logistic regression
k-means
SOM
RNN
neural network
Neocognitron
kernel SVM
LSTM
k-NN
random forest
推論・探索
知識
機械学習
1950
1960
1970
1980
1990
2000
2010

no free lunch theorem

コスト関数の極値を探索するあらゆるアルゴリズムは、全ての可能なコスト関数に適用した結果を平均すると同じ性能となる

Wolpert and Macready,1995

- 問題領域に関する事前知識がなければ、どの問題に対しても最適なアルゴリズムは存在しない。
- 問題領域の知識を有効活用し、その領域に限定した最適化アルゴリズムを使用するべき。
- 現在のデータに対して、複数の最適化アルゴリズムを適用し、最適なアルゴリズムを選択する。

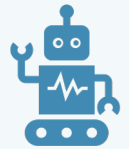
教師あり学習

回帰問題　　分類問題

半教師あり学習

クラスタリング　　次元削減

教師なし学習



強化学習

最適化問題

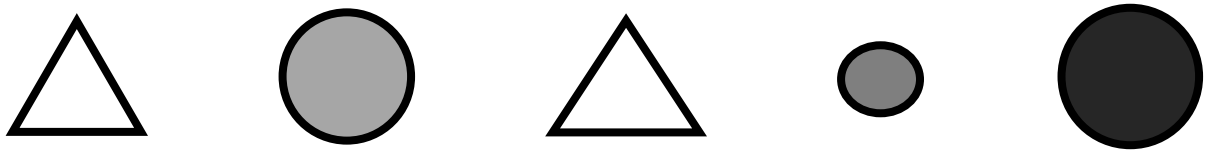
教師あり学習

教師あり学習は、訓練データに特徴量と正解ラベルがペアとして与えられるときに行う学習方法である。教師あり学習は、分類問題および回帰問題などに応用される。教師あり学習で使われる学習アルゴリズムには、次のようなものがある。

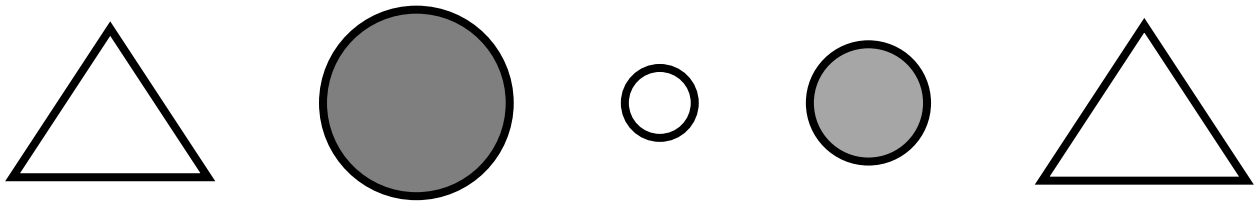
- ニューラルネットワーク
- ロジスティック回帰
- サポートベクトルマシン（SVM）
- 決定木
- ランダムフォレスト
- k 近傍法
- 線形回帰
- スパース回帰

分類問題

円形と三角形を分類したい場合、どの特徴に着目すればいいのか？



x_1	1.0	0.5	1.0	0.4	0.1
x_2	0.8	1.0	0.8	0.9	1.0



x_1	1.0	0.3	1.0	0.5	1.0
x_2	0.8	1.0	1.0	1.0	0.7

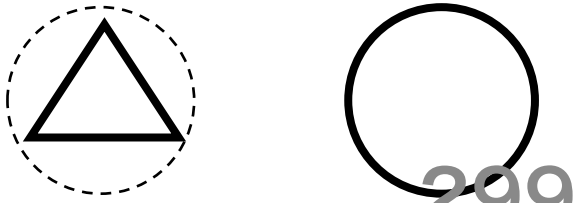
色

色に着目すると、三角形は白が多く、円形は黒い場合が多い。ここで、白を 1、黒を 0 とする色の指標を考える。

0 1

外接円との面積比

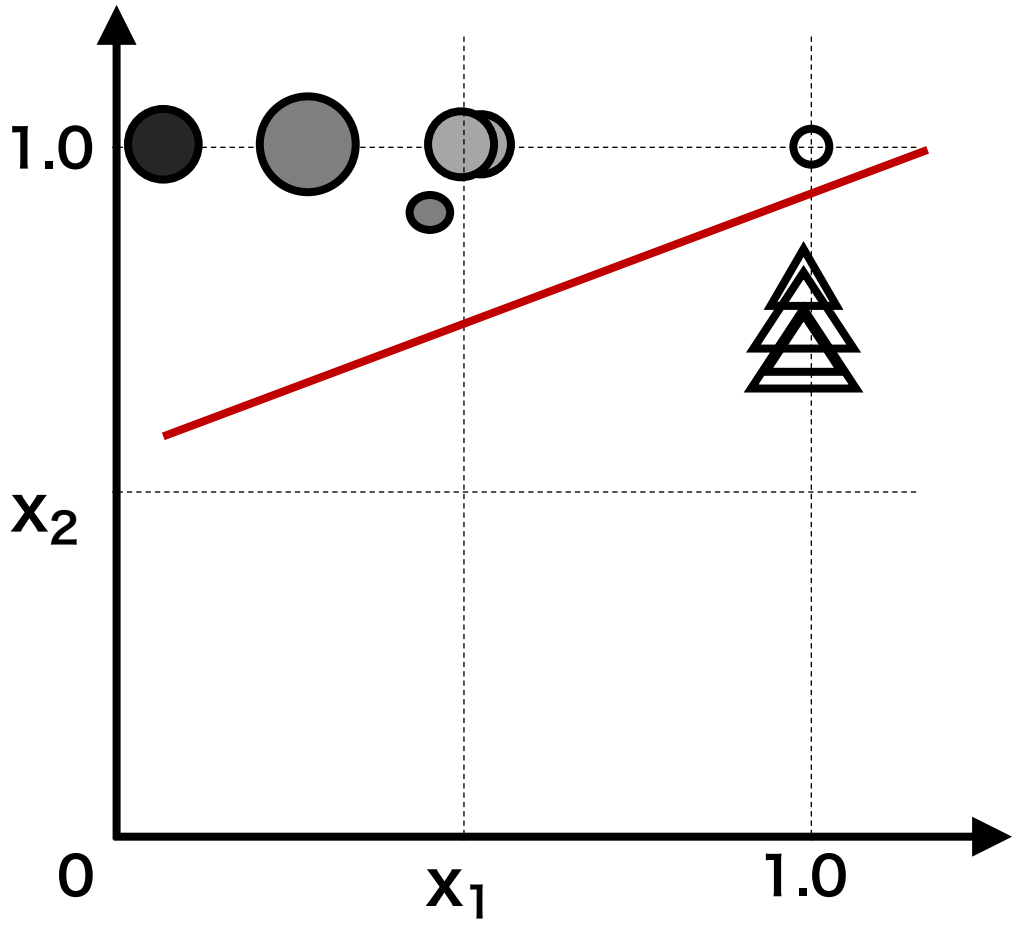
分類したい形とその外接円の面積の比に着目すると、三角形の場合は 1 よりも小さく、円形の場合はほぼ 1 になる。



分類問題

円形と三角形を分類したい場合、どの特徴に着目すればいいのか？

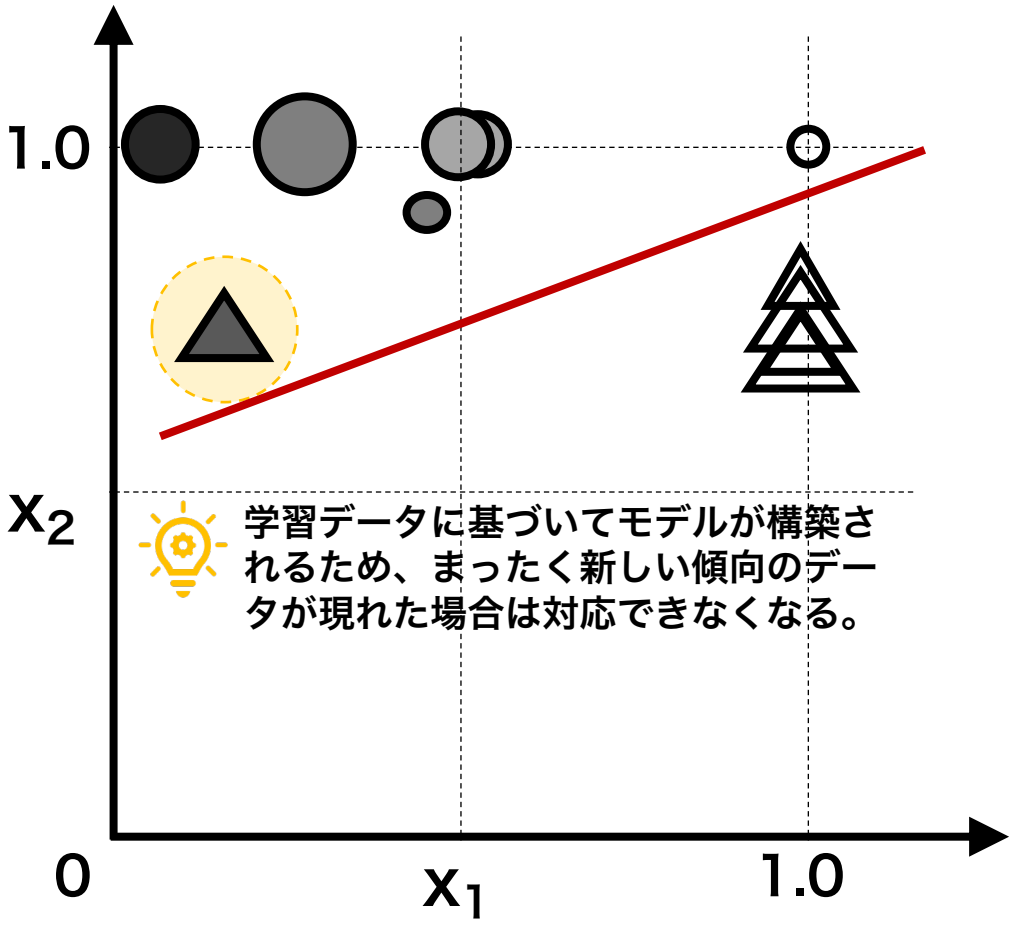
ラベル	X_1	X_2
1	1.0	0.8
0	0.7	1.0
1	1.0	0.8
0	0.8	0.9
0	0.9	1.0
1	1.0	0.8
0	0.8	1.0
0	1.0	1.0
0	0.7	1.0
1	1.0	0.7



分類問題

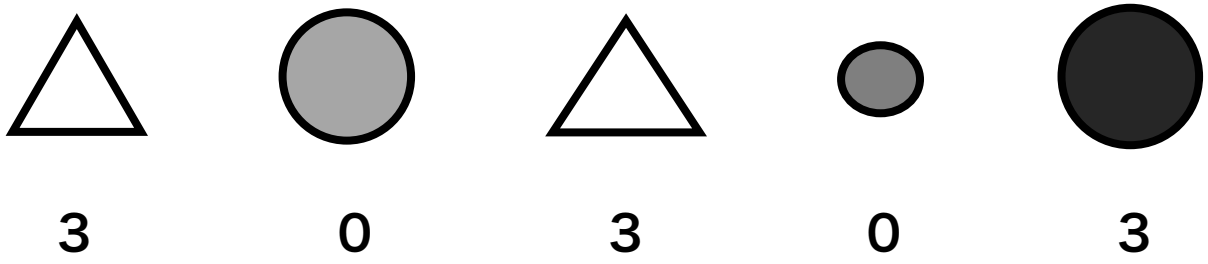
円形と三角形を分類したい場合、どの特徴に着目すればいいのか？

ラベル	X_1	X_2
1	1.0	0.8
0	0.7	1.0
1	1.0	0.8
0	0.8	0.9
0	0.9	1.0
1	1.0	0.8
0	0.8	1.0
0	1.0	1.0
0	0.7	1.0
1	1.0	0.7

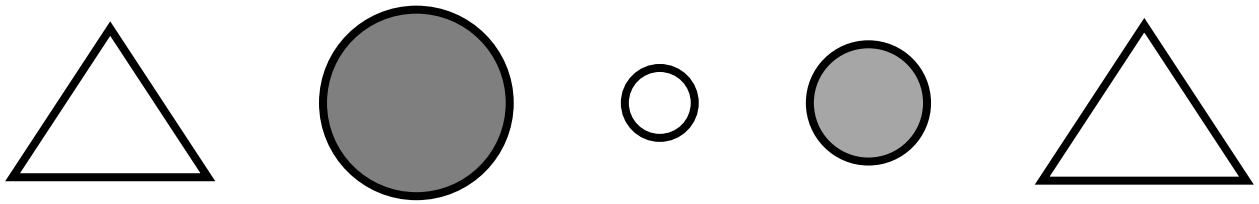


分類問題

円形と三角形を分類したい場合、どの特徴に着目すればいいのか？



x_1	3	0	3	0	3



x_1	3	0	0	0	3

角の数

円形の角は 0 個で、三角形の角の数は 3 個である。そのため、角の数を特徴量をして使えば、この 1 つの特徴量だけで円形と三角形を分けられるようになる。

回帰問題

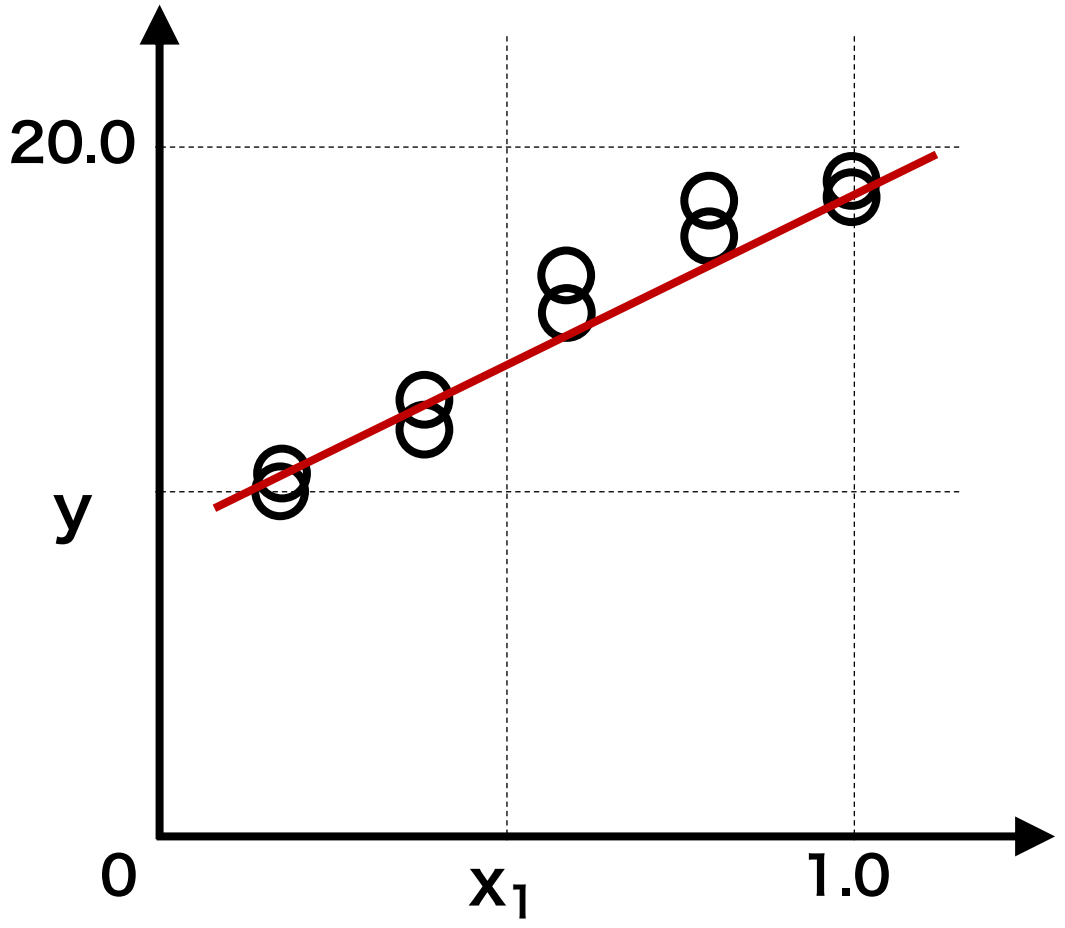
施肥量を利用して収穫量を予測するにはどうすればいいのか？

実験区画	区画 1	区画 2	区画 3	区画 4	区画 5
施肥量	0.2	0.4	0.6	0.8	1.0
収穫量					
	10.2	12.4	16.1	17.8	18.2
					
	10.1	11.9	17.2	18.1	18.0

回帰問題

施肥量を利用して収穫量を予測するにはどうすればいいのか？

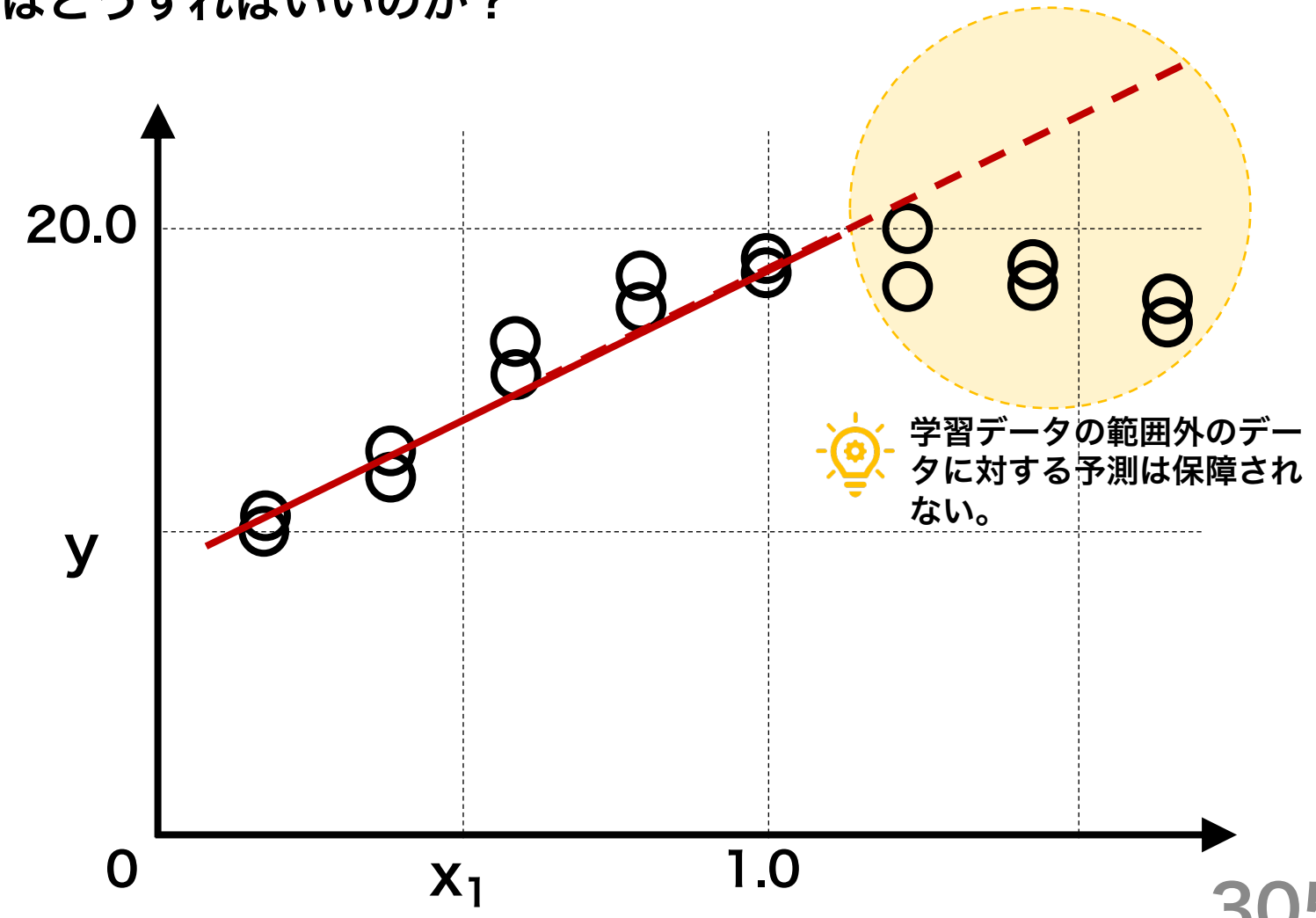
ラベル	X_1
10.2	0.2
10.1	0.2
12.4	0.4
11.9	0.4
16.1	0.6
17.2	0.6
17.8	0.8
18.1	0.8
18.2	1.0
18.0	1.0



回帰問題

施肥量を利用して収穫量を予測するにはどうすればいいのか？

ラベル	X_1
10.2	0.2
10.1	0.2
12.4	0.4
11.9	0.4
16.1	0.6
17.2	0.6
17.8	0.8
18.1	0.8
18.2	1.0
18.0	1.0



教師なし学習

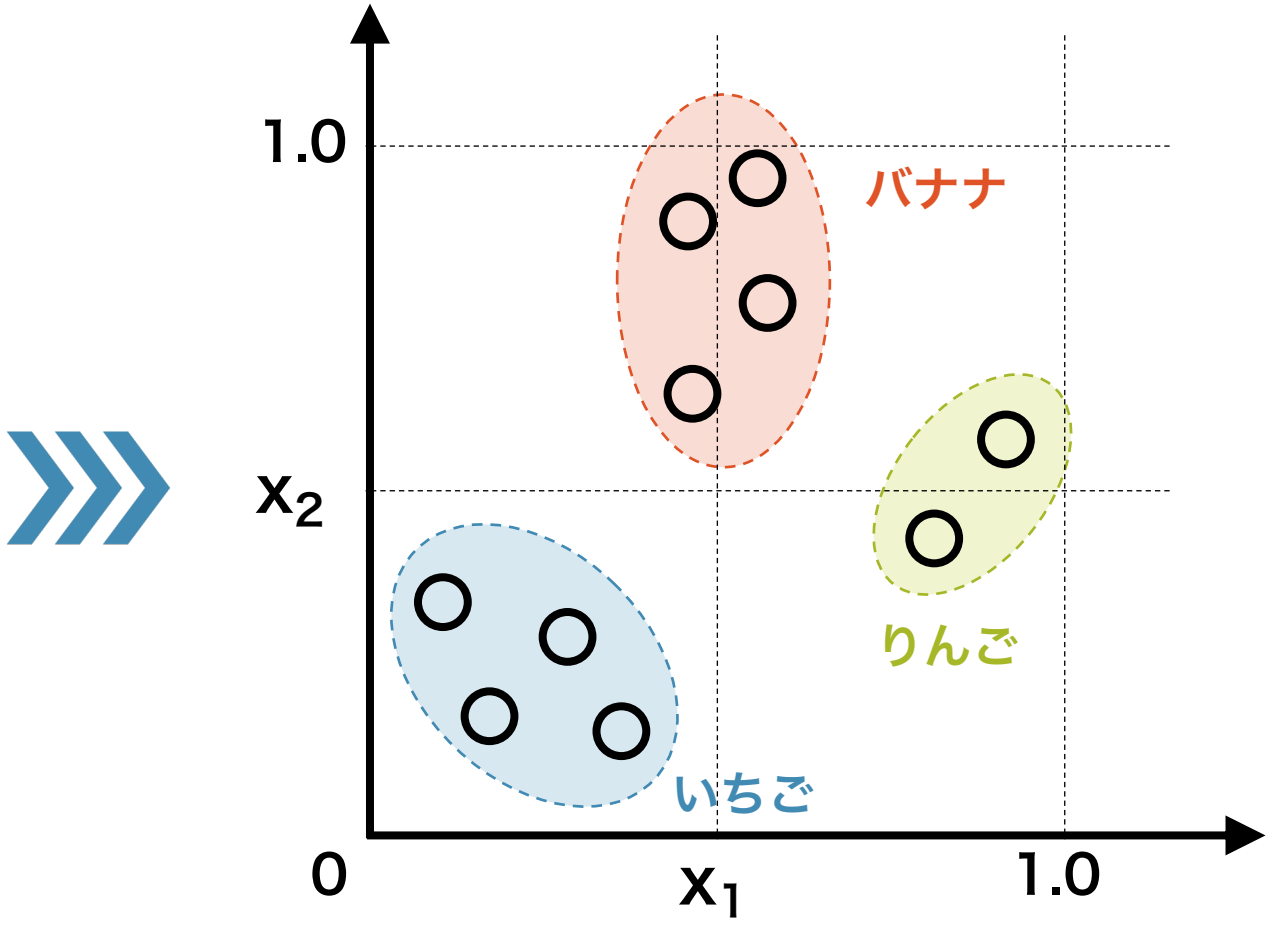
教師なし学習は、訓練データとして特徴量が与えられるときに行う学習方法である。機械が大量なデータを解釈し、データに隠されたパターンを抽出できるようになる。教師なし学習は、クラスタリングや次元削減・特徴抽出、外れ値検出などに適用される。教師あり学習で使われる学習アルゴリズムには、次のようなものがある。

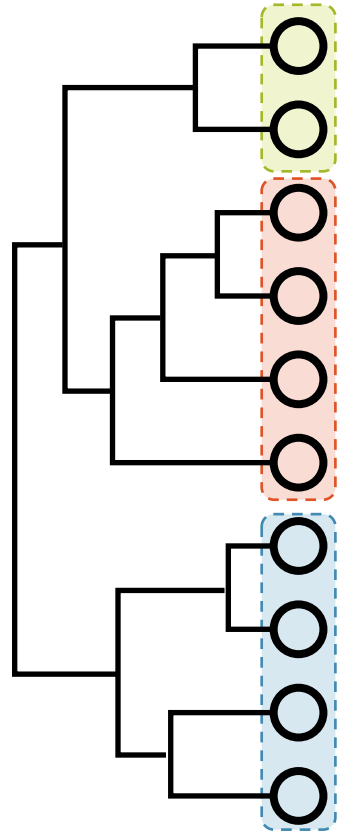
- 階層型クラスタリング
- k-means
- 自己組織化モデル（SOM）
- トピックモデル（pLSI, LDA）
- 主成分分析（PCA）
- 線形判別分析（LDA）

クラスタリング

10 個の果物に対して、長さ X_1 と重さ X_2 を測定した。果物間の関係を調べるにはどうすればいいのか？

X_1	X_2
0.12	0.38
0.92	0.57
0.39	0.21
0.52	0.98
0.49	0.88
0.26	0.25
0.34	0.36
0.49	0.61
0.53	0.73
0.82	0.46

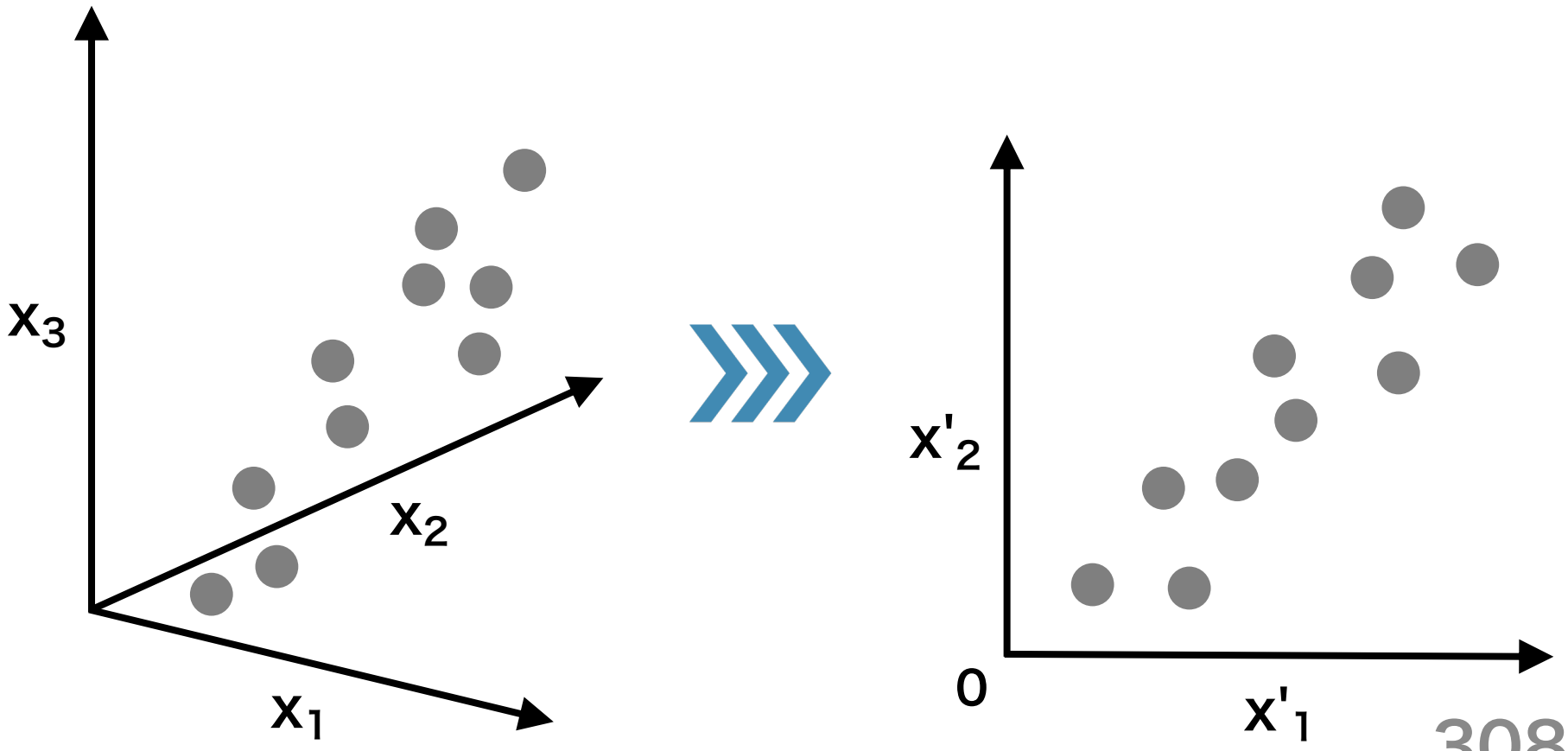




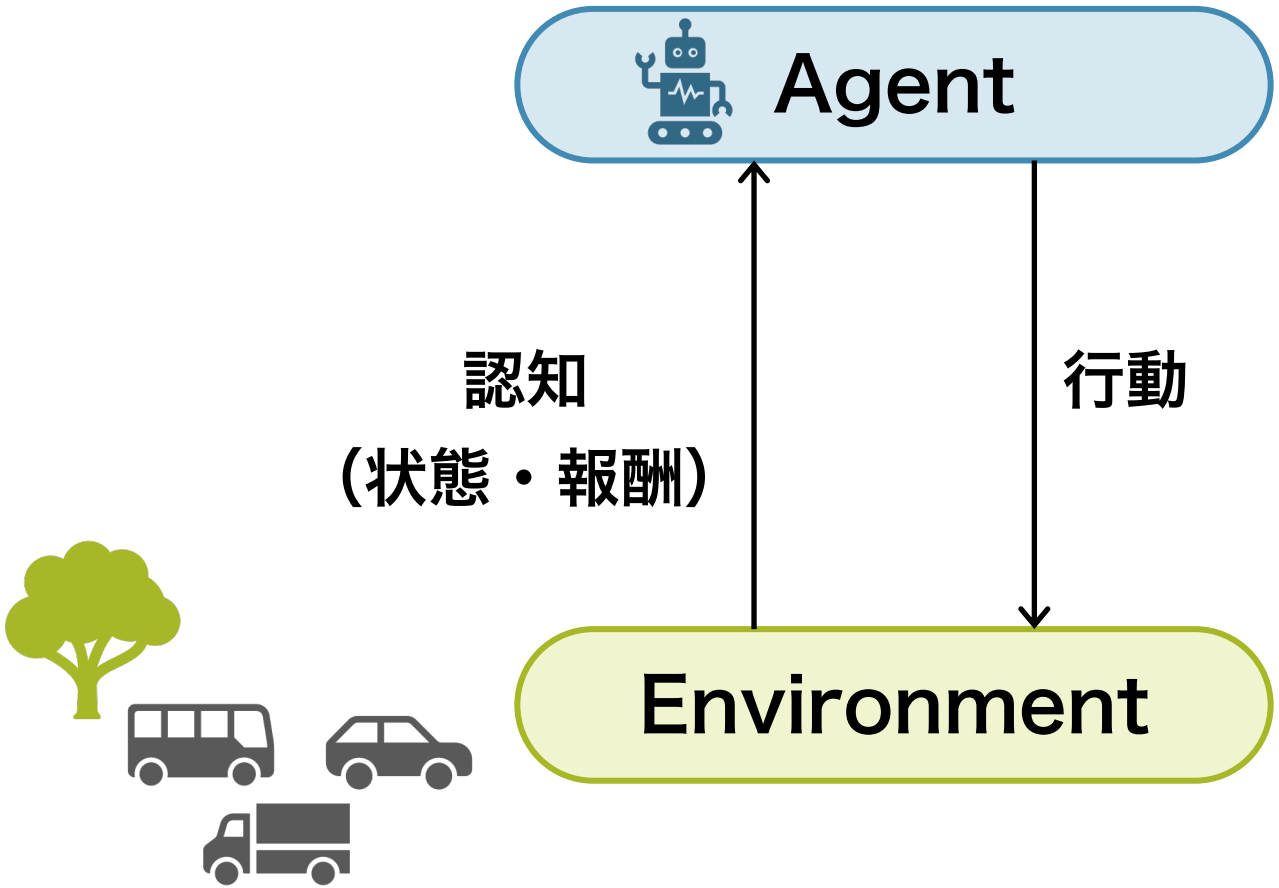
次元削減・特徴量抽出

多次元の情報を二次元の情報に落として図示したい。

X_1	X_2	X_3
0.12	0.38	0.54
0.92	0.57	0.12
0.39	0.21	0.42
0.52	0.98	0.85
0.49	0.88	0.24
0.26	0.25	0.53
0.34	0.36	0.87
0.49	0.61	0.42
0.53	0.73	0.13
0.82	0.46	0.56



強化学習



機械学習概略

- 人工知能と機械学習
- 開発ワークフロー
- ニューラルネットワーク

モデル構築

入力データを使用して、パラメーターチューニングを行い、最適なモデルを構築する。

データ収集

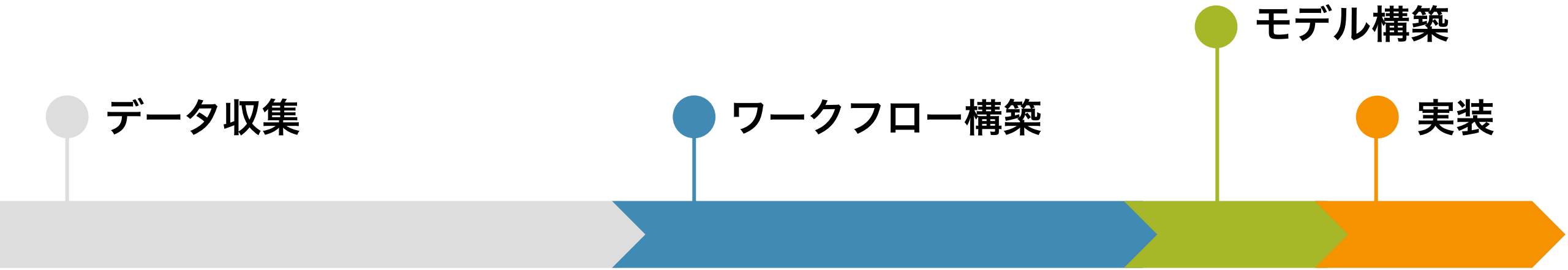
市場調査を行い、ニーズを確定し、モデル構築に必要なデータを収集する。

ワークフロー構築

収集されたデータに対して欠損値や異常値処理を行い、データを機械学習が使える形に整形し、機械学習アルゴリズムに渡すまでのワークフローを構築する。

実装

機械学習モデルをアプリ化し、スマホや専用機器に実装し、実用化する。



市場調査を行い、ニーズを確定し、モデル構築に必要なデータを収集する。

収集されたデータに対して欠損値や異常値処理を行い、データを機械学習が使える形に整形し、機械学習アルゴリズムに渡すまでのワークフローを構築する。

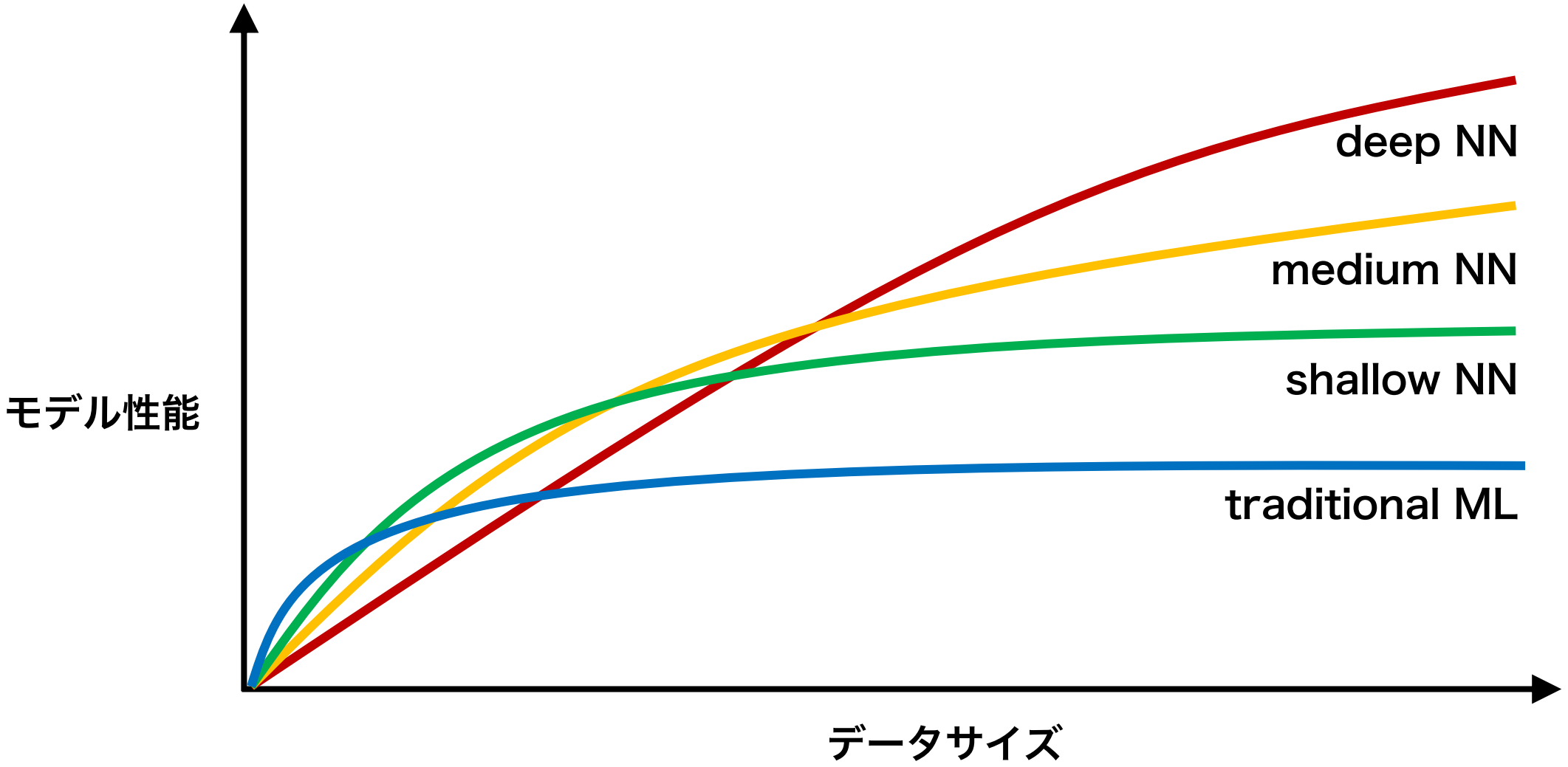
入力データを使用して、パラメーターチューニングを行い、最適なモデルを構築する。

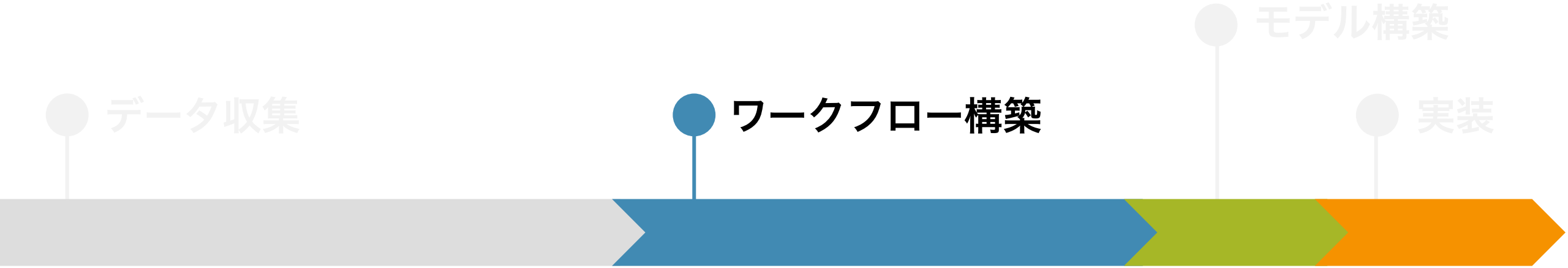
機械学習モデルをアプリ化し、スマホや専用機器に実装し、実用化する。

- 解決すべき問題を明確に定義する。
- 問題を解決するためのデータを収集する。
  - 機械学習の成否はデータの量と質に影響される。
  - 継続的なデータ収集フローを制定する。
- 手作業によるデータ分析を行い、効率的な機械学習アルゴリズムを検討する。

データの重要性

https://youtu.be/F1ka6a13S9I





- 開発時と運用時のシステム運用方法を検討する。
  - 運用時に収集されたデータを学習に再利用できるようになっているか。
  - 運用時に専門家によるチェック過程が組み込まれているか。
- 運用時のソフトウェア・ハードウェアを検討する。
  - どのようなユーザーが、どのような使い方を想定し、どのような API を必要とするのか。

- 単一のモデルだけで、問題をすべて解決できない。
  - 複数のモデルを組み合わせて１つの問題を解決することが多い。
- 既存のアルゴリズムを優先的に使う。
  - 既存のアルゴリズムは開発コストが少なく、応用実績も多い。
- 機械学習は必ず成功するとは限らない。

テキスト抽出モデル

EXIT

言語推定モデル

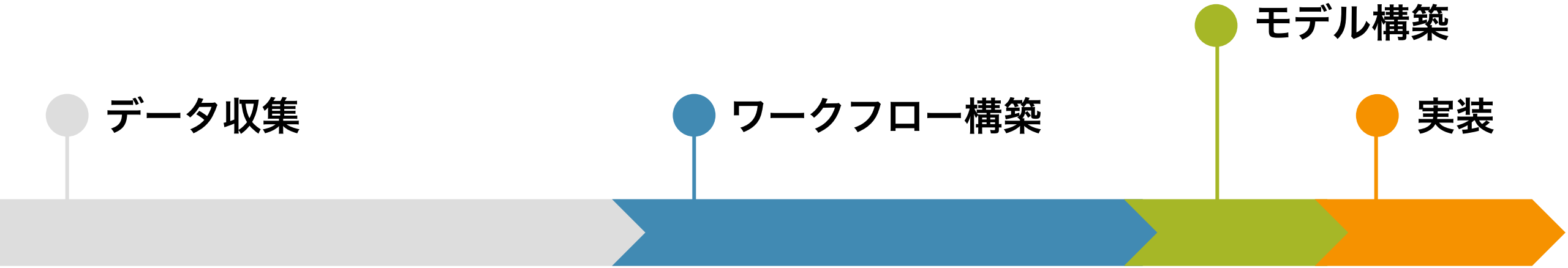
{English: EXIT}

翻訳モデル

{中文: 退出}

画像置換モデル

- アプリケーションへの実装は外注で行う。
  - 研究時間を確保するために、研究要素の少ない実装は外注する。
  - 開発会社の技術力が高く、使いやすくてセキュリティの堅い実装を可能にする。
- ユーザーによる評価やフィードバックを積極的に対応する。



市場調査を行い、ニーズを確定し、モデル構築に必要なデータを収集する。

収集されたデータに対して欠損値や異常値処理を行い、データを機械学習が使える形に整形し、機械学習アルゴリズムに渡すまでのワークフローを構築する。

入力データを使用して、パラメーターチューニングを行い、最適なモデルを構築する。

機械学習モデルをアプリ化し、スマホや専用機器に実装し、実用化する。

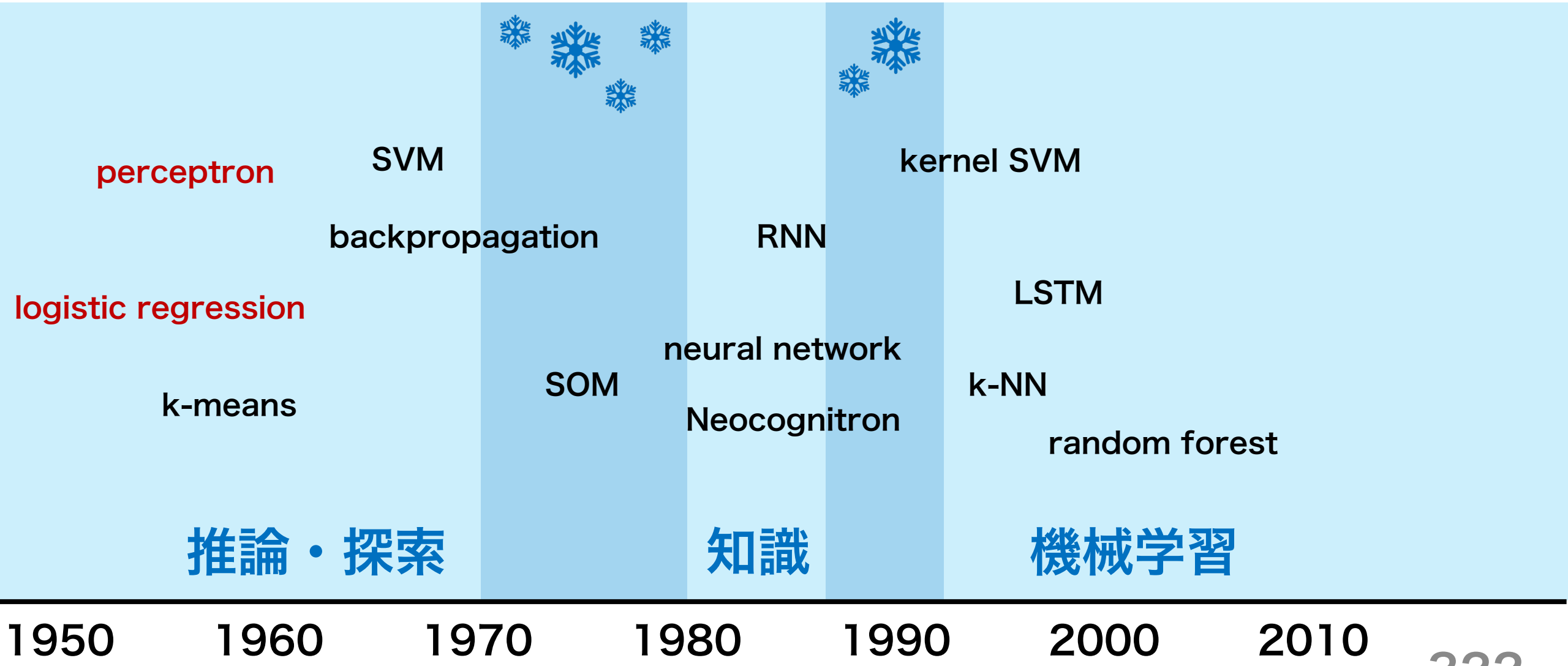
機械学習概略

- 人工知能と機械学習
- 開発ワークフロー
- ニューラルネットワーク

ニューラルネットワーク

- パーセプトロン
- ニューラルネットワーク
- CNN
- RNN

機械学習アルゴリズム

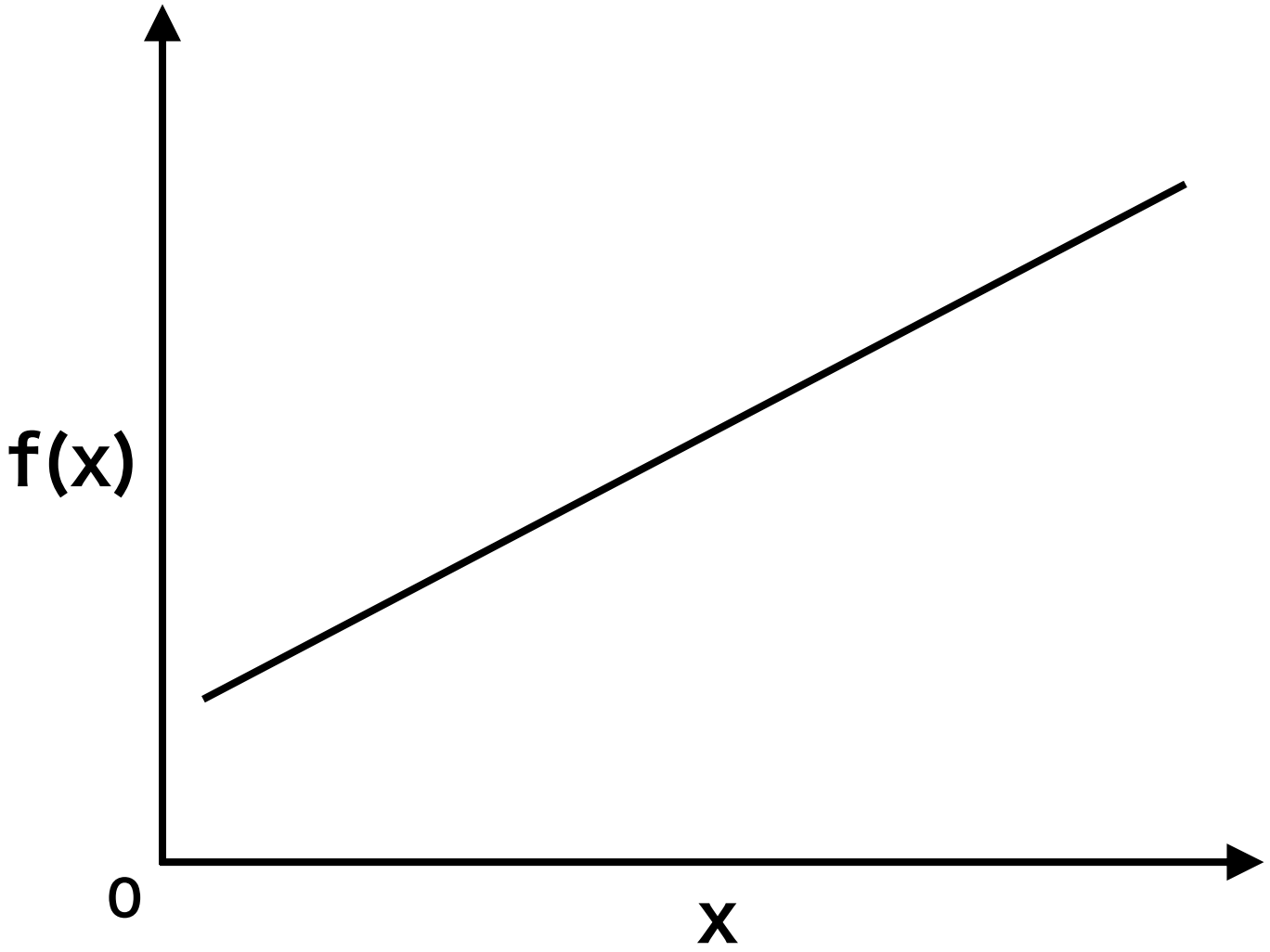

perceptron
SVM
kernel SVM
backpropagation
RNN
logistic regression
LSTM
neural network
SOM
k-NN
k-means
Neocognitron
random forest
推論・探索
知識
機械学習
1950
1960
1970
1980
1990
2000
2010

関数

１次関数 f を考える。f は入力値 x を受け取り、x に対して何らかな演算を行なって、その結果を返す関数となる。例えば、気温 x を関数 f を与えると、関数の中で気温に対して何らかの演算を行なって、穀物の収量を出力する。

$$f(x) = w_0 + w_1 x$$

▲ 収量　　▲ 気温

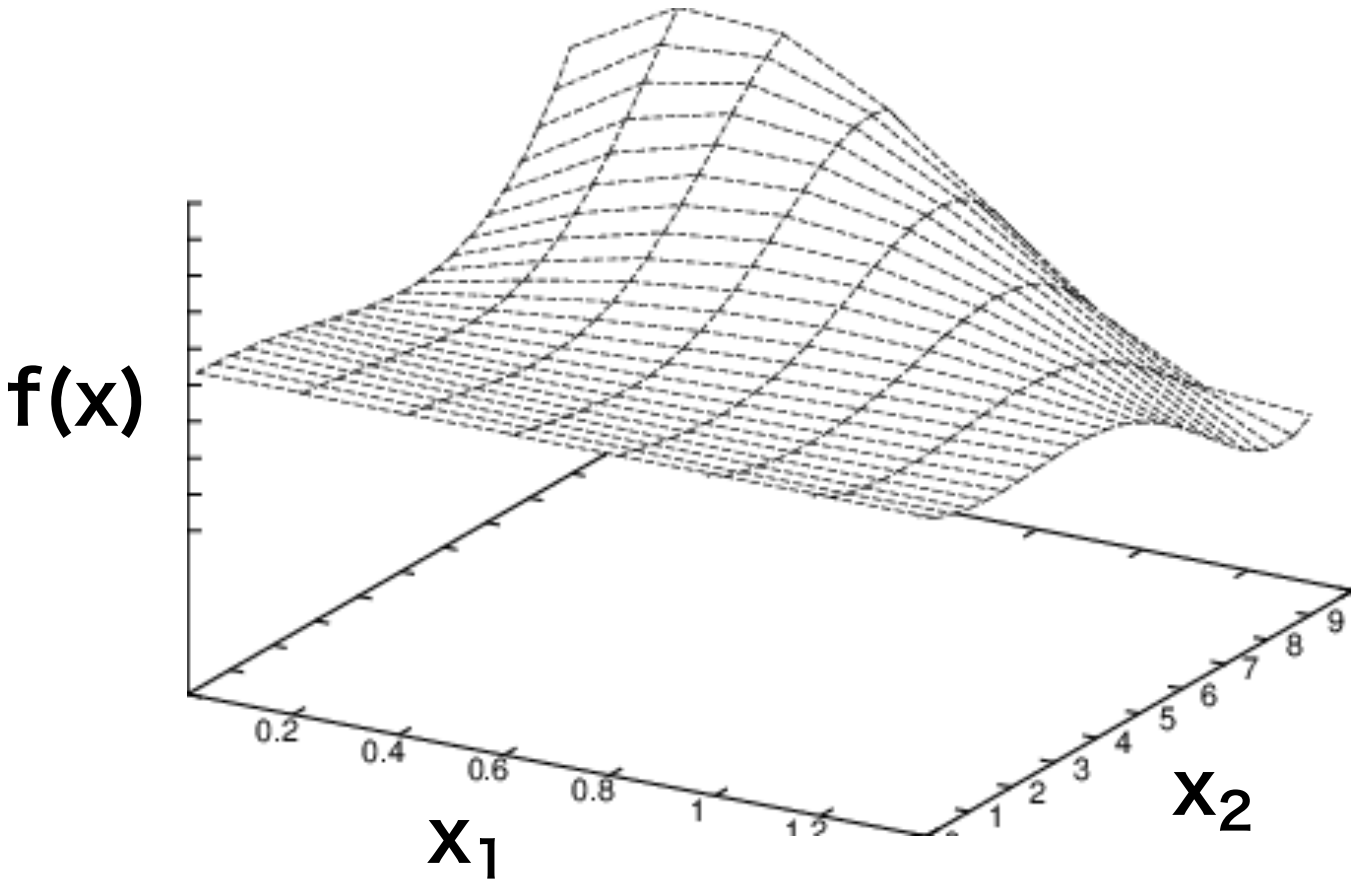


多変量関数

複数の値を受け取り、1 つの値を返す関数を多変量関数という。例えば、穀物の収量は、気温だけでなく、施肥量にも影響される。そこで、気温 x_1 と施肥量 x_2 を関数 f に与えると、関数の中で気温と施肥量に対して何らかの演算を行なって、穀物の収量を出力する。

$$f(x) = w_0 + w_1 x_1 + w_2 x_2$$

▲ 収量　▲ 気温　▲ 施肥量



パーセプトロン

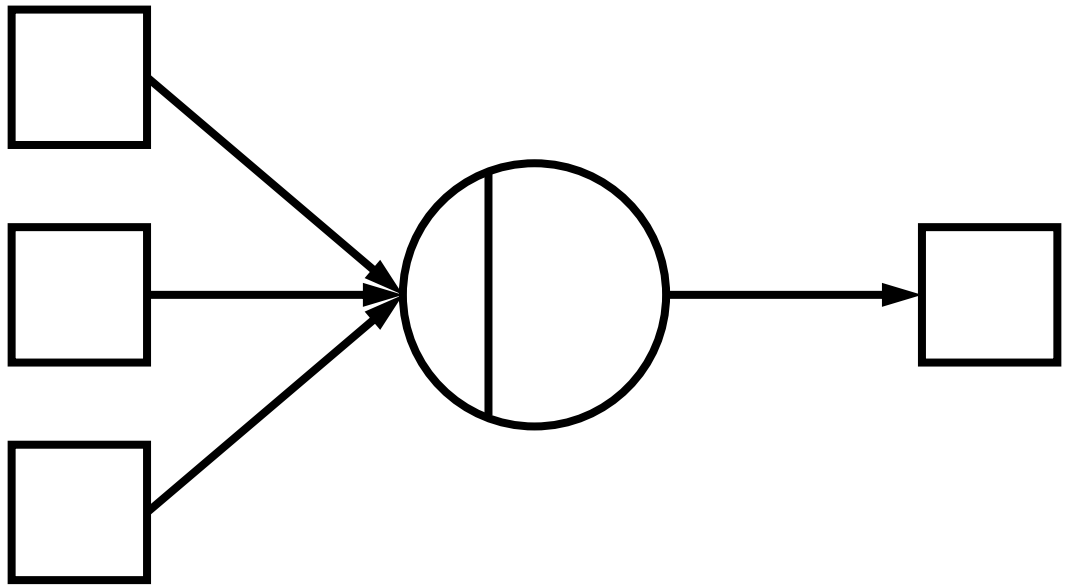
パーセプトロンは、複数の入力値に重みをかけ、その和を計算し、その和の符号に応じて値を出力するアルゴリズムである。

入力　x_1　x_2

処理　$z = w_0 + w_1 x_1 + w_2 x_2$

処理　$$\phi(z) = \begin{cases} 0, & z < 0 \\ 1, & z \geq 0 \end{cases}$$

出力　1



パーセプトロン

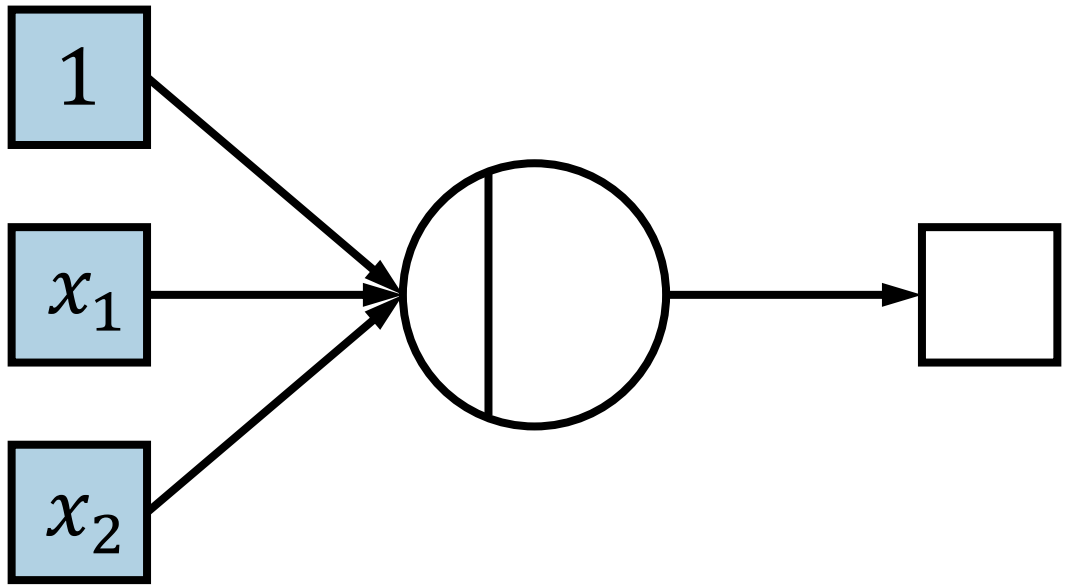
パーセプトロンは、複数の入力値に重みをかけ、その和を計算し、その和の符号に応じて値を出力するアルゴリズムである。

入力　x_1　x_2

処理　$z = w_0 + w_1 x_1 + w_2 x_2$

処理　$$\phi(z) = \begin{cases} 0, & z < 0 \\ 1, & z \geq 0 \end{cases}$$

出力　1



パーセプトロン

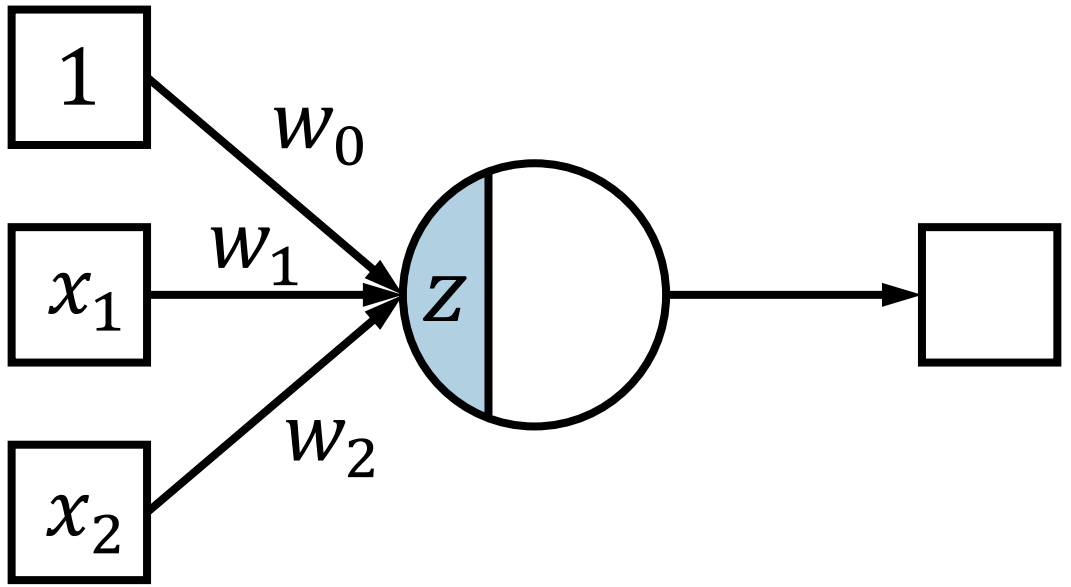
パーセプトロンは、複数の入力値に重みをかけ、その和を計算し、その和の符号に応じて値を出力するアルゴリズムである。

入力　x_1　x_2

処理　$z = w_0 + w_1 x_1 + w_2 x_2$

処理　$$\phi(z) = \begin{cases} 0, & z < 0 \\ 1, & z \geq 0 \end{cases}$$

出力　1



パーセプトロン

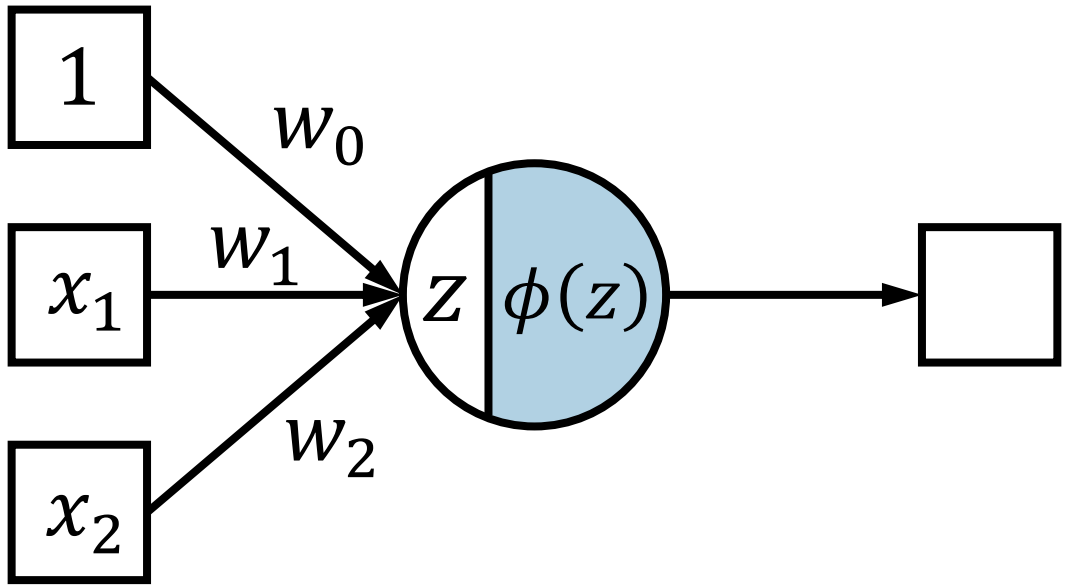
パーセプトロンは、複数の入力値に重みをかけ、その和を計算し、その和の符号に応じて値を出力するアルゴリズムである。

入力　x_1　x_2

処理　$z = w_0 + w_1 x_1 + w_2 x_2$

処理　$\phi(z) = \begin{cases} 0, & z < 0 \\ 1, & z \geq 0 \end{cases}$

出力　1



パーセプトロン

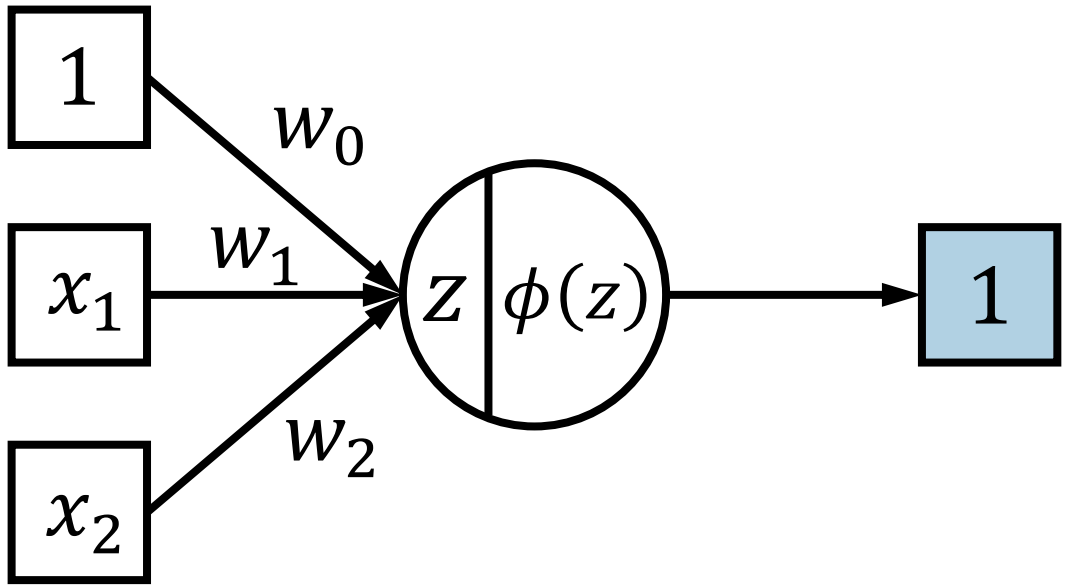
パーセプトロンは、複数の入力値に重みをかけ、その和を計算し、その和の符号に応じて値を出力するアルゴリズムである。

入力　x_1　x_2

処理　$z = w_0 + w_1 x_1 + w_2 x_2$

処理　$\phi(z) = \begin{cases} 0, & z < 0 \\ 1, & z \geq 0 \end{cases}$

出力　1

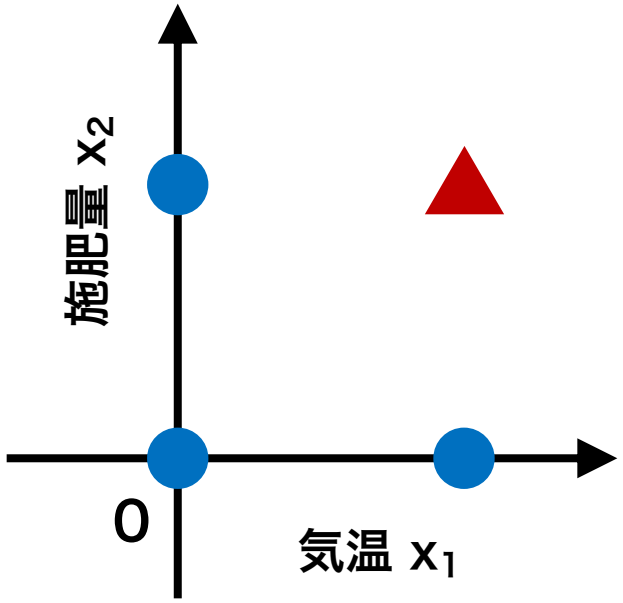


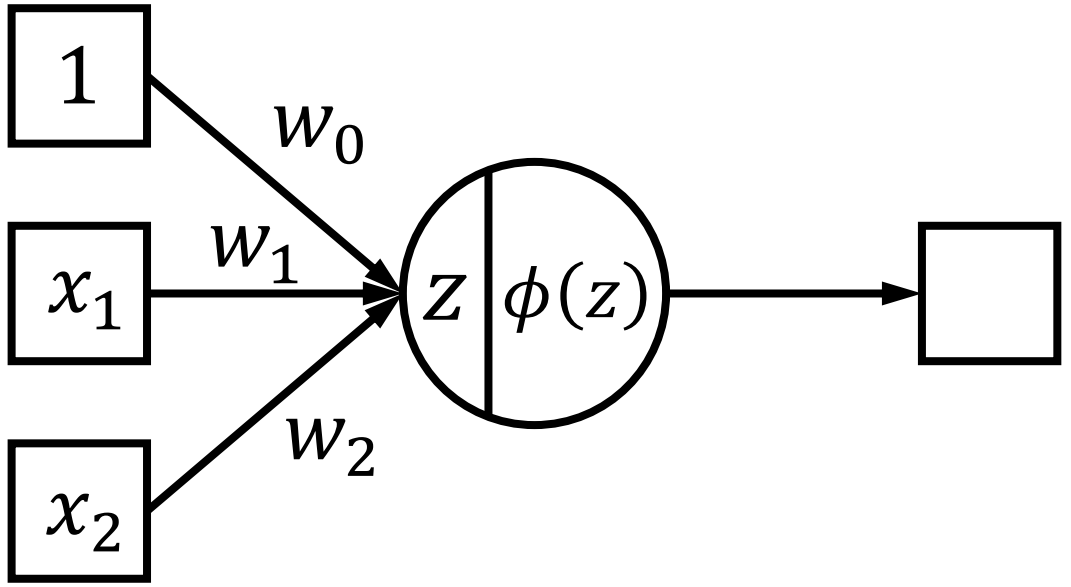
パーセプトロン学習則

学習データ

気温・施肥量を使って開花予測を行うために、フィールド調査を行い次のようなデータを得た。

気温	施肥量	開花
0	0	0
1	0	0
0	1	0
1	1	1





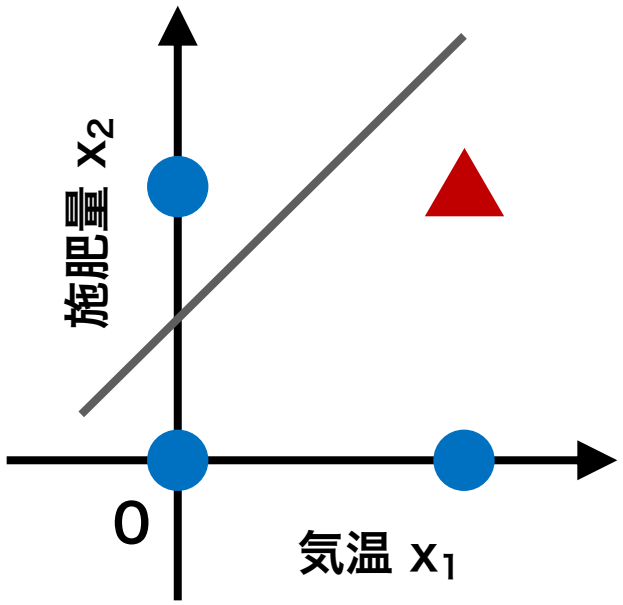
パーセプトロン学習則

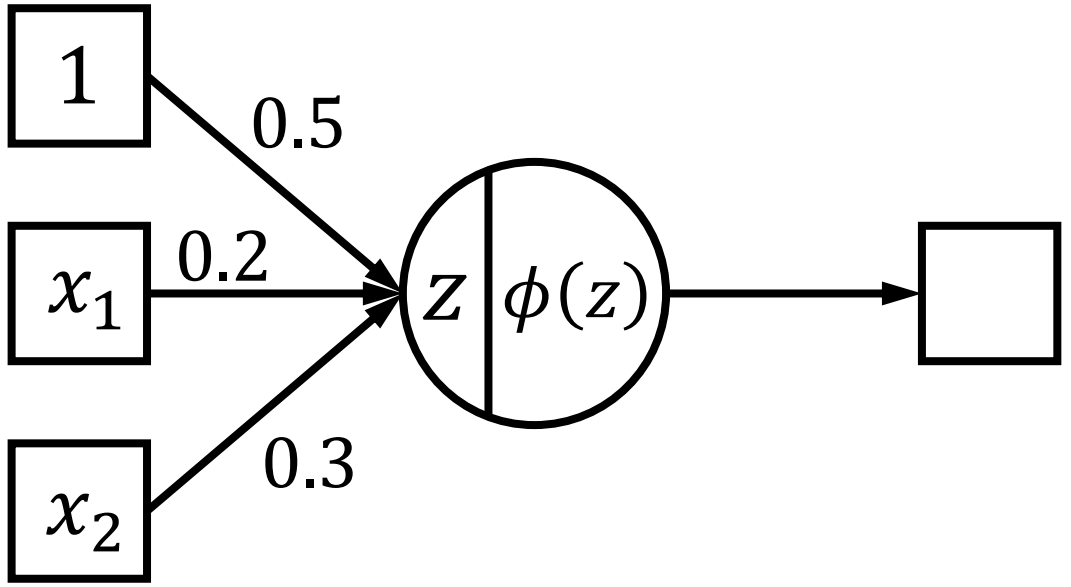
学習データ

気温・施肥量を使って開花予測を行うために、フィールド調査を行い次のようなデータを得た。

気温	施肥量	開花
0	0	0
1	0	0
0	1	0
1	1	1

※図はイメージである。





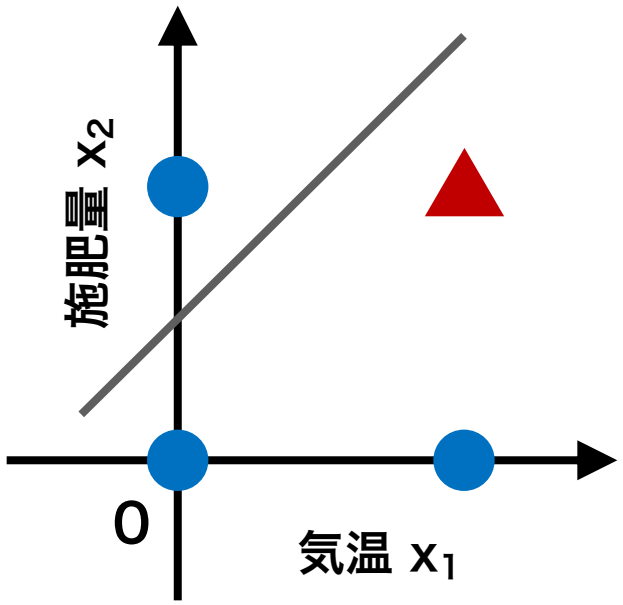
パーセプトロン学習則

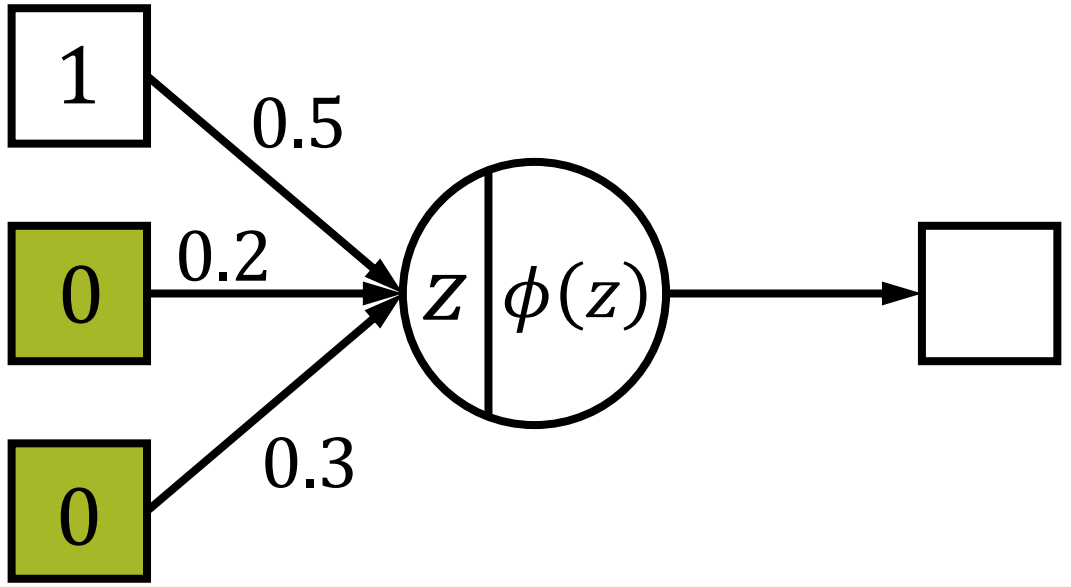
学習データ

気温・施肥量を使って開花予測を行うために、フィールド調査を行い次のようなデータを得た。

気温	施肥量	開花
0	0	0
1	0	0
0	1	0
1	1	1

※図はイメージである。





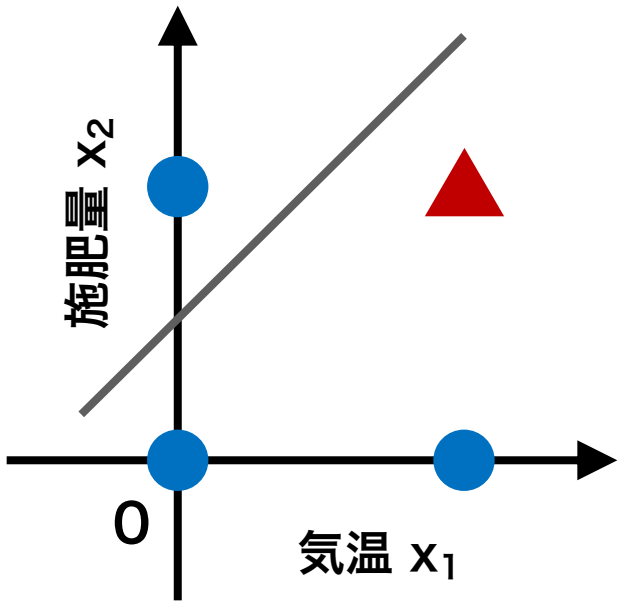
パーセプトロン学習則

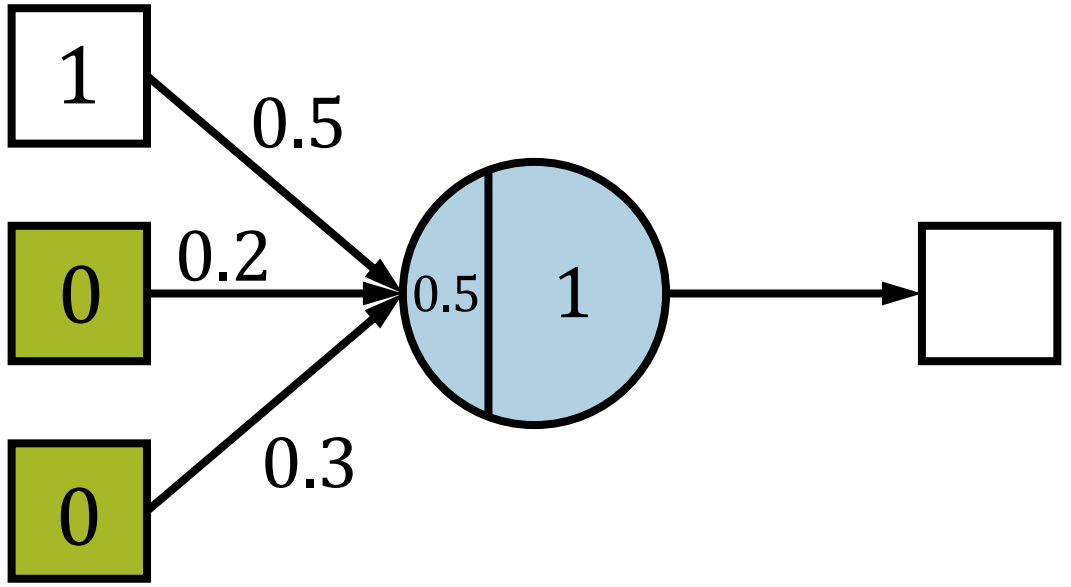
学習データ

気温・施肥量を使って開花予測を行うために、フィールド調査を行い次のようなデータを得た。

気温	施肥量	開花
0	0	0
1	0	0
0	1	0
1	1	1

※図はイメージである。





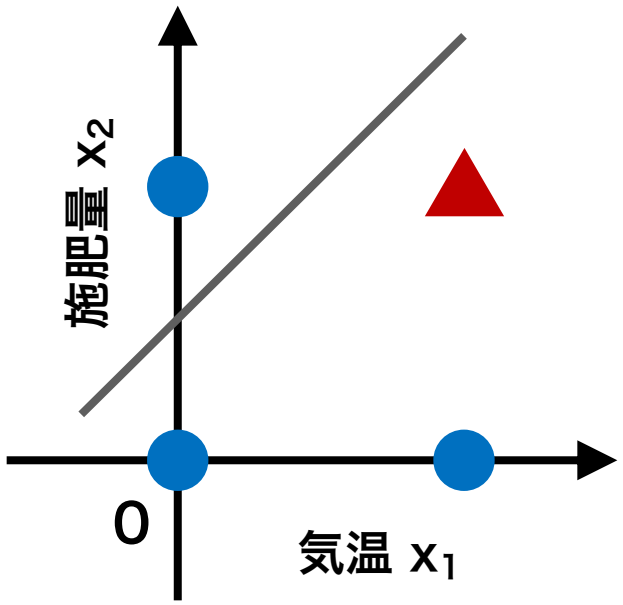
パーセプトロン学習則

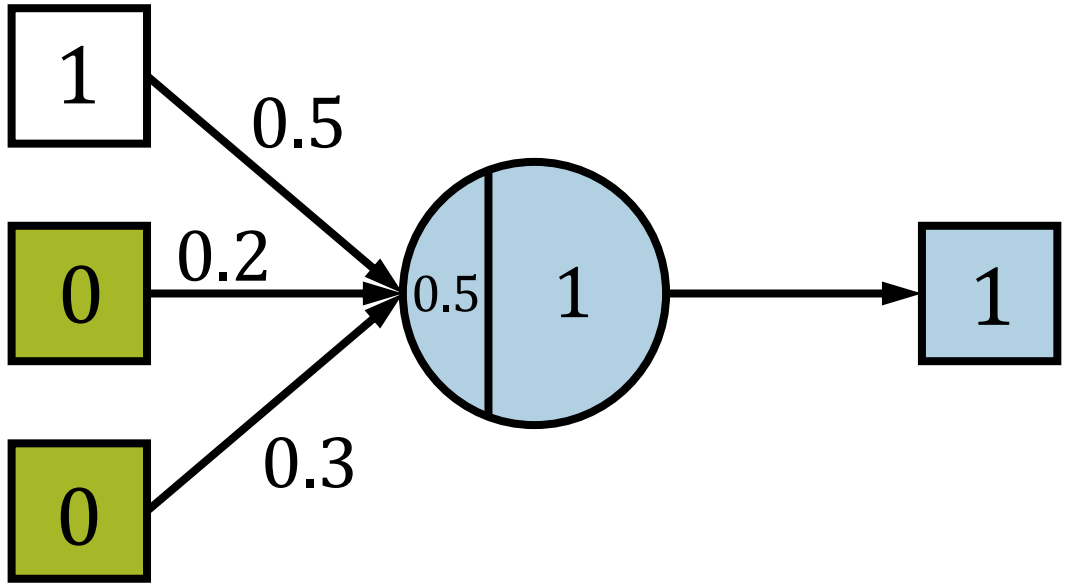
学習データ

気温・施肥量を使って開花予測を行うために、フィールド調査を行い次のようなデータを得た。

気温	施肥量	開花
0	0	0
1	0	0
0	1	0
1	1	1

※図はイメージである。





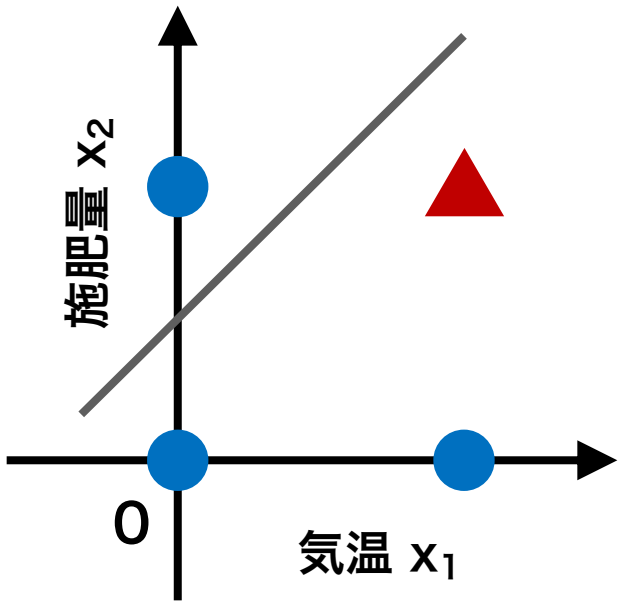
パーセプトロン学習則

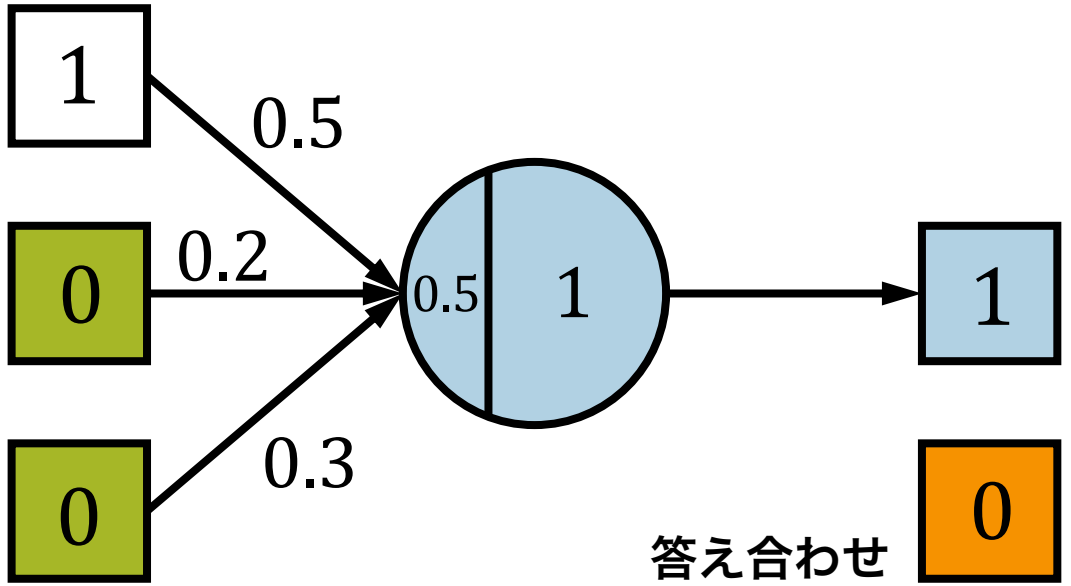
学習データ

気温・施肥量を使って開花予測を行うために、フィールド調査を行い次のようなデータを得た。

気温	施肥量	開花
0	0	0
1	0	0
0	1	0
1	1	1

※図はイメージである。





損失　1

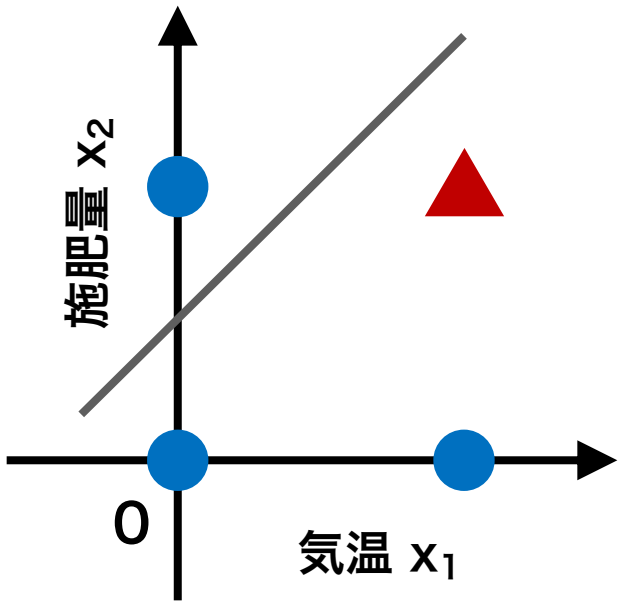
パーセプトロン学習則

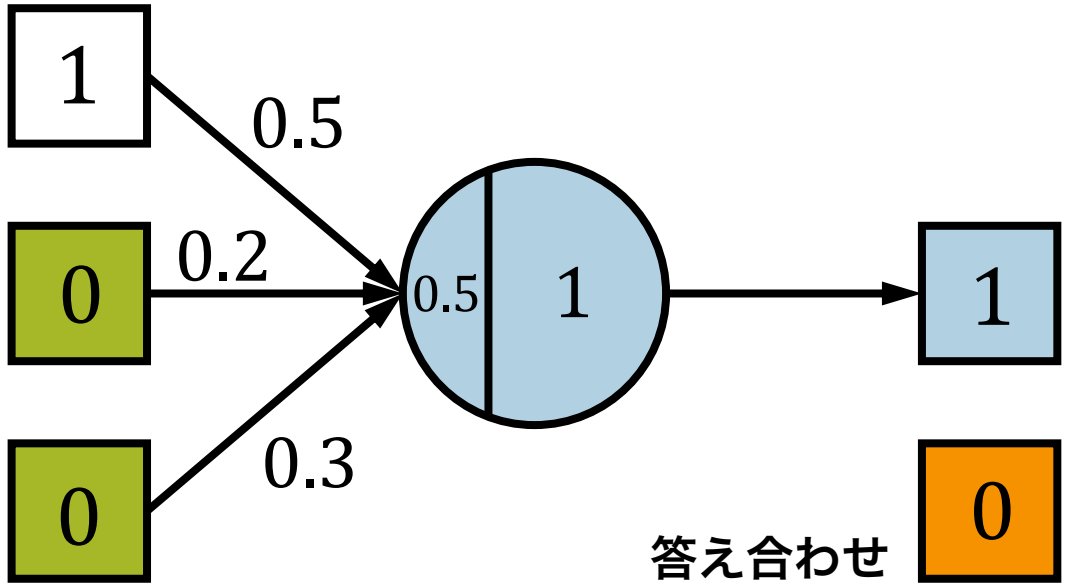
学習データ

気温・施肥量を使って開花予測を行うために、フィールド調査を行い次のようなデータを得た。

気温	施肥量	開花
0	0	0
1	0	0
0	1	0
1	1	1

※図はイメージである。





$$w_0^{(new)} = 0.5 + 1(0-1)0.1 = 0.4$$

$$w_1^{(new)} = 0.2 + 0(0-1)0.1 = 0.2$$

$$w_2^{(new)} = 0.3 + 0(0-1)0.1 = 0.3$$

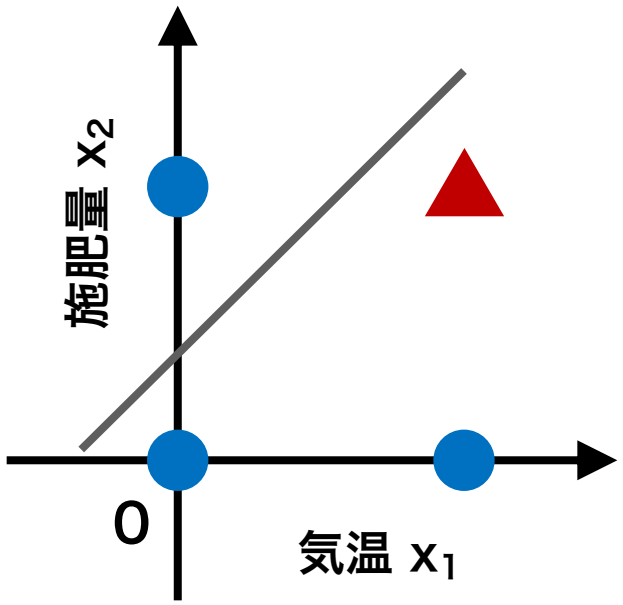
パーセプトロン学習則

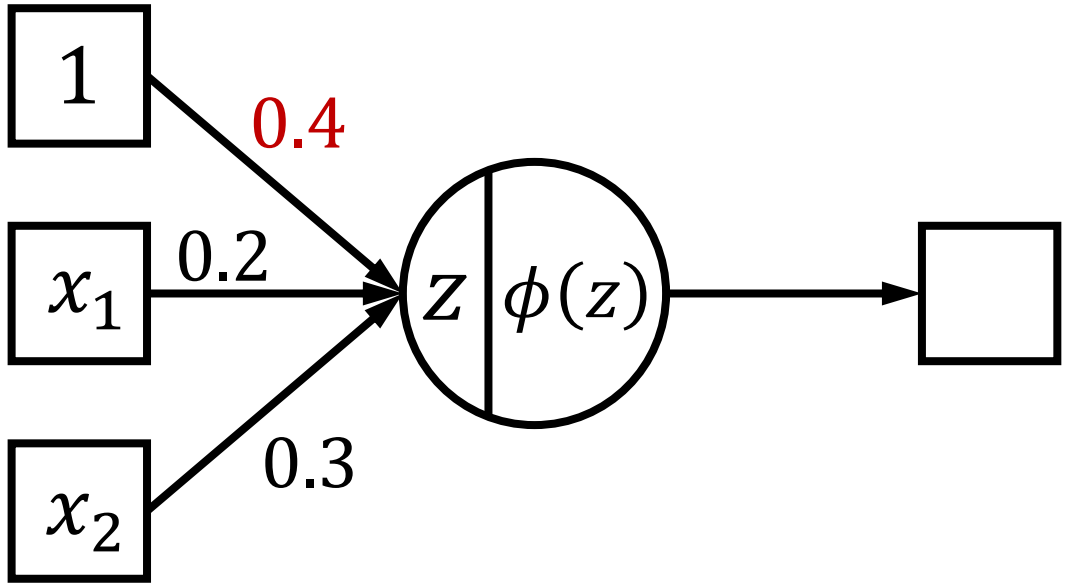
学習データ

気温・施肥量を使って開花予測を行うために、フィールド調査を行い次のようなデータを得た。

気温	施肥量	開花
0	0	0
1	0	0
0	1	0
1	1	1

※図はイメージである。





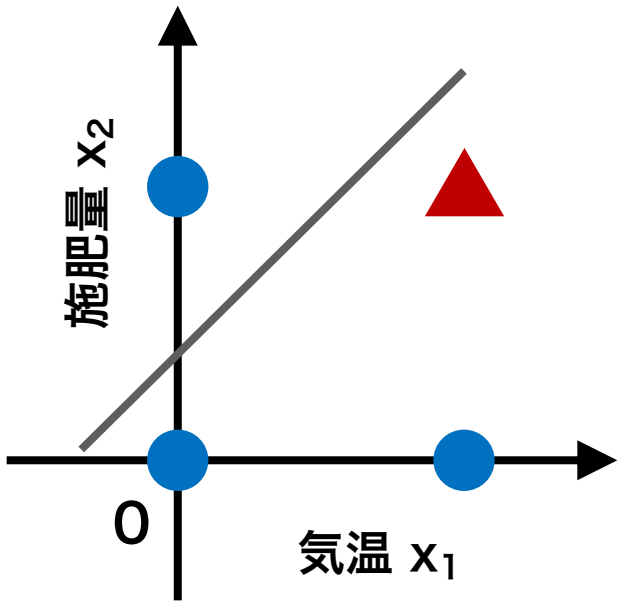
パーセプトロン学習則

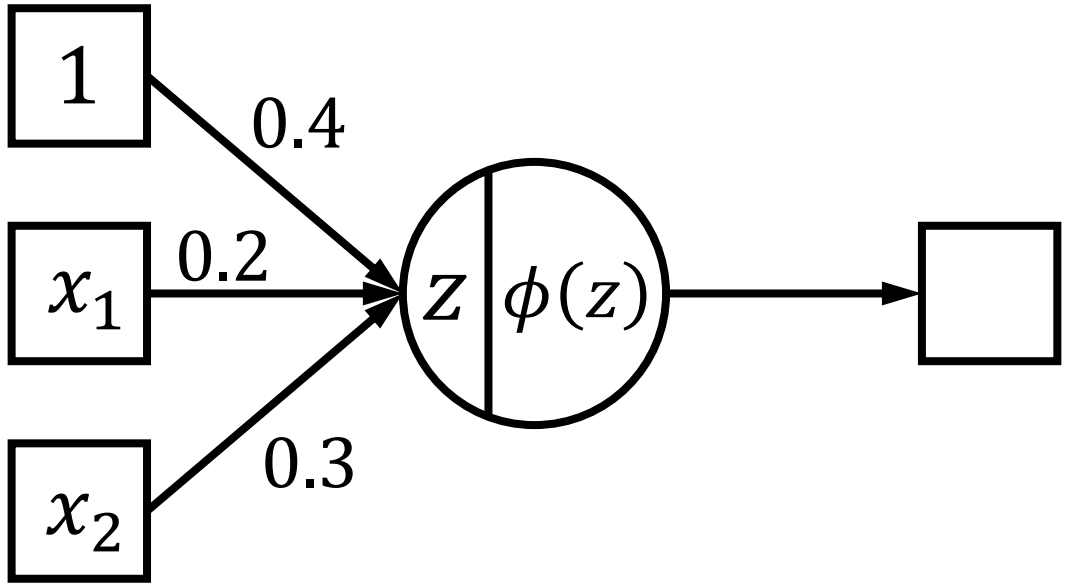
学習データ

気温・施肥量を使って開花予測を行うために、フィールド調査を行い次のようなデータを得た。

気温	施肥量	開花
0	0	0
1	0	0
0	1	0
1	1	1

※図はイメージである。





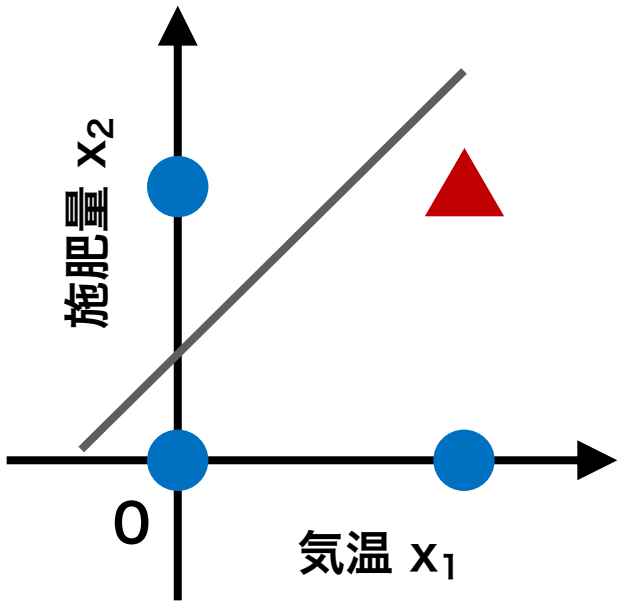
パーセプトロン学習則

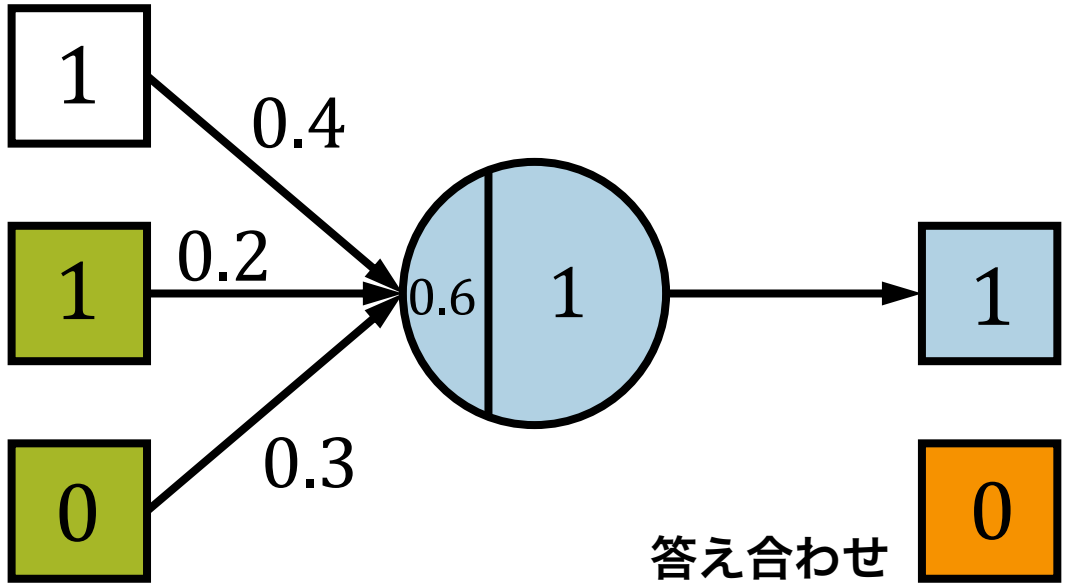
学習データ

気温・施肥量を使って開花予測を行うために、フィールド調査を行い次のようなデータを得た。

気温	施肥量	開花
0	0	0
1	0	0
0	1	0
1	1	1

※図はイメージである。





$$w_0^{(new)} = 0.4 + 1(0-1)0.1 = 0.3$$

$$w_1^{(new)} = 0.2 + 1(0-1)0.1 = 0.1$$

$$w_2^{(new)} = 0.3 + 0(0-1)0.1 = 0.3$$

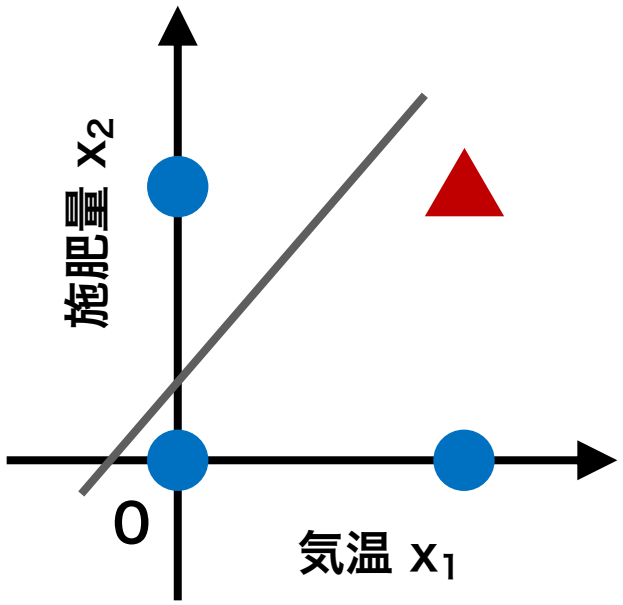
パーセプトロン学習則

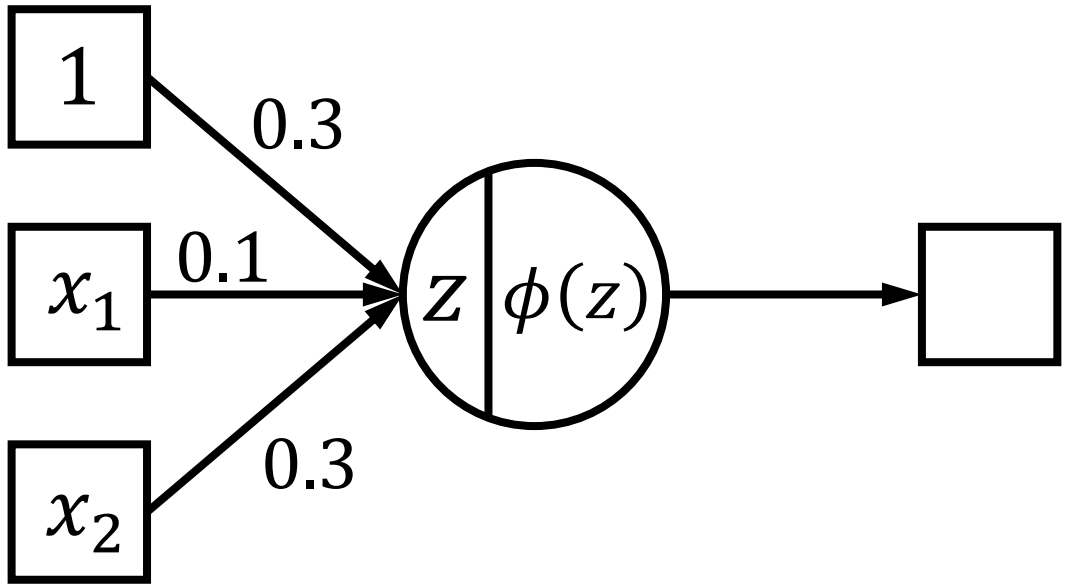
学習データ

気温・施肥量を使って開花予測を行うために、フィールド調査を行い次のようなデータを得た。

気温	施肥量	開花
0	0	0
1	0	0
0	1	0
1	1	1

※図はイメージである。





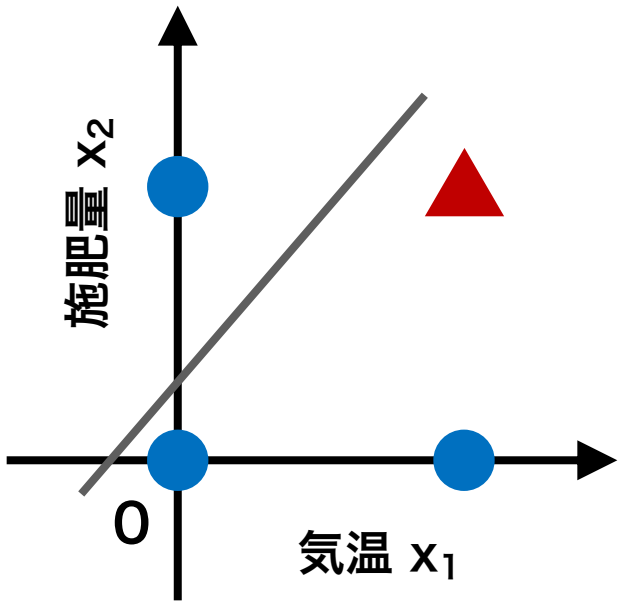
パーセプトロン学習則

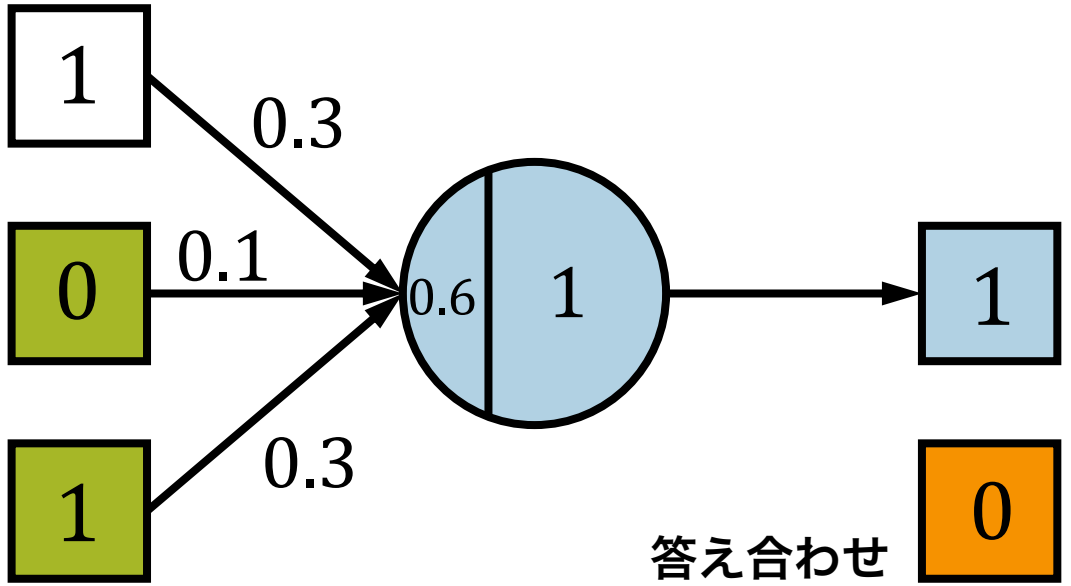
学習データ

気温・施肥量を使って開花予測を行うために、フィールド調査を行い次のようなデータを得た。

気温	施肥量	開花
0	0	0
1	0	0
0	1	0
1	1	1

※図はイメージである。





$$w_0^{(new)} = 0.3 + 1(0-1)0.1 = 0.2$$

$$w_1^{(new)} = 0.1 + 0(0-1)0.1 = 0.1$$

$$w_2^{(new)} = 0.3 + 1(0-1)0.1 = 0.2$$

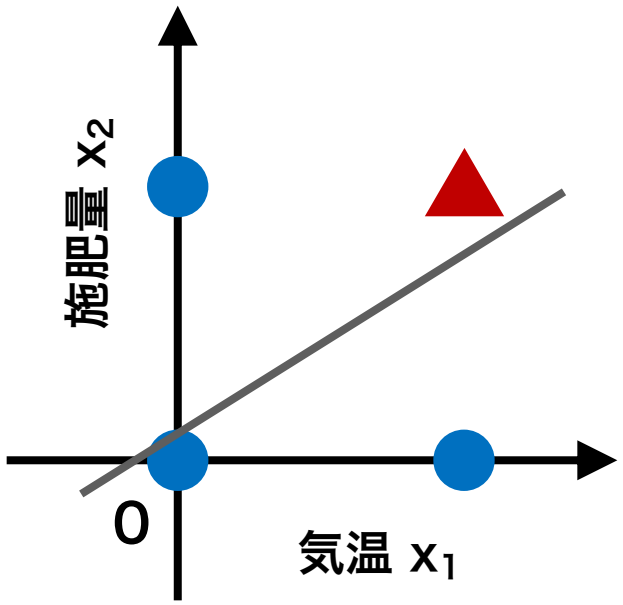
パーセプトロン学習則

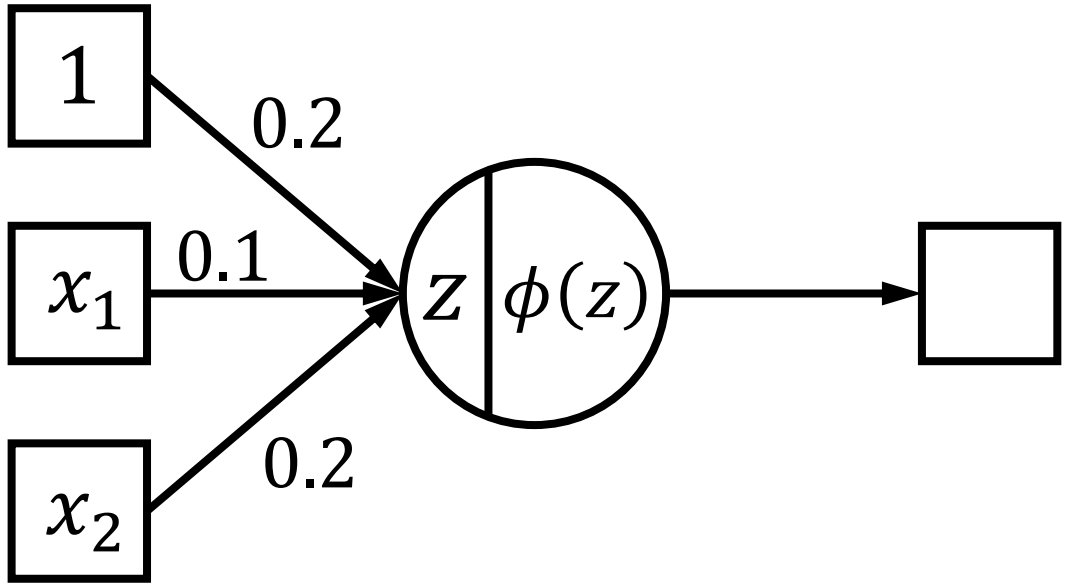
学習データ

気温・施肥量を使って開花予測を行うために、フィールド調査を行い次のようなデータを得た。

気温	施肥量	開花
0	0	0
1	0	0
0	1	0
1	1	1

※図はイメージである。





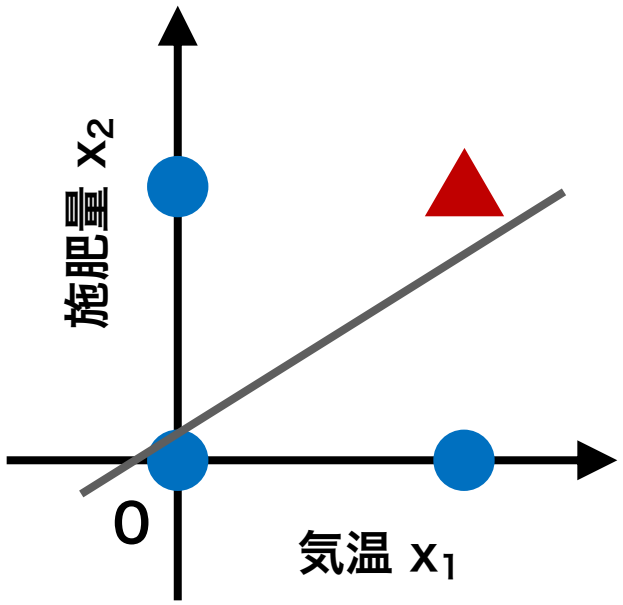
パーセプトロン学習則

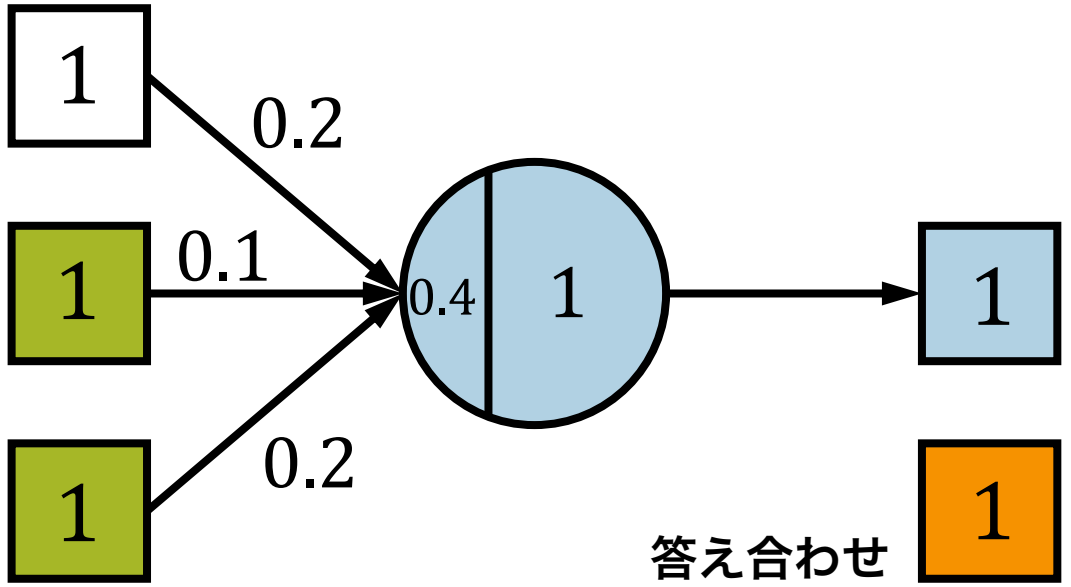
学習データ

気温・施肥量を使って開花予測を行うために、フィールド調査を行い次のようなデータを得た。

気温	施肥量	開花
0	0	0
1	0	0
0	1	0
1	1	1

※図はイメージである。





$$w_0^{(new)} = 0.2 + 1(1-1)0.1 = 0.2$$

$$w_1^{(new)} = 0.1 + 1(1-1)0.1 = 0.1$$

$$w_2^{(new)} = 0.2 + 1(1-1)0.1 = 0.2$$

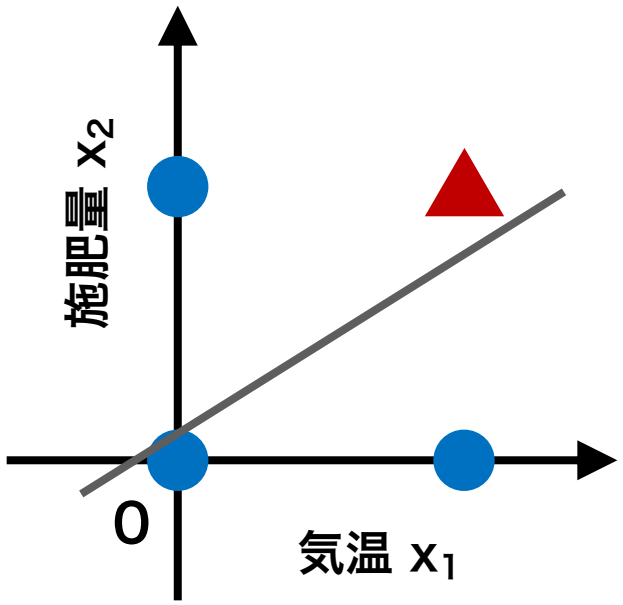
パーセプトロン学習則

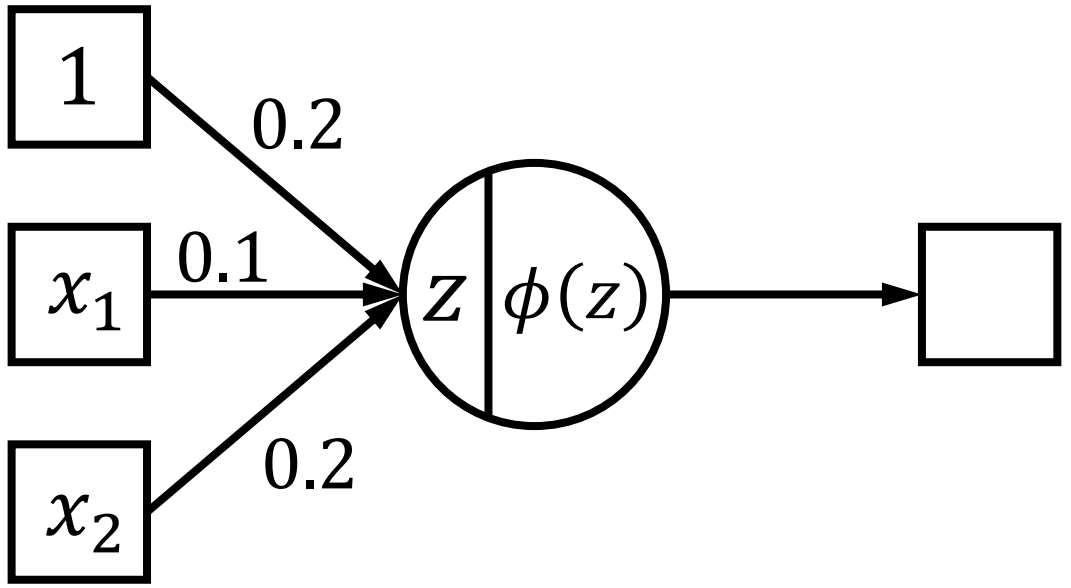
学習データ

気温・施肥量を使って開花予測を行うために、フィールド調査を行い次のようなデータを得た。

気温	施肥量	開花
0	0	0
1	0	0
0	1	0
1	1	1

※図はイメージである。





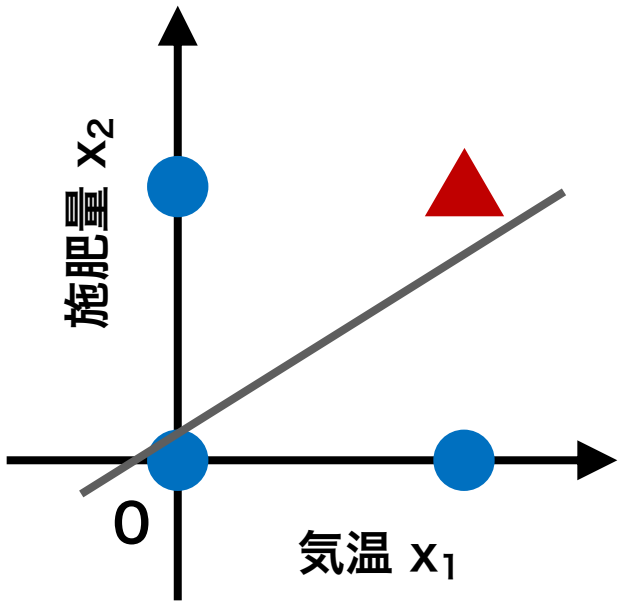
パーセプトロン学習則

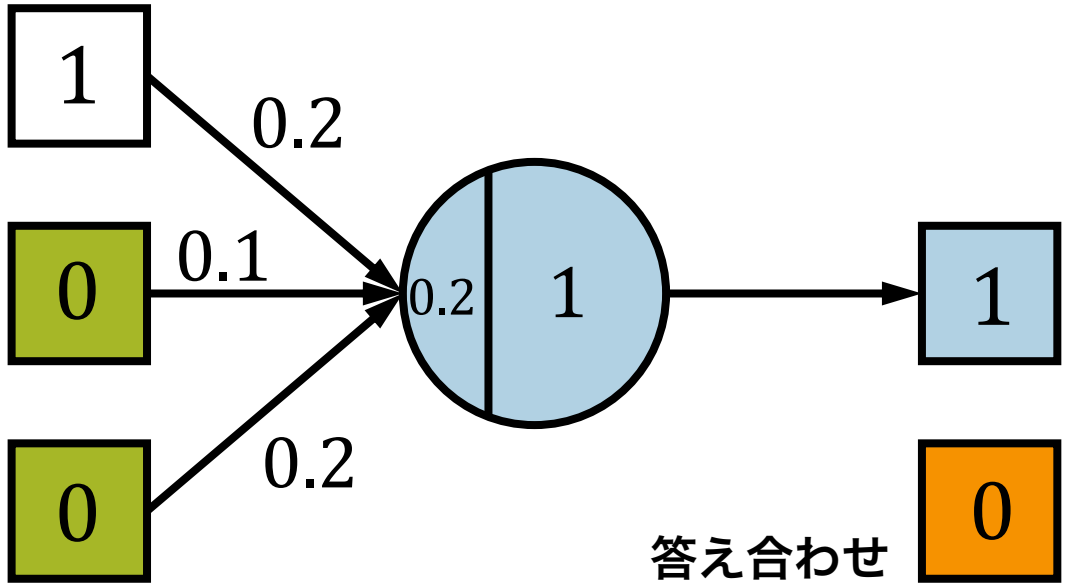
学習データ

気温・施肥量を使って開花予測を行うために、フィールド調査を行い次のようなデータを得た。

気温	施肥量	開花
0	0	0
1	0	0
0	1	0
1	1	1

※図はイメージである。





$$w_0^{(new)} = 0.2 + 1(0-1)0.1 = 0.1$$

$$w_1^{(new)} = 0.1 + 0(0-1)0.1 = 0.1$$

$$w_2^{(new)} = 0.2 + 0(0-1)0.1 = 0.2$$

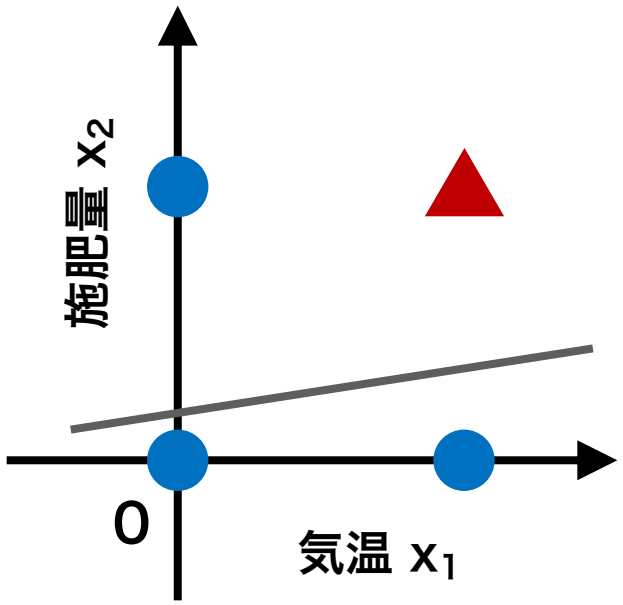
パーセプトロン学習則

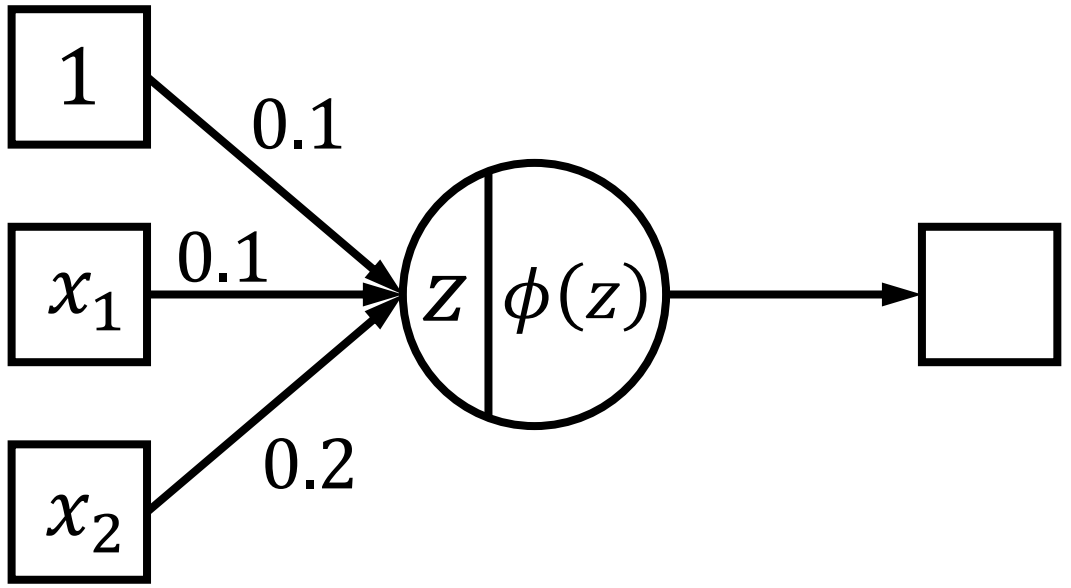
学習データ

気温・施肥量を使って開花予測を行うために、フィールド調査を行い次のようなデータを得た。

気温	施肥量	開花
0	0	0
1	0	0
0	1	0
1	1	1

※図はイメージである。





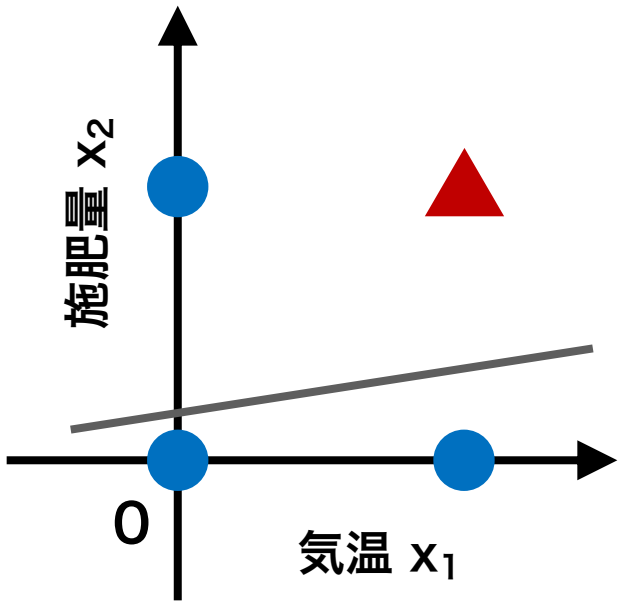
パーセプトロン学習則

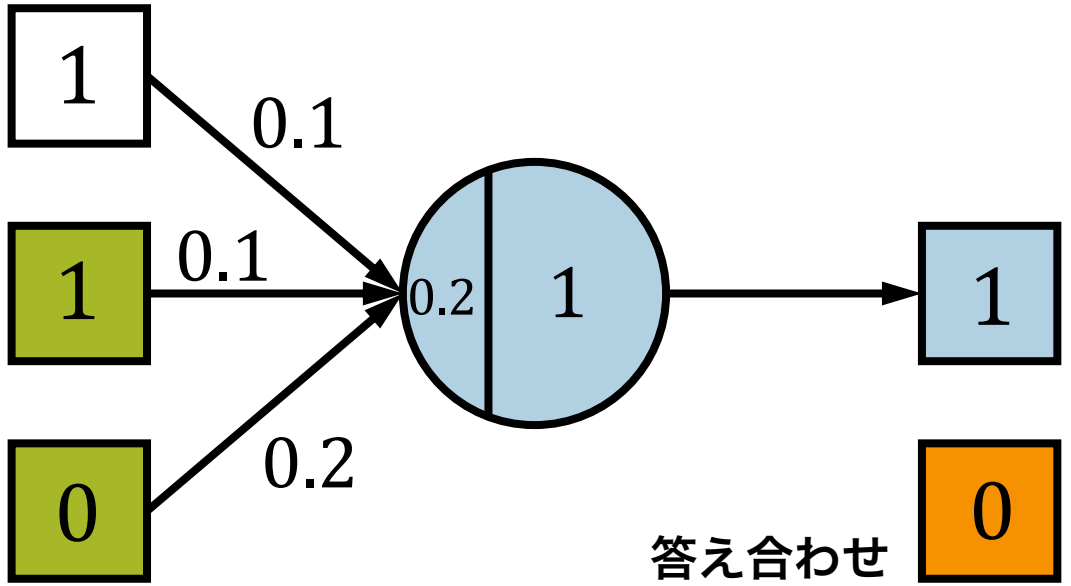
学習データ

気温・施肥量を使って開花予測を行うために、フィールド調査を行い次のようなデータを得た。

気温	施肥量	開花
0	0	0
1	0	0
0	1	0
1	1	1

※図はイメージである。





$$w_0^{(new)} = 0.1 + 1(0-1)0.1 = 0.0$$

$$w_1^{(new)} = 0.1 + 1(0-1)0.1 = 0.0$$

$$w_2^{(new)} = 0.2 + 0(0-1)0.1 = 0.2$$

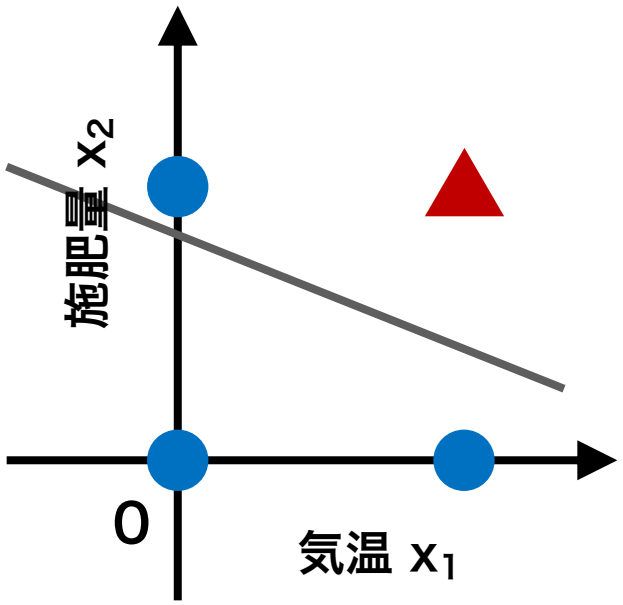
パーセプトロン学習則

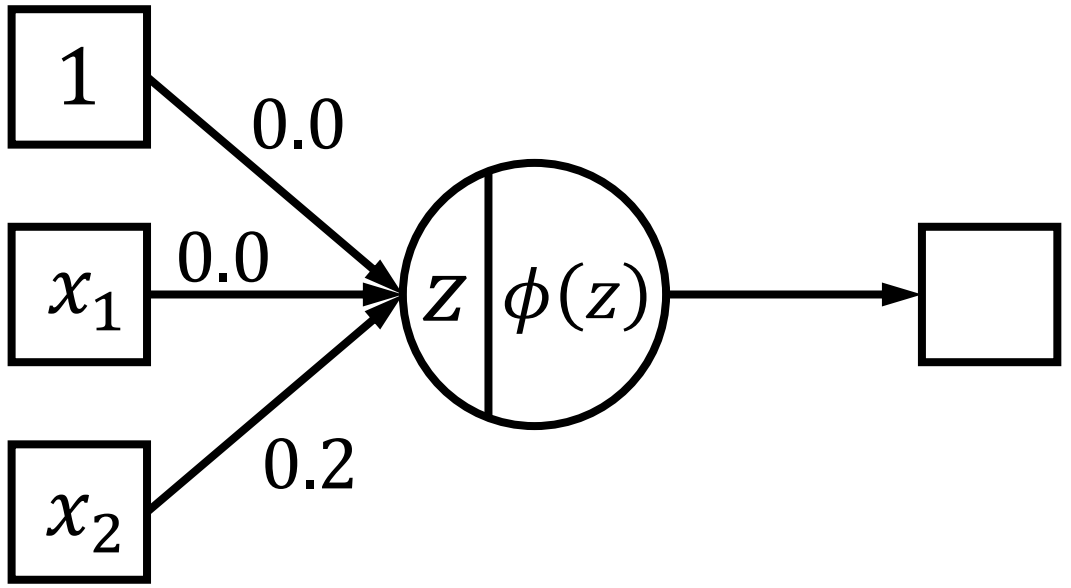
学習データ

気温・施肥量を使って開花予測を行うために、フィールド調査を行い次のようなデータを得た。

気温	施肥量	開花
0	0	0
1	0	0
0	1	0
1	1	1

※図はイメージである。





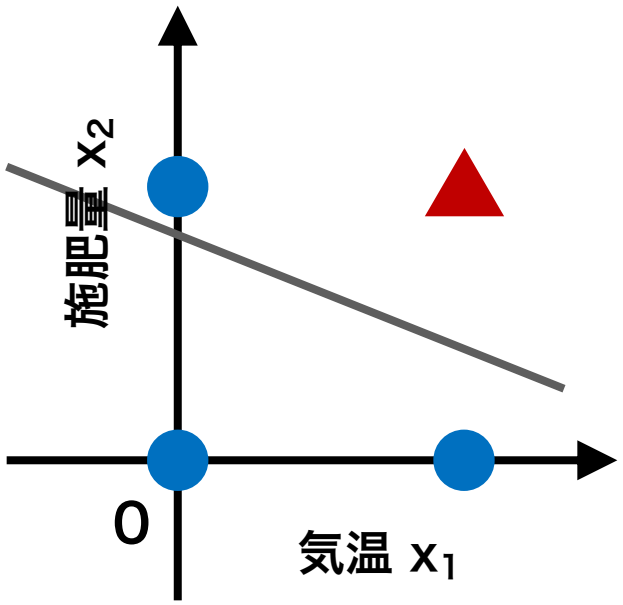
パーセプトロン学習則

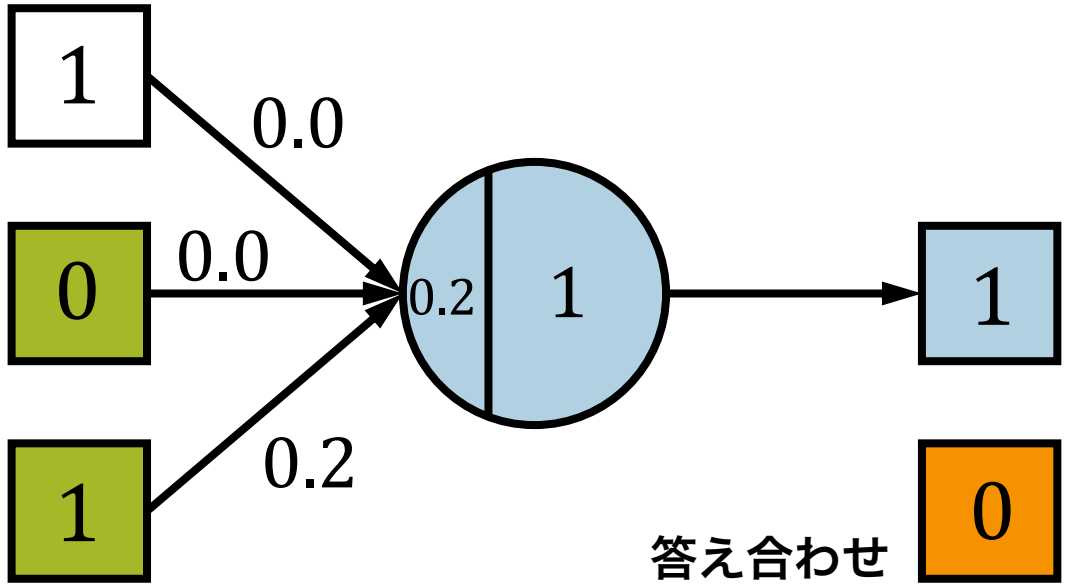
学習データ

気温・施肥量を使って開花予測を行うために、フィールド調査を行い次のようなデータを得た。

気温	施肥量	開花
0	0	0
1	0	0
0	1	0
1	1	1

※図はイメージである。





$$w_0^{(new)} = 0.0 + 1(0-1)0.1 = -0.1$$

$$w_1^{(new)} = 0.0 + 0(0-1)0.1 = 0.0$$

$$w_2^{(new)} = 0.2 + 1(0-1)0.1 = 0.1$$

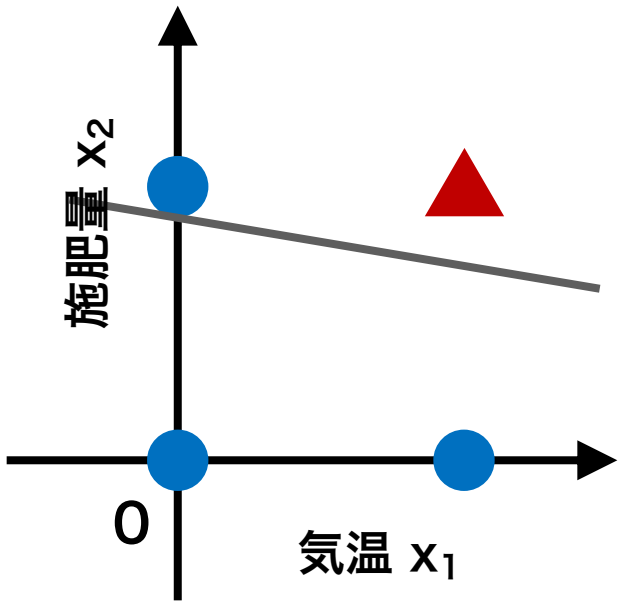
パーセプトロン学習則

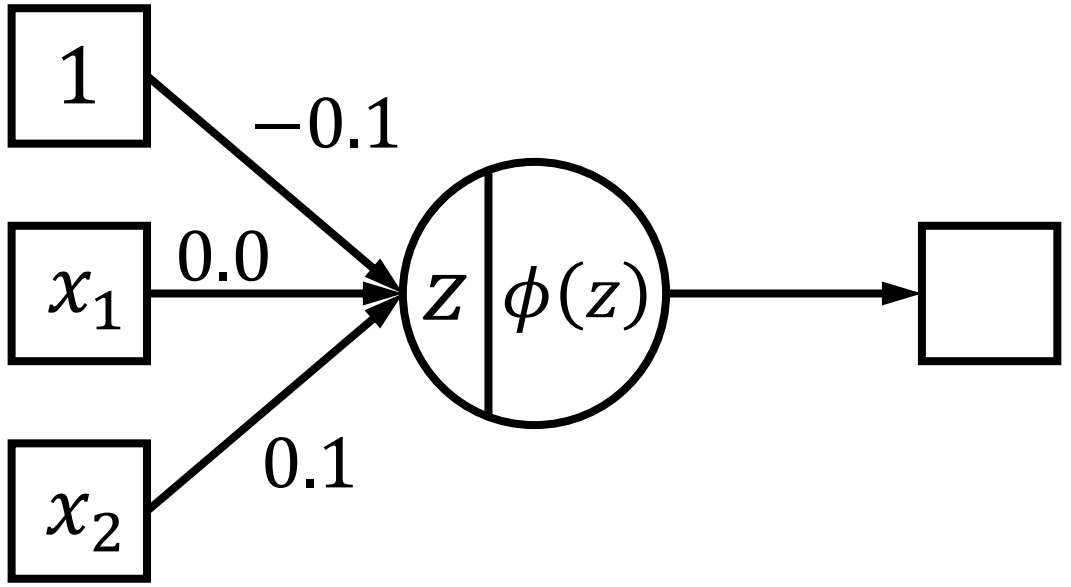
学習データ

気温・施肥量を使って開花予測を行うために、フィールド調査を行い次のようなデータを得た。

気温	施肥量	開花
0	0	0
1	0	0
0	1	0
1	1	1

※図はイメージである。





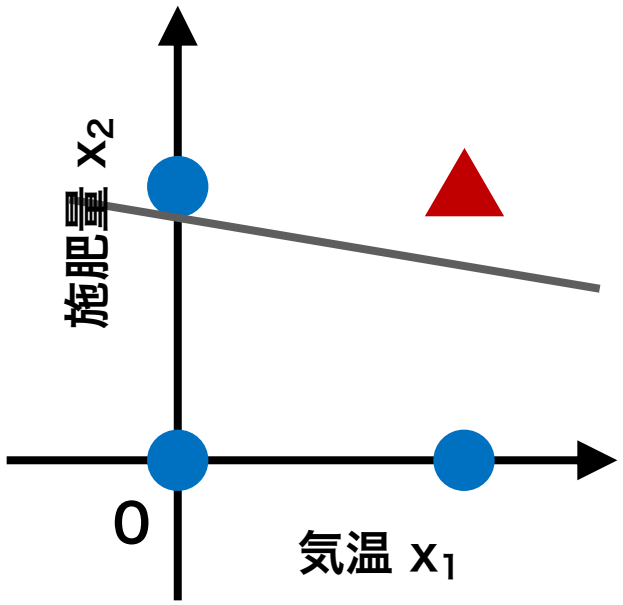
パーセプトロン学習則

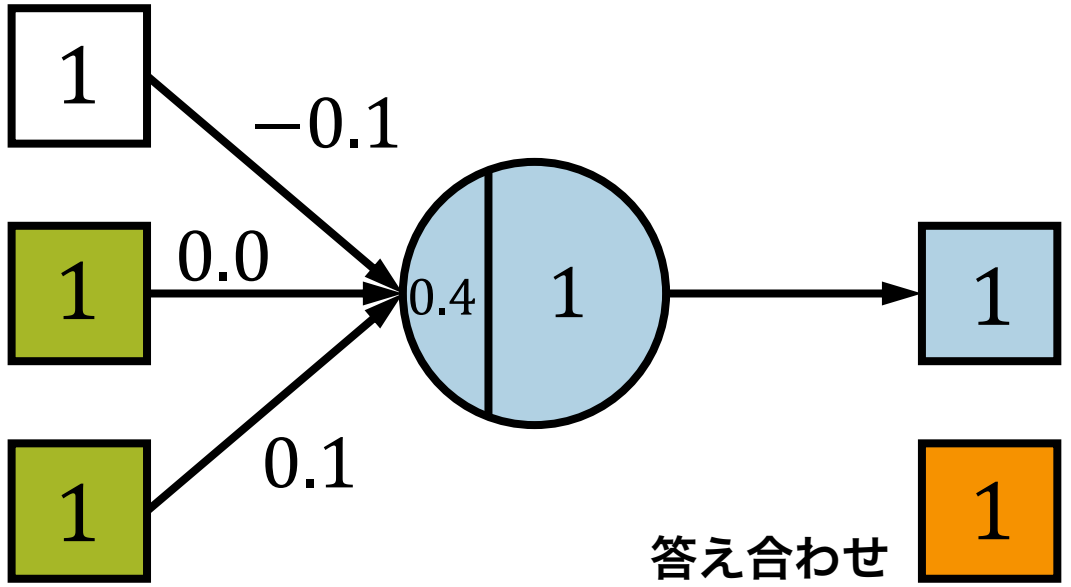
学習データ

気温・施肥量を使って開花予測を行うために、フィールド調査を行い次のようなデータを得た。

気温	施肥量	開花
0	0	0
1	0	0
0	1	0
1	1	1

※図はイメージである。





$$w_0^{(new)} = -0.1 + 1(1-1)0.1 = -0.1$$

$$w_1^{(new)} = 0.0 + 0(1-1)0.1 = 0.0$$

$$w_2^{(new)} = 0.1 + 1(1-1)0.1 = 0.1$$

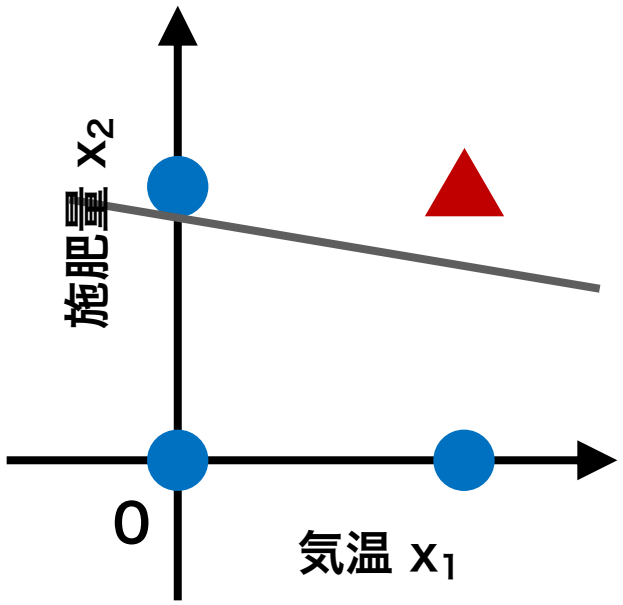
パーセプトロン学習則

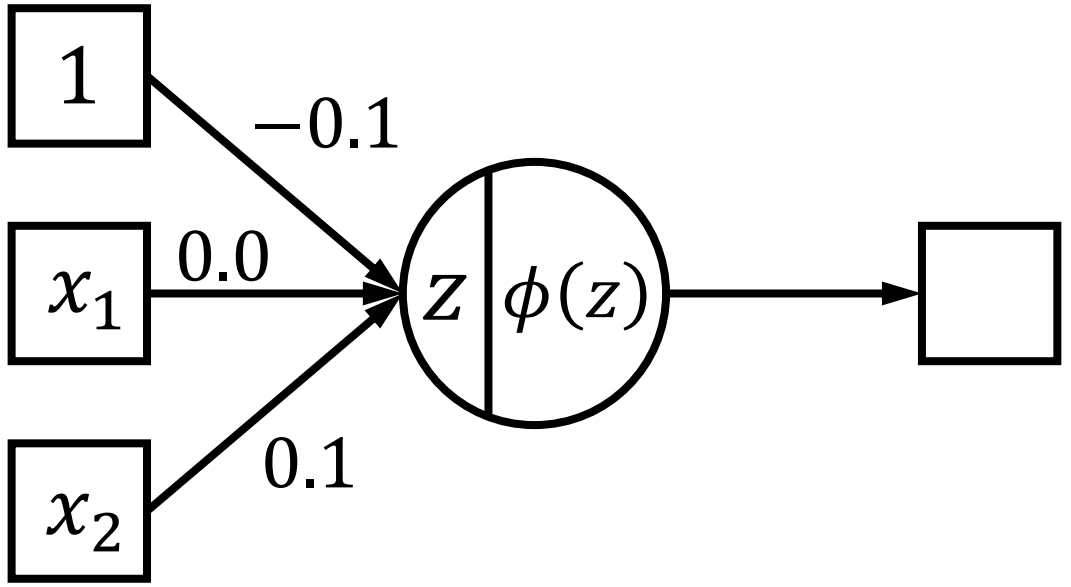
学習データ

気温・施肥量を使って開花予測を行うために、フィールド調査を行い次のようなデータを得た。

気温	施肥量	開花
0	0	0
1	0	0
0	1	0
1	1	1

※図はイメージである。





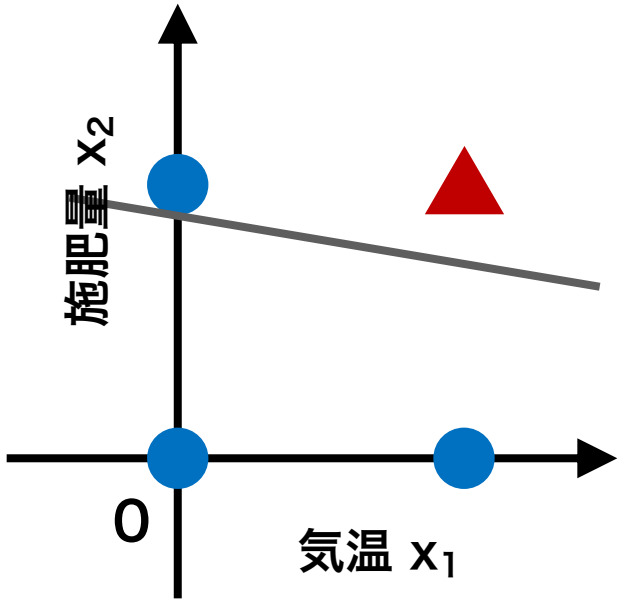
パーセプトロン学習則

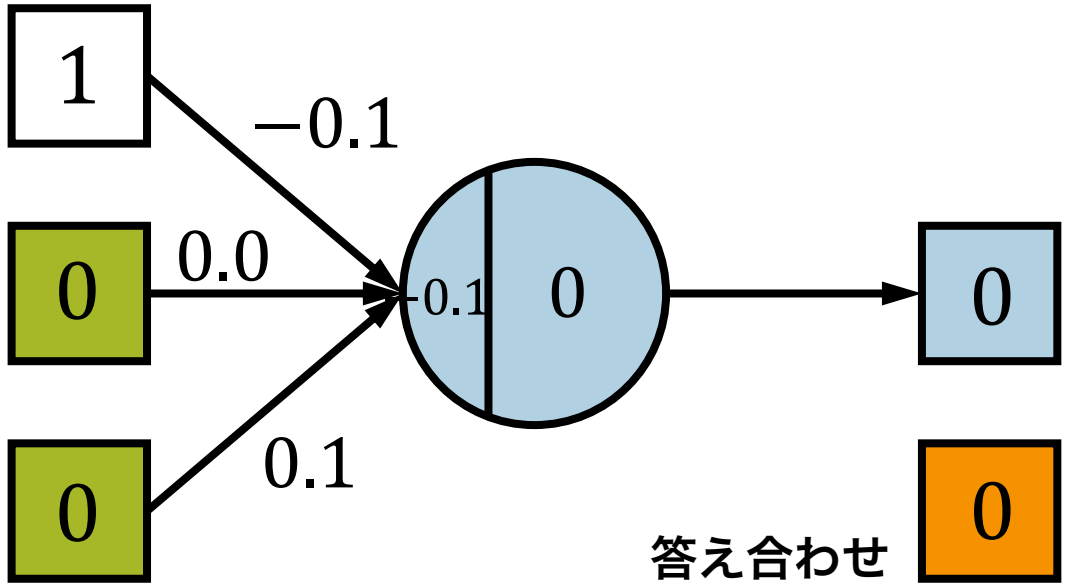
学習データ

気温・施肥量を使って開花予測を行うために、フィールド調査を行い次のようなデータを得た。

気温	施肥量	開花
0	0	0
1	0	0
0	1	0
1	1	1

※図はイメージである。





$$w_0^{(new)} = -0.1 + 1(0-0)0.1 = -0.1$$

$$w_1^{(new)} = 0.0 + 0(0-0)0.1 = 0.0$$

$$w_2^{(new)} = 0.1 + 0(0-0)0.1 = 0.1$$

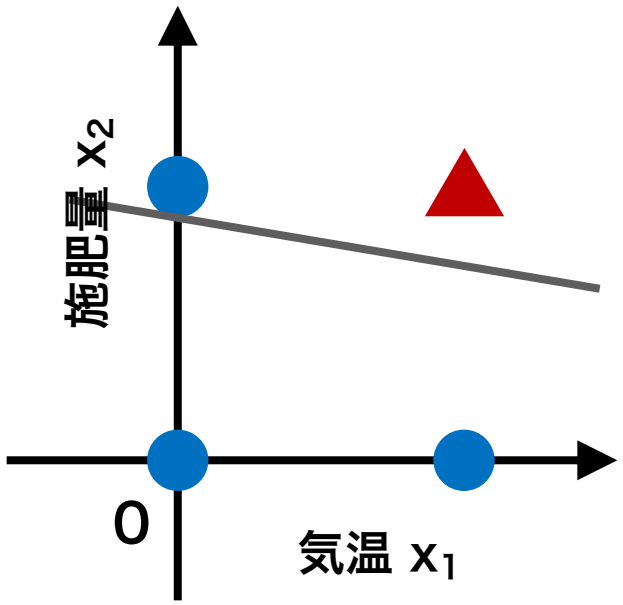
パーセプトロン学習則

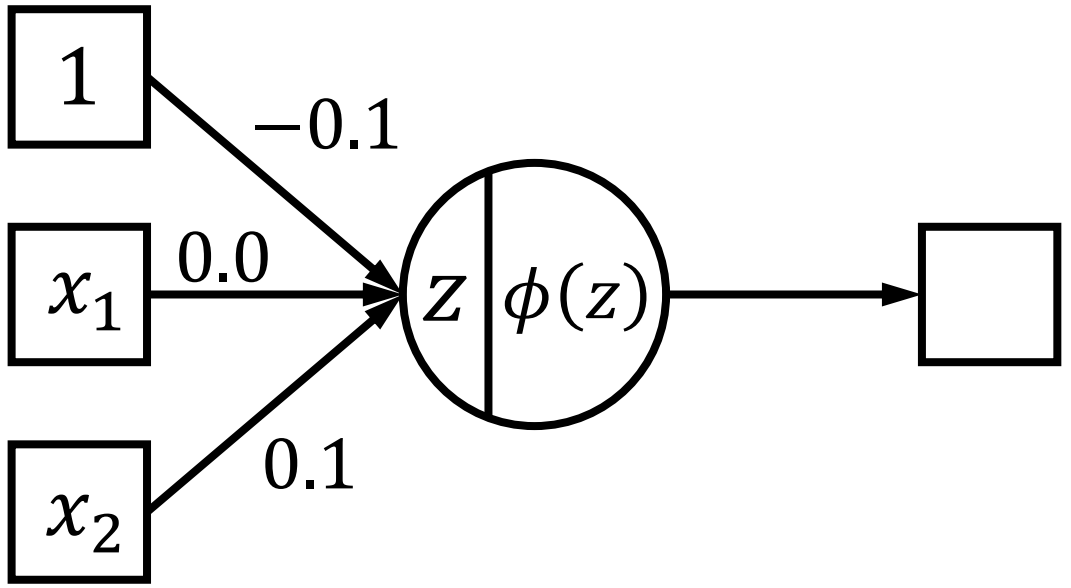
学習データ

気温・施肥量を使って開花予測を行うために、フィールド調査を行い次のようなデータを得た。

気温	施肥量	開花
0	0	0
1	0	0
0	1	0
1	1	1

※図はイメージである。





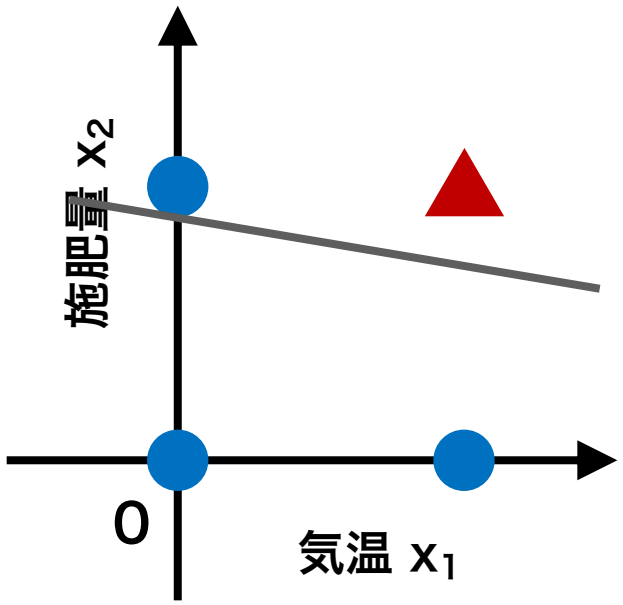
パーセプトロン学習則

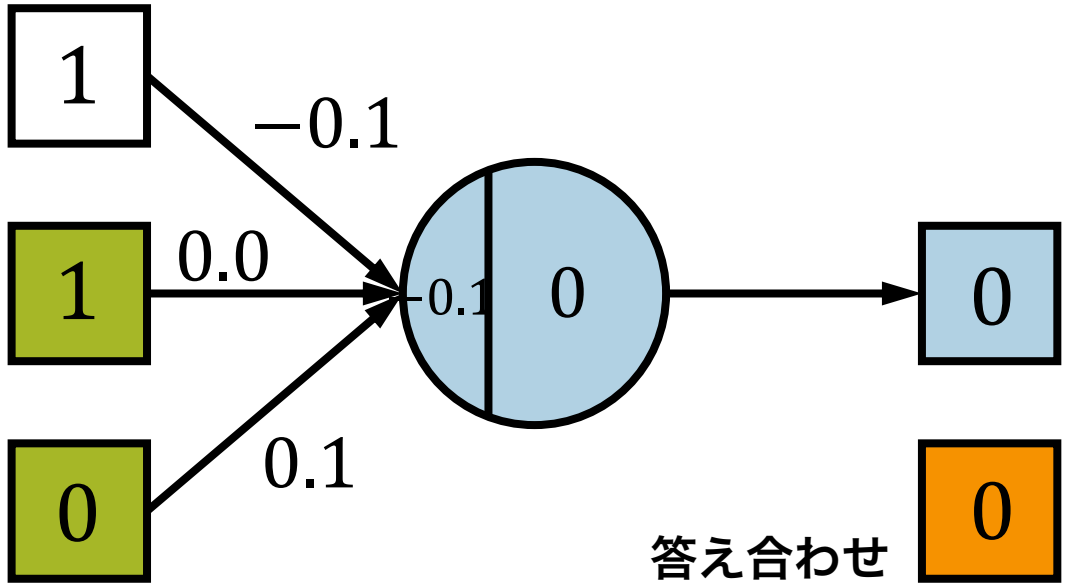
学習データ

気温・施肥量を使って開花予測を行うために、フィールド調査を行い次のようなデータを得た。

気温	施肥量	開花
0	0	0
1	0	0
0	1	0
1	1	1

※図はイメージである。





$$w_0^{(new)} = -0.1 + 1(0-0)0.1 = -0.1$$

$$w_1^{(new)} = 0.0 + 0(0-0)0.1 = 0.0$$

$$w_2^{(new)} = 0.1 + 0(0-0)0.1 = 0.1$$

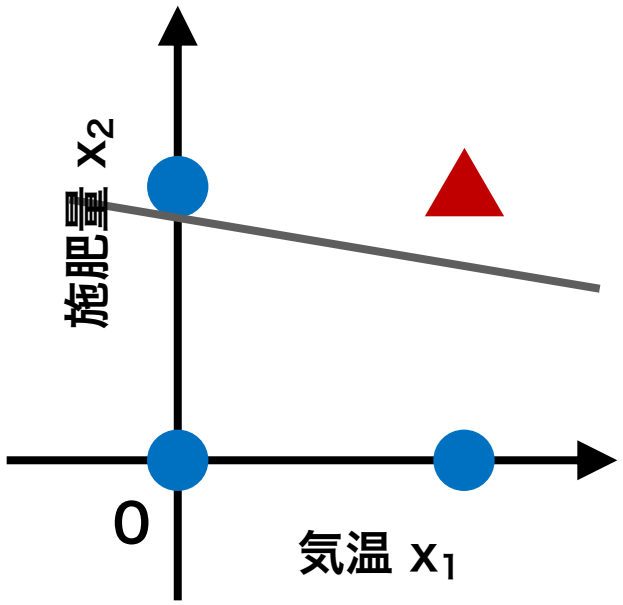
パーセプトロン学習則

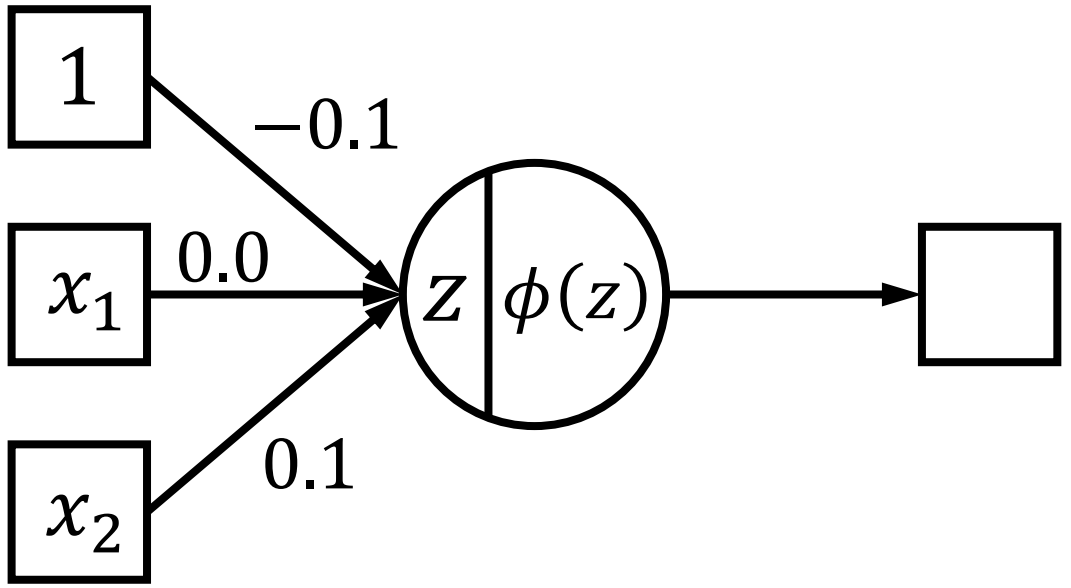
学習データ

気温・施肥量を使って開花予測を行うために、フィールド調査を行い次のようなデータを得た。

気温	施肥量	開花
0	0	0
1	0	0
0	1	0
1	1	1

※図はイメージである。





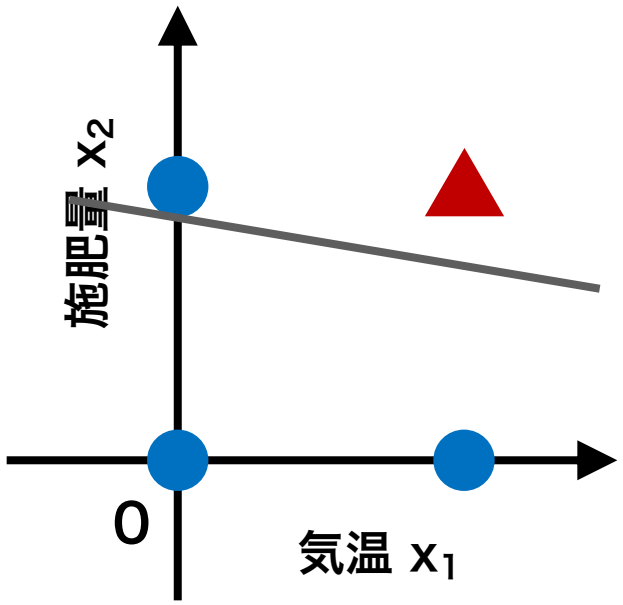
パーセプトロン学習則

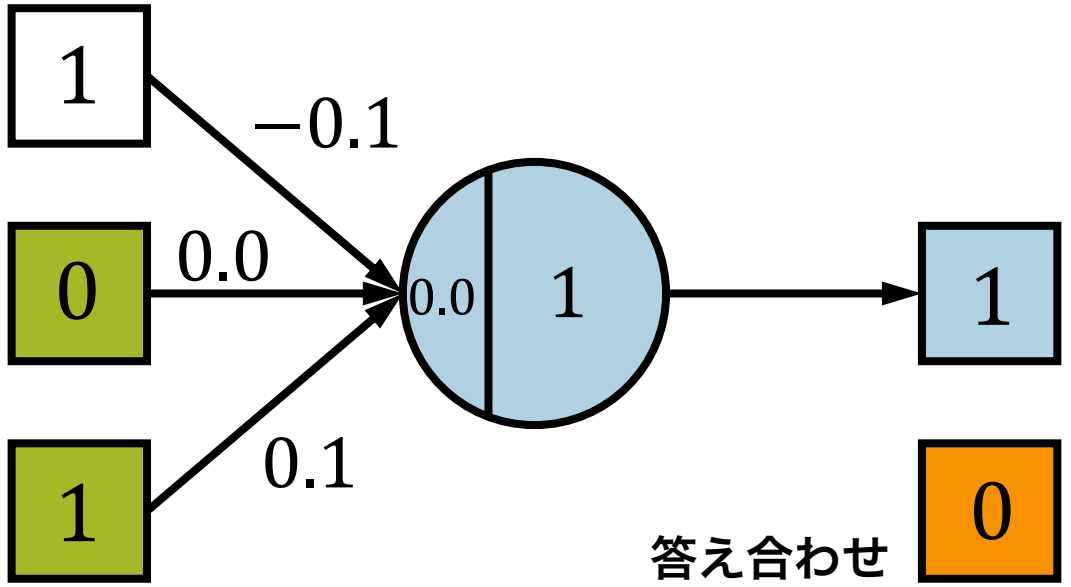
学習データ

気温・施肥量を使って開花予測を行うために、フィールド調査を行い次のようなデータを得た。

気温	施肥量	開花
0	0	0
1	0	0
0	1	0
1	1	1

※図はイメージである。





$$w_0^{(new)} = -0.1 + 1(0-1)0.1 = -0.2$$

$$w_1^{(new)} = 0.0 + 0(0-1)0.1 = 0.0$$

$$w_2^{(new)} = 0.1 + 1(0-1)0.1 = 0.0$$

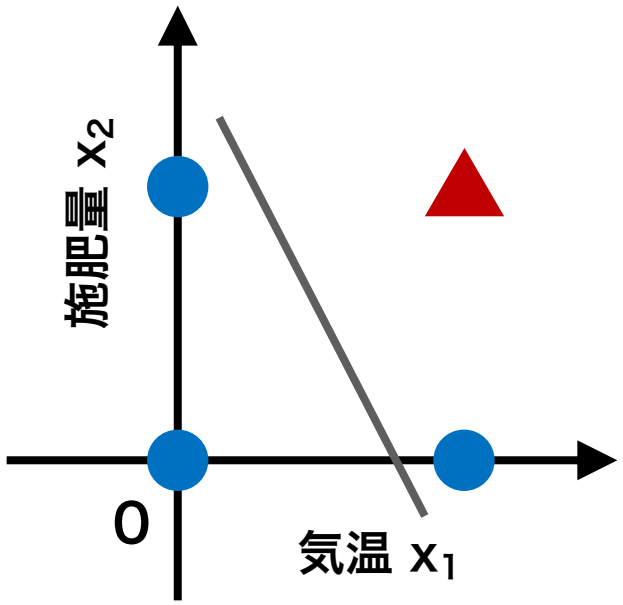
パーセプトロン学習則

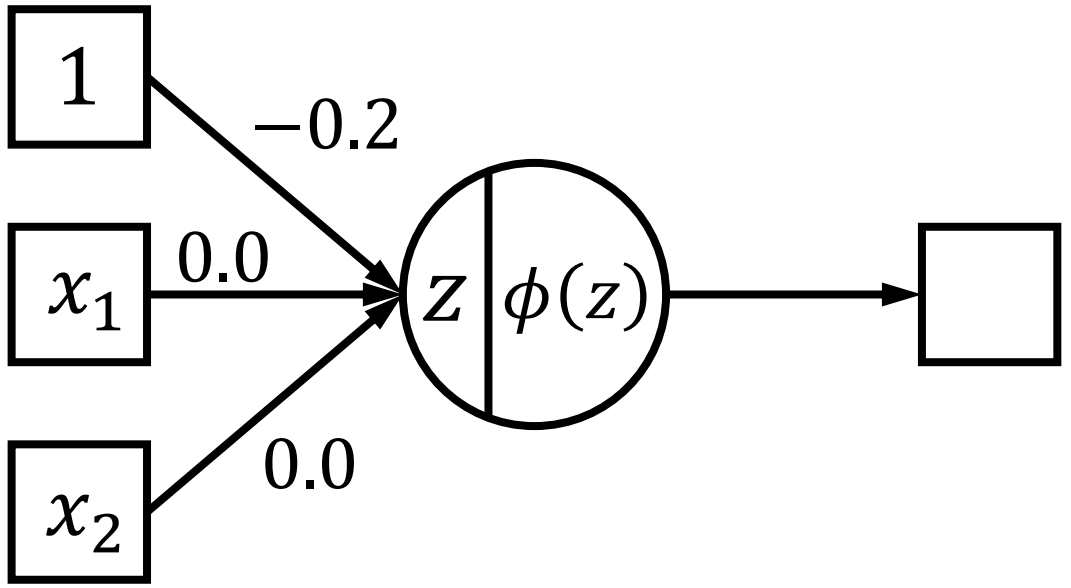
学習データ

気温・施肥量を使って開花予測を行うために、フィールド調査を行い次のようなデータを得た。

気温	施肥量	開花
0	0	0
1	0	0
0	1	0
1	1	1

※図はイメージである。





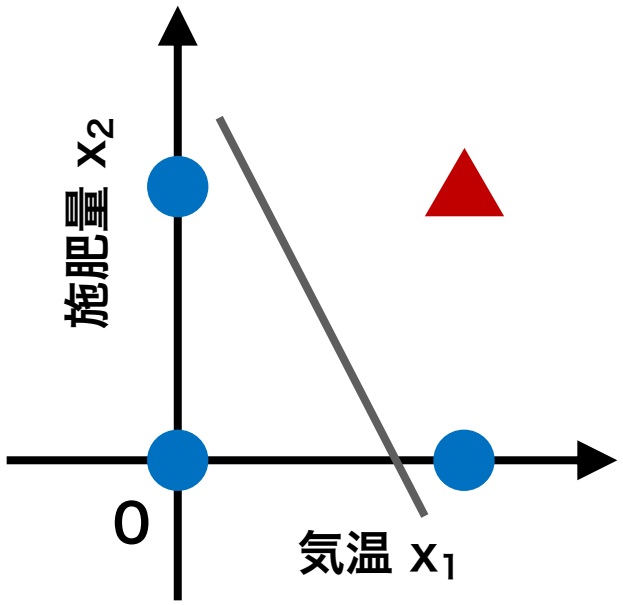
パーセプトロン学習則

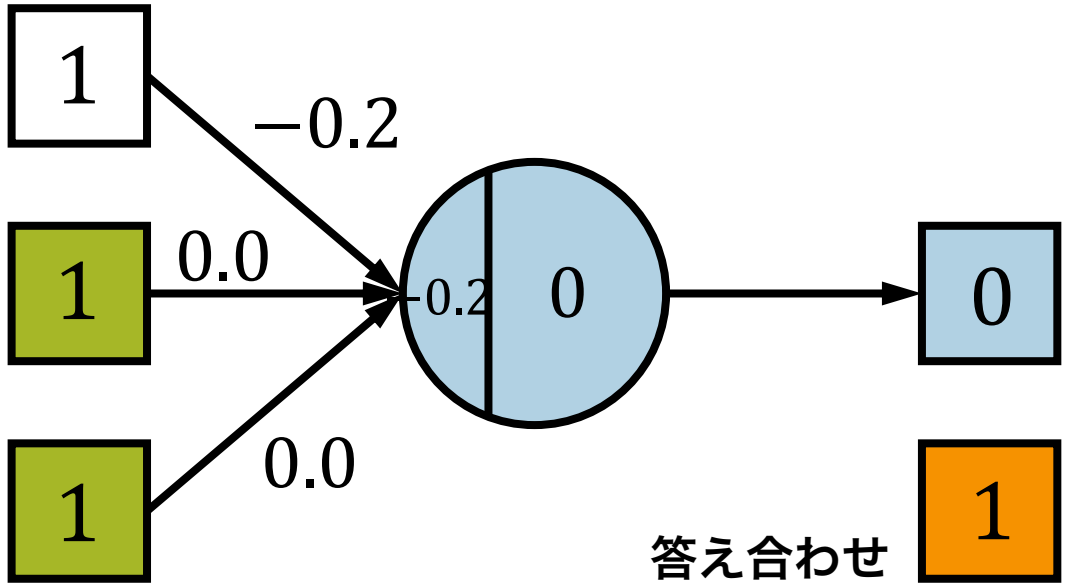
学習データ

気温・施肥量を使って開花予測を行うために、フィールド調査を行い次のようなデータを得た。

気温	施肥量	開花
0	0	0
1	0	0
0	1	0
1	1	1

※図はイメージである。





$$w_0^{(new)} = -0.2 + 1(1-0)0.1 = -0.1$$

$$w_1^{(new)} = 0.0 + 1(1-0)0.1 = 0.1$$

$$w_2^{(new)} = 0.1 + 1(1-0)0.1 = 0.2$$

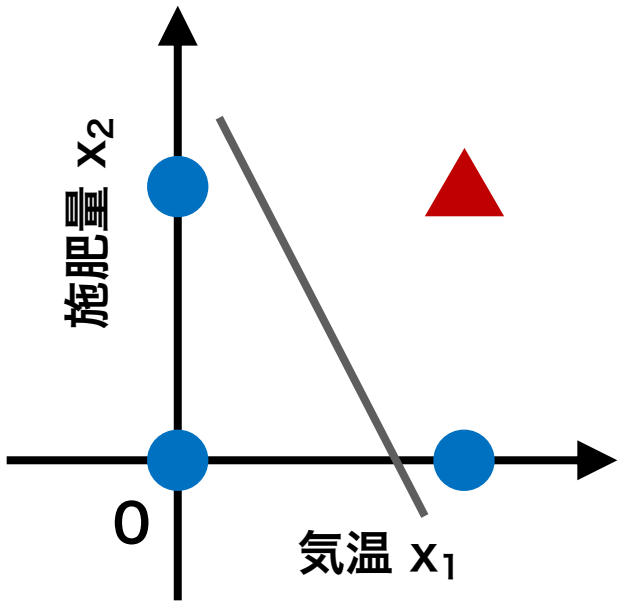
パーセプトロン学習則

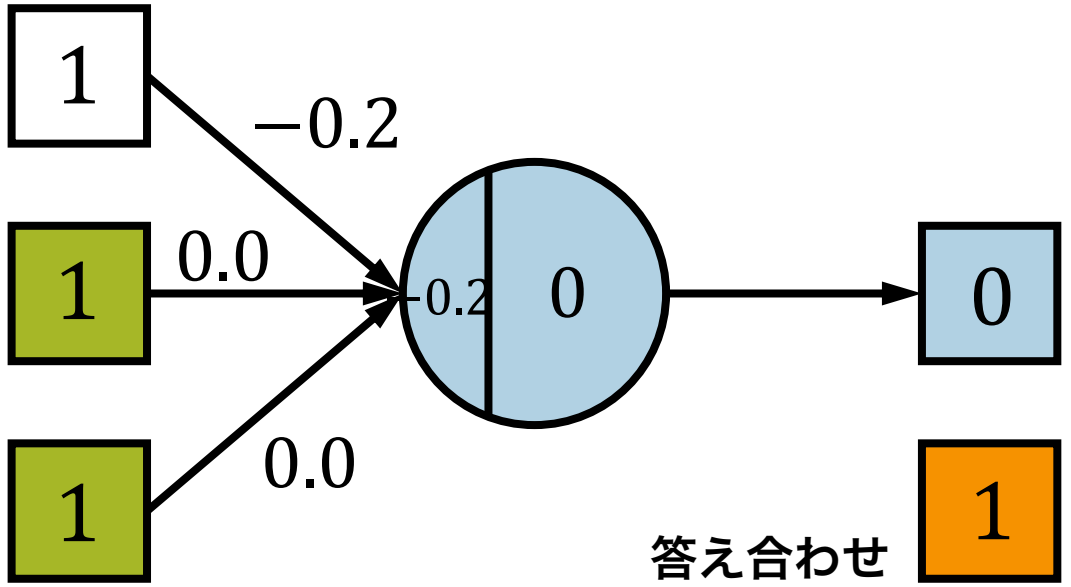
学習データ

気温・施肥量を使って開花予測を行うために、フィールド調査を行い次のようなデータを得た。

気温	施肥量	開花
0	0	0
1	0	0
0	1	0
1	1	1

※図はイメージである。





$$w_0^{(new)} = -0.2 + 1(1-0)0.1 = -0.1$$

$$w_1^{(new)} = 0.0 + 1(1-0)0.1 = 0.1$$

$$w_2^{(new)} = 0.1 + 1(1-0)0.1 = 0.2$$

パーセプトロン学習則

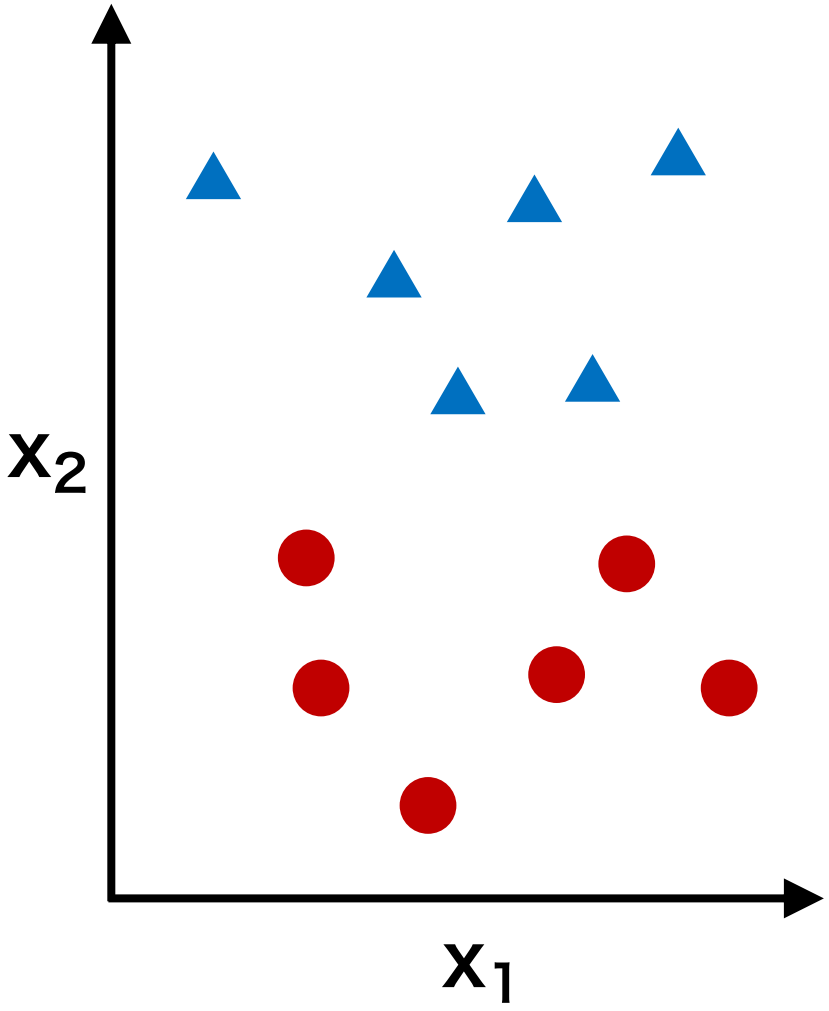
$$w_j^{new} = w_j^{old} + x_j \underbrace{(y - \hat{y})}_{\text{損失}} \underbrace{\eta}_{\text{学習率}}$$

ニューラルネットワーク

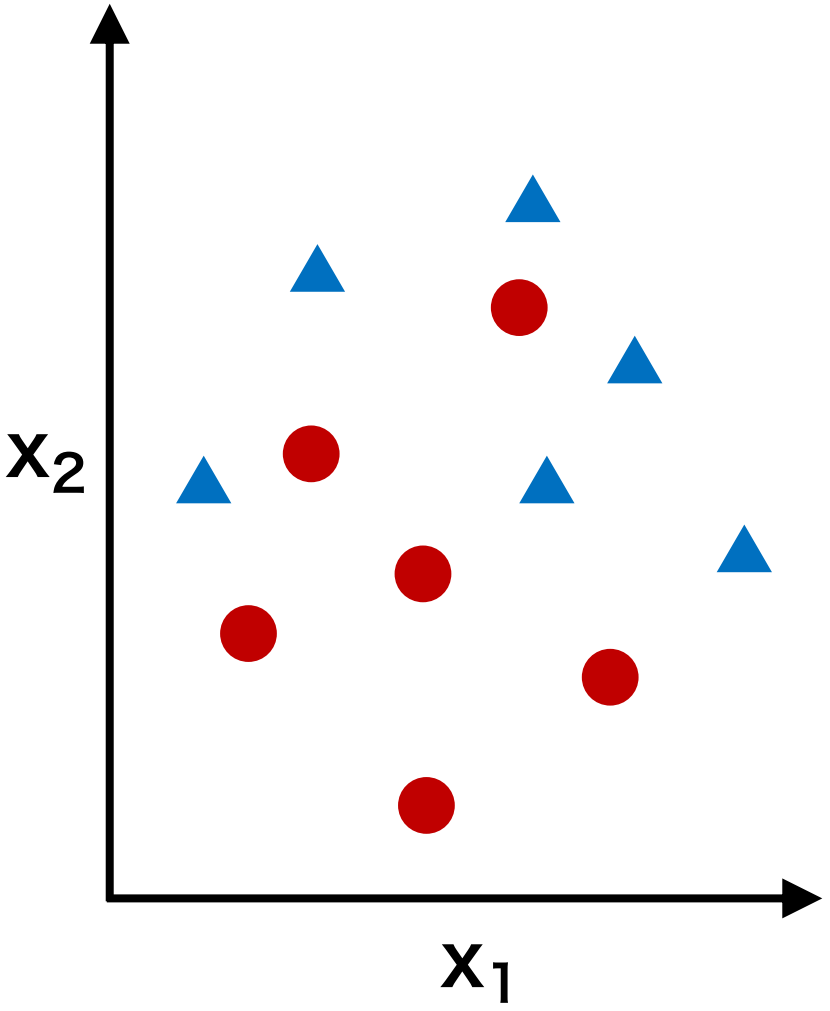
- パーセプトロン
- ニューラルネットワーク
- CNN
- RNN

線形分離

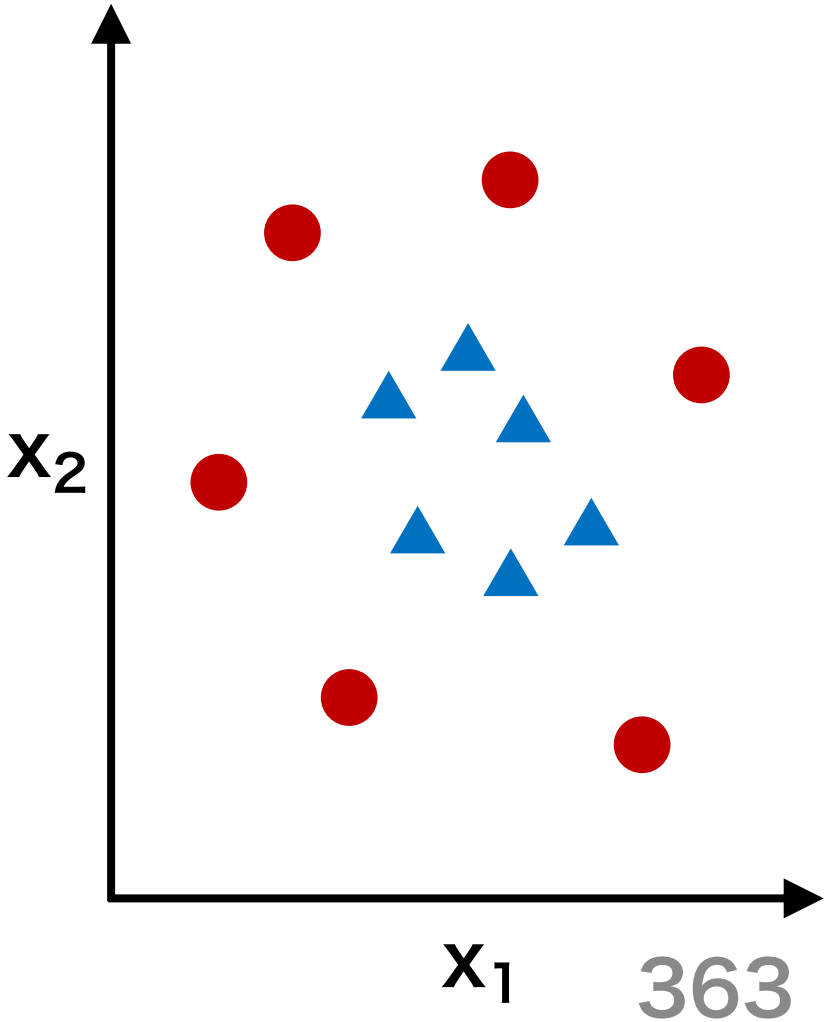
線形分離可能



線形分離不可

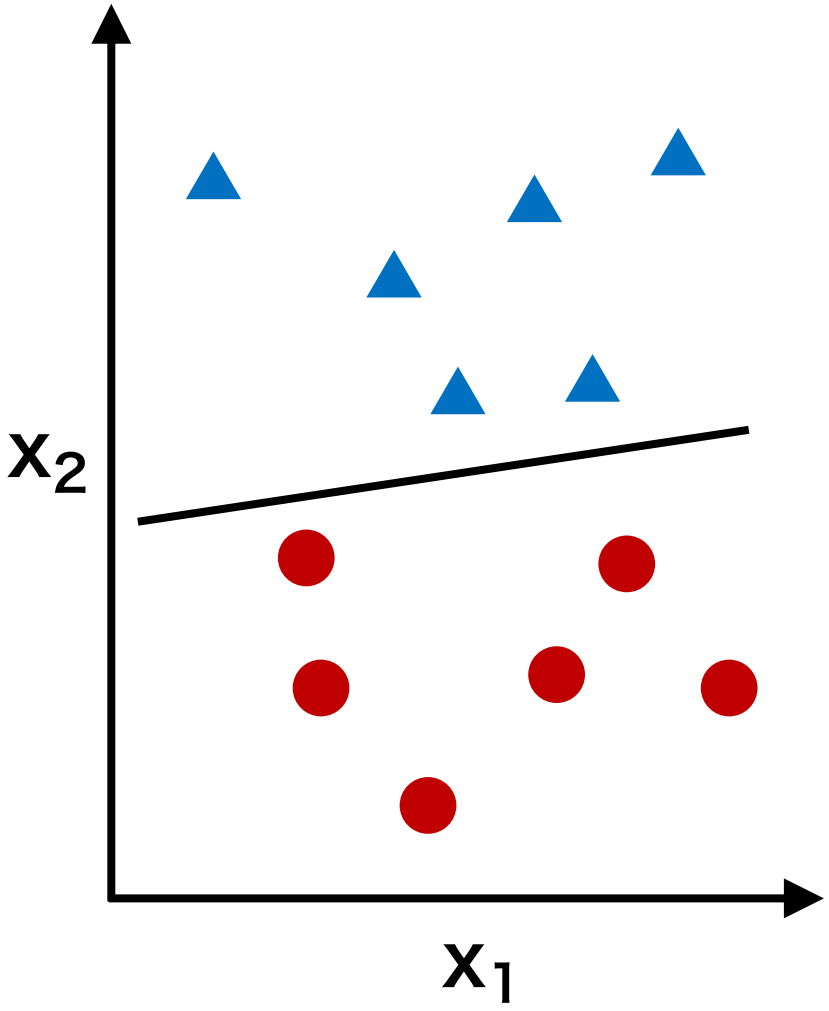


線形分離不可

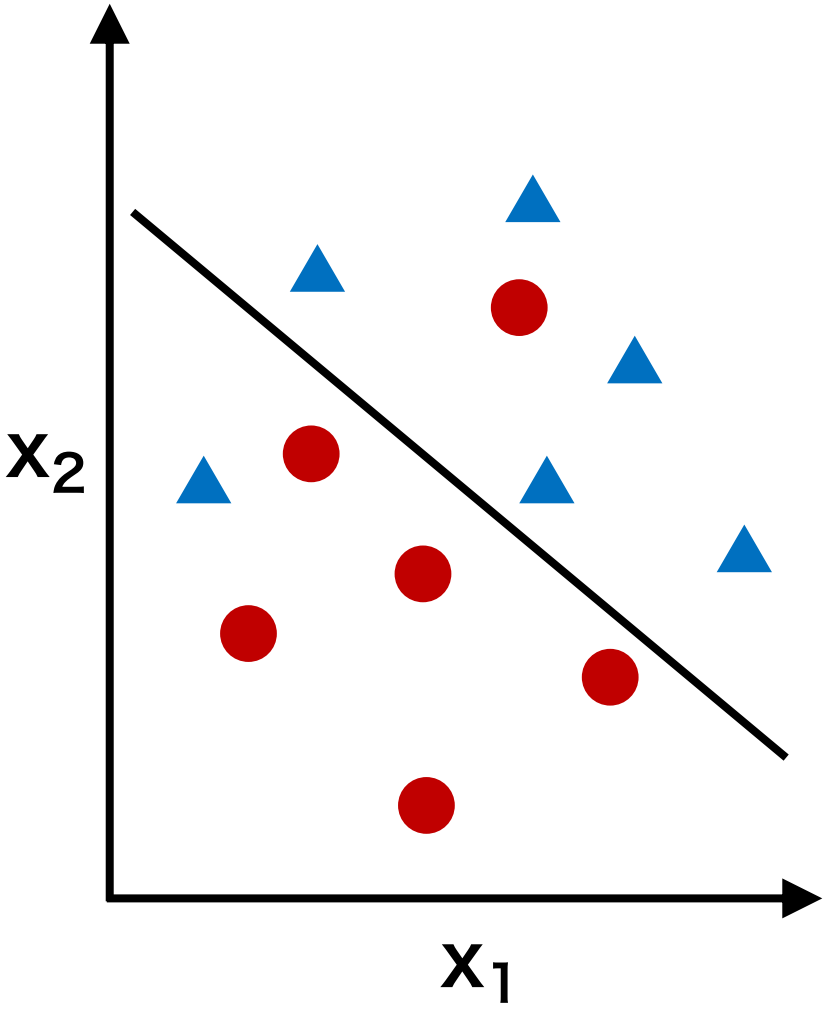


線形分離

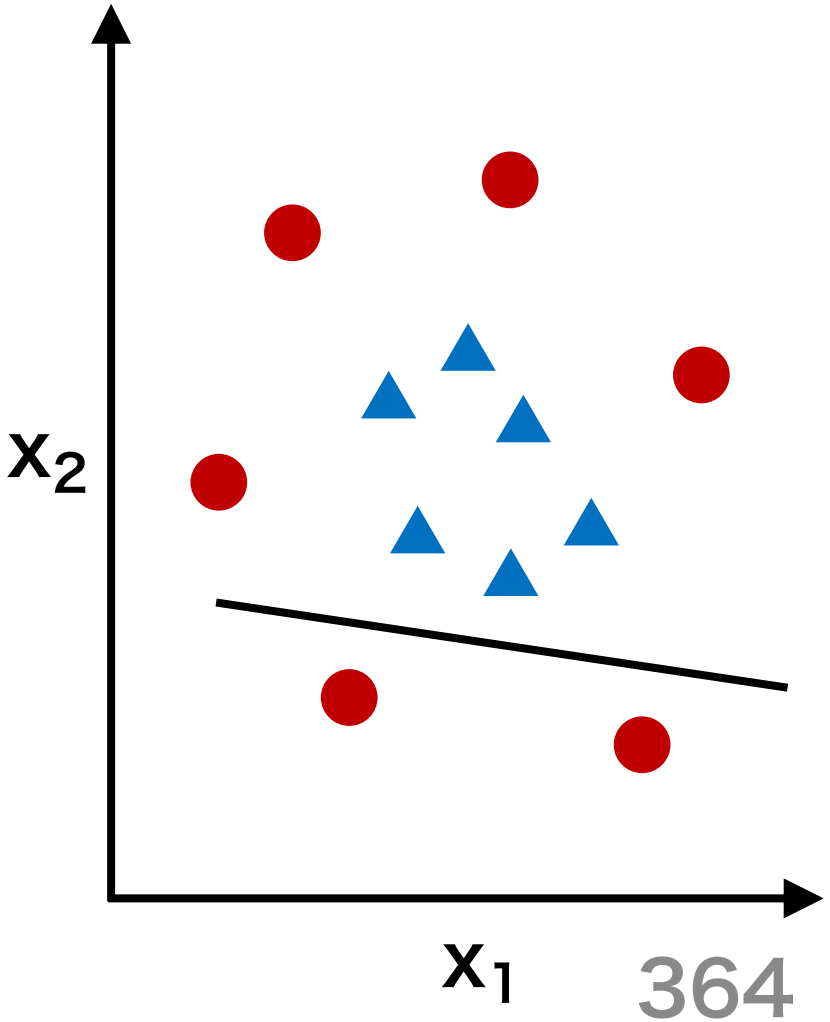
線形分離可能



線形分離不可

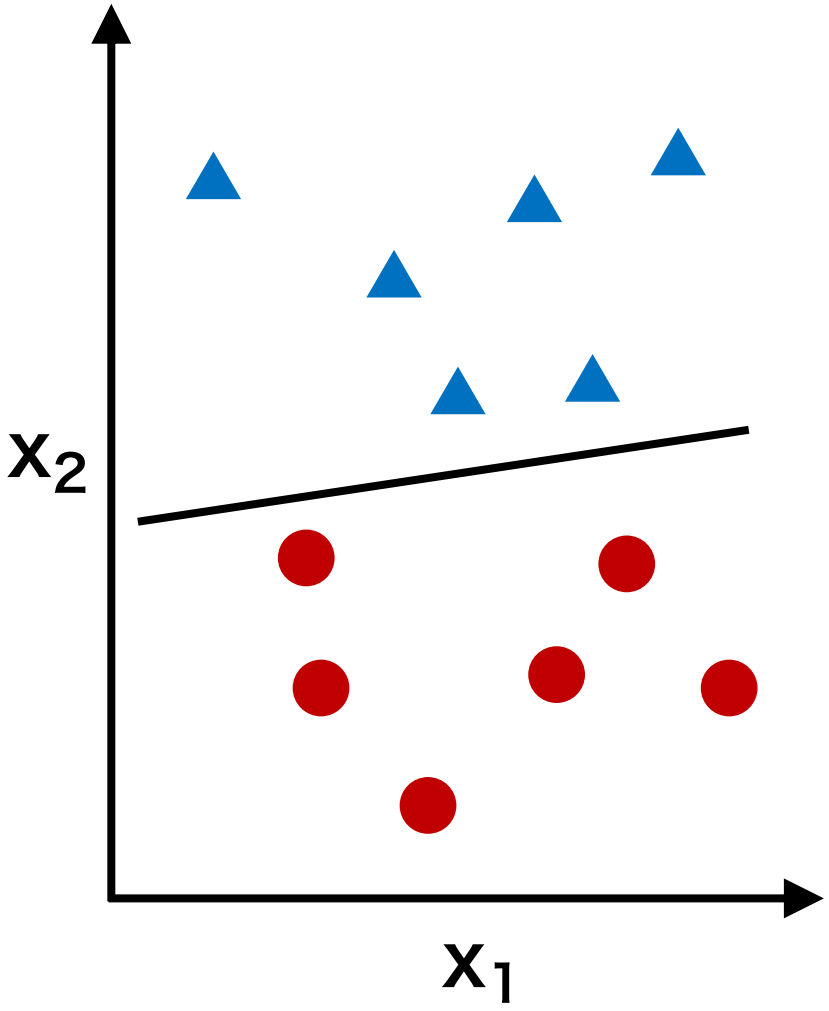


線形分離不可

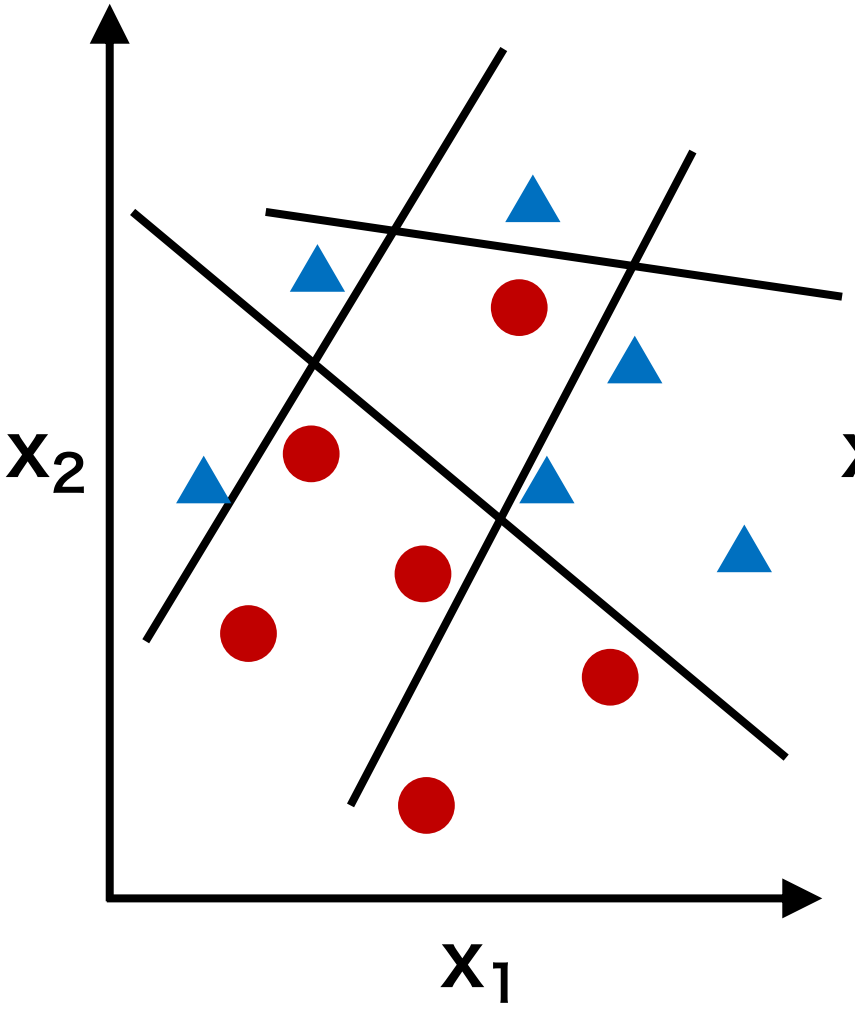


線形分離

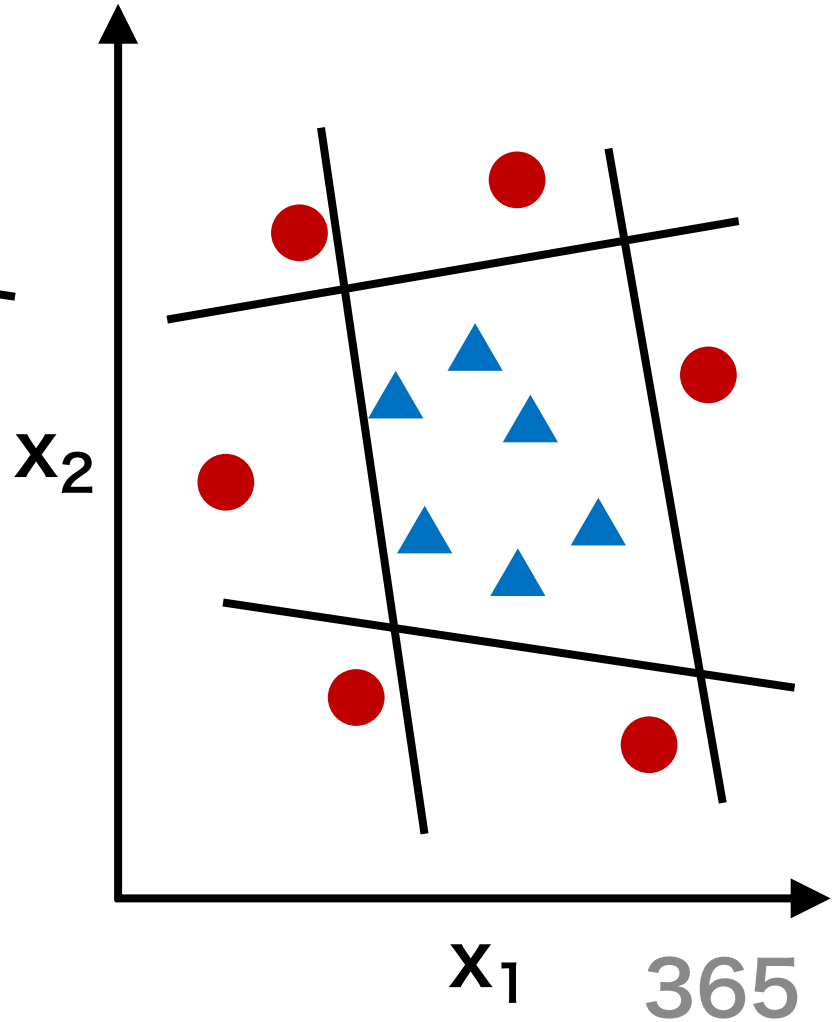
線形分離可能



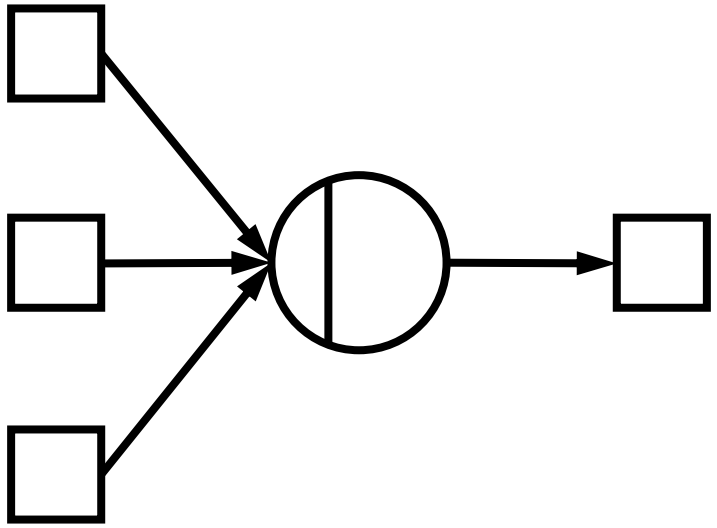
線形分離不可

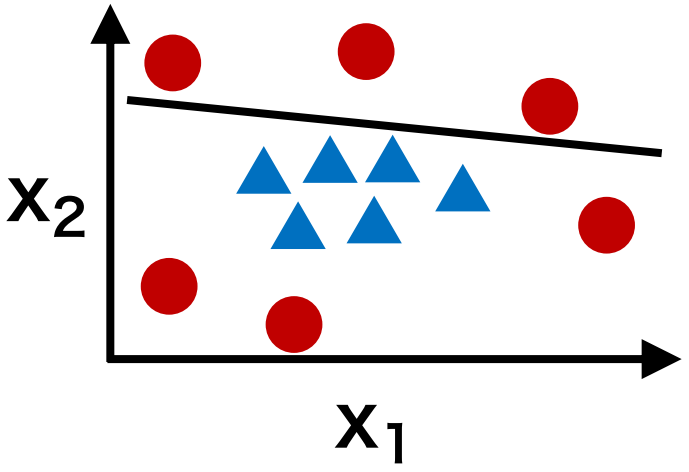


線形分離不可

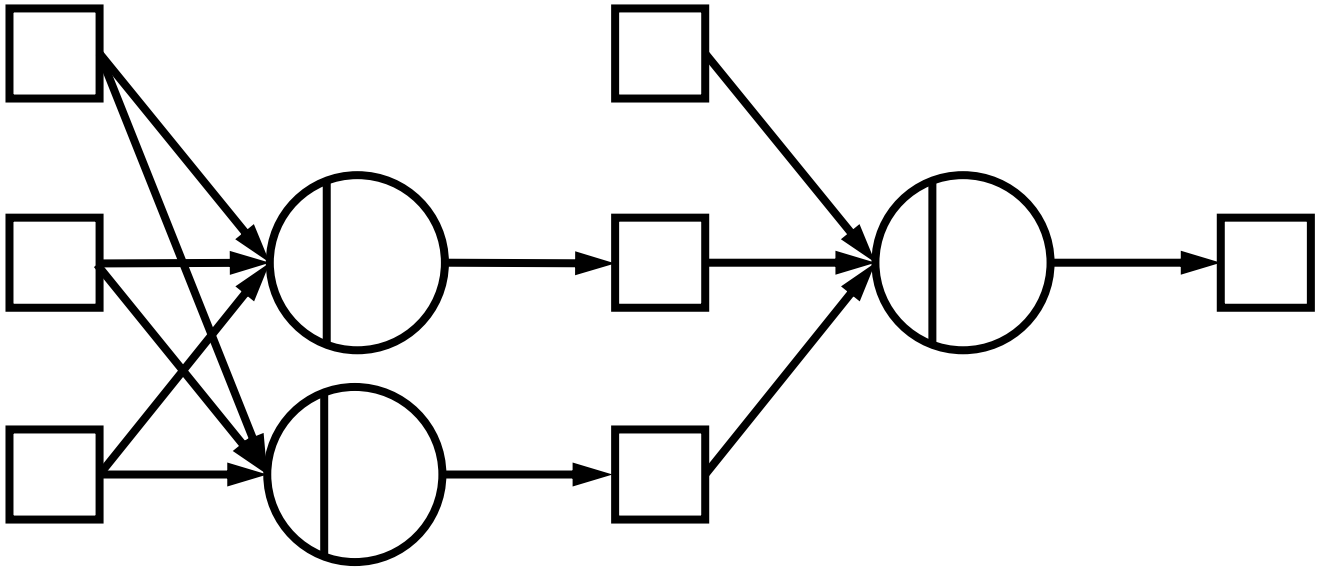


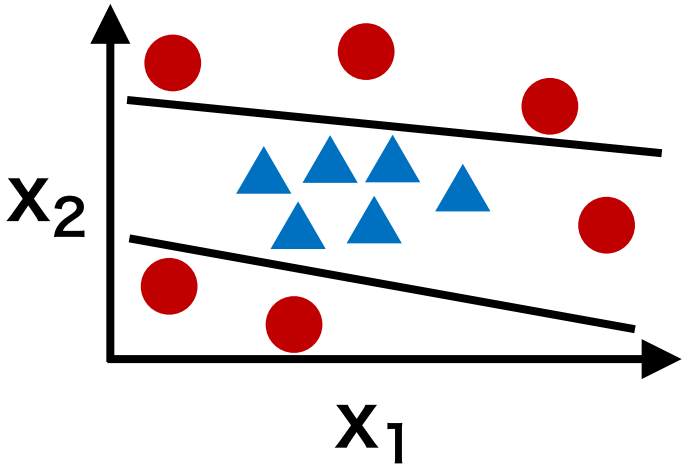
ニューラルネットワーク



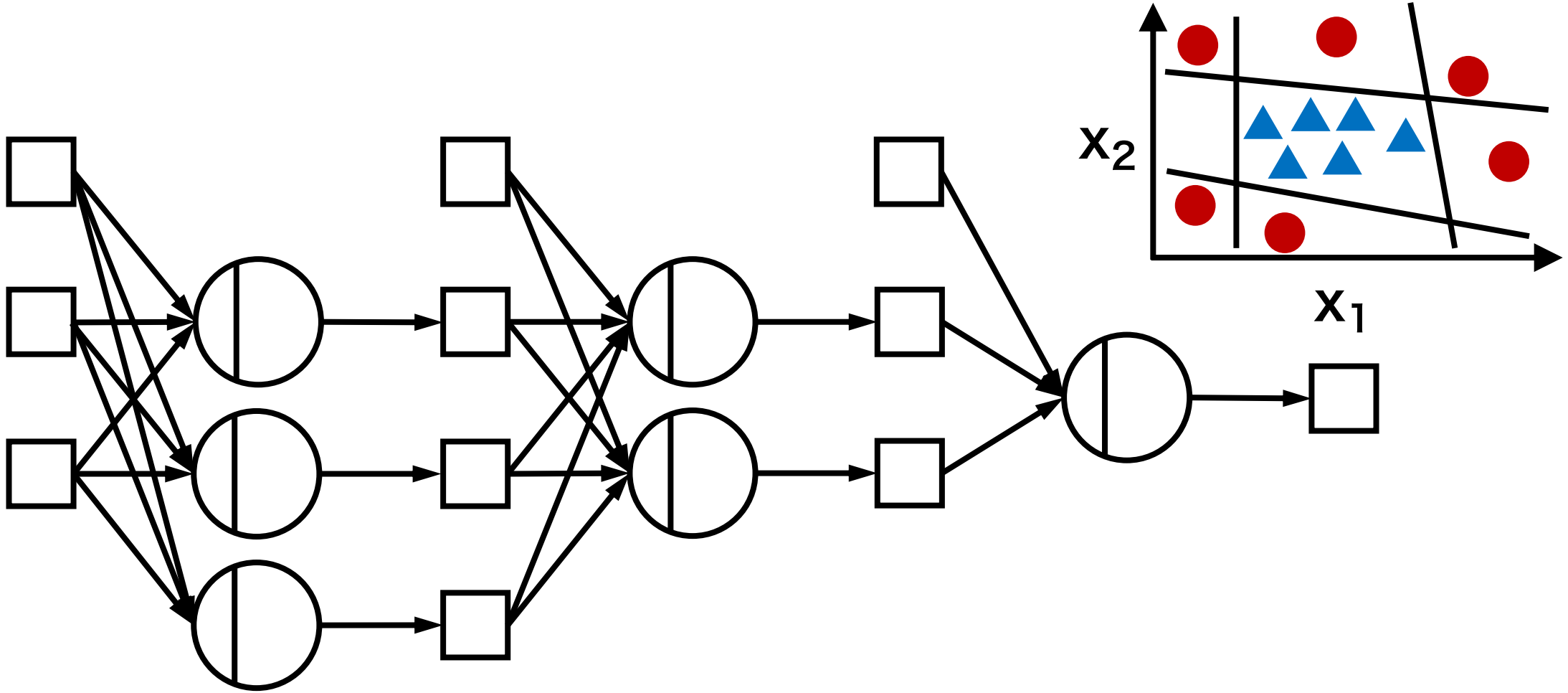


ニューラルネットワーク

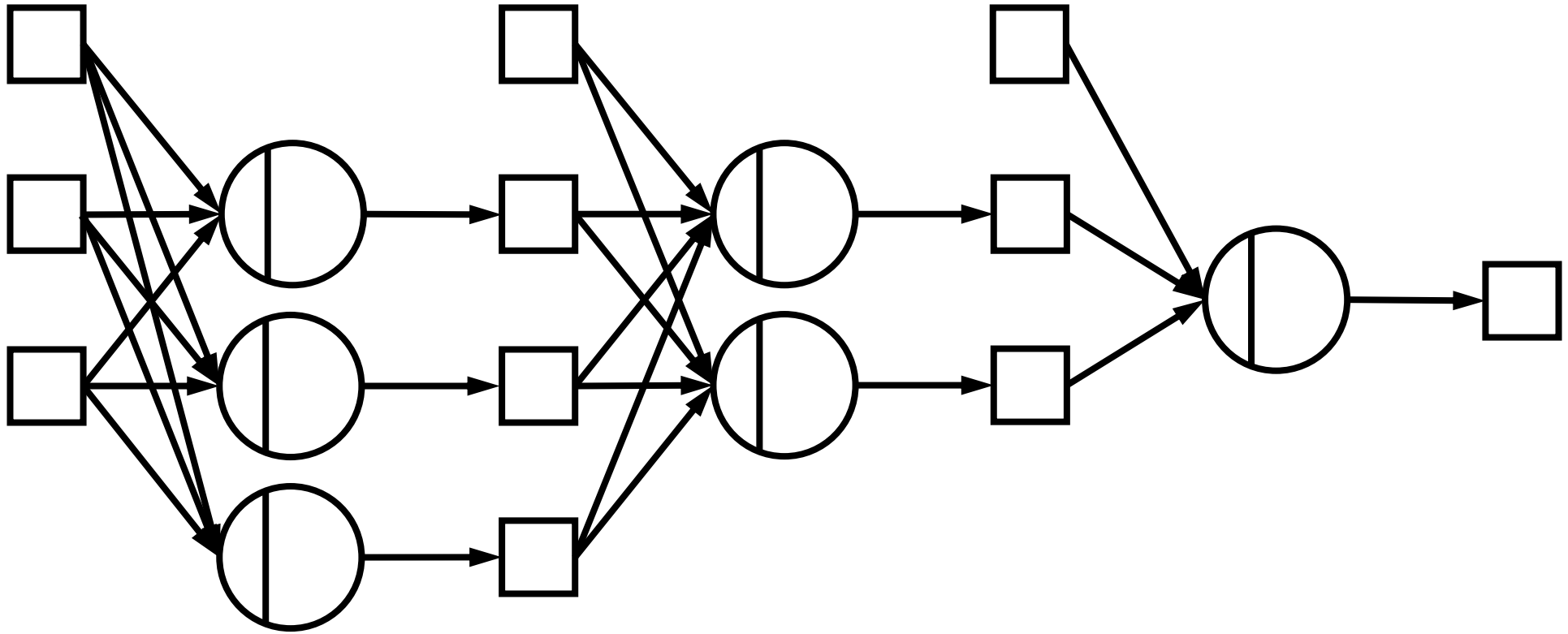




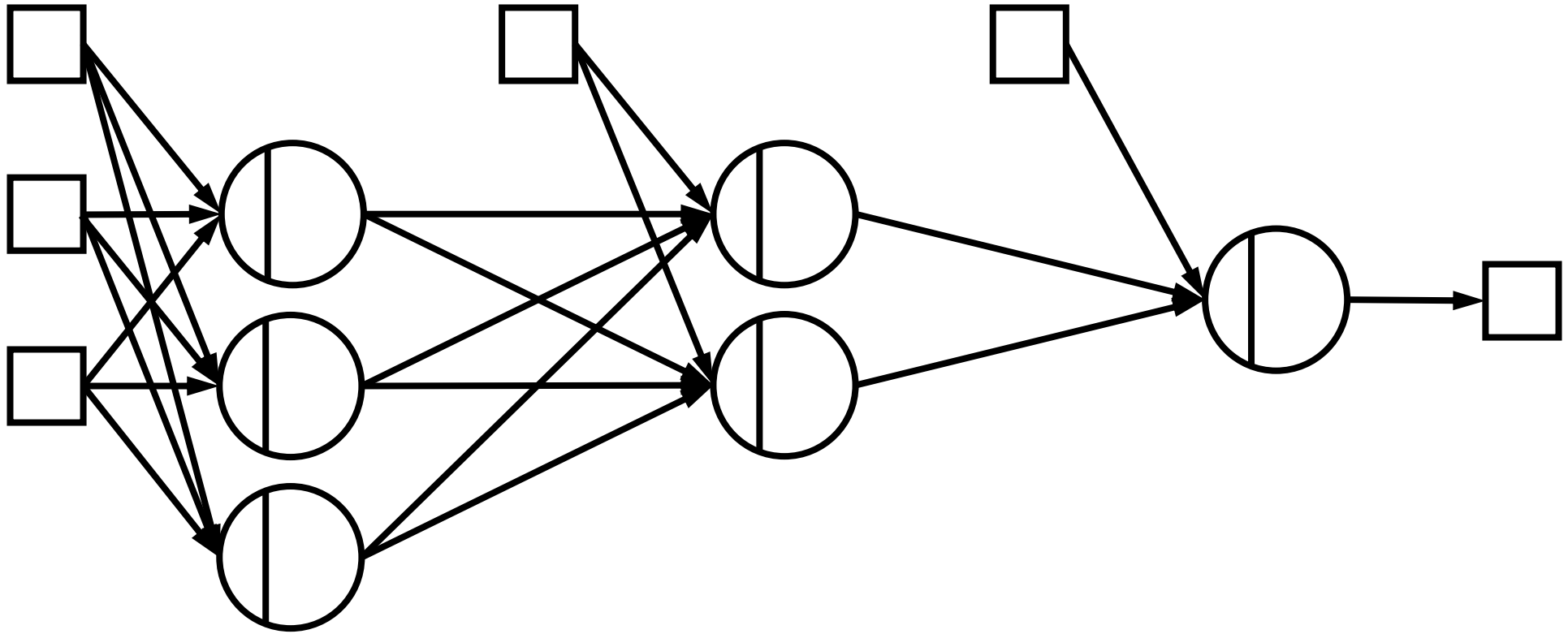
ニューラルネットワーク



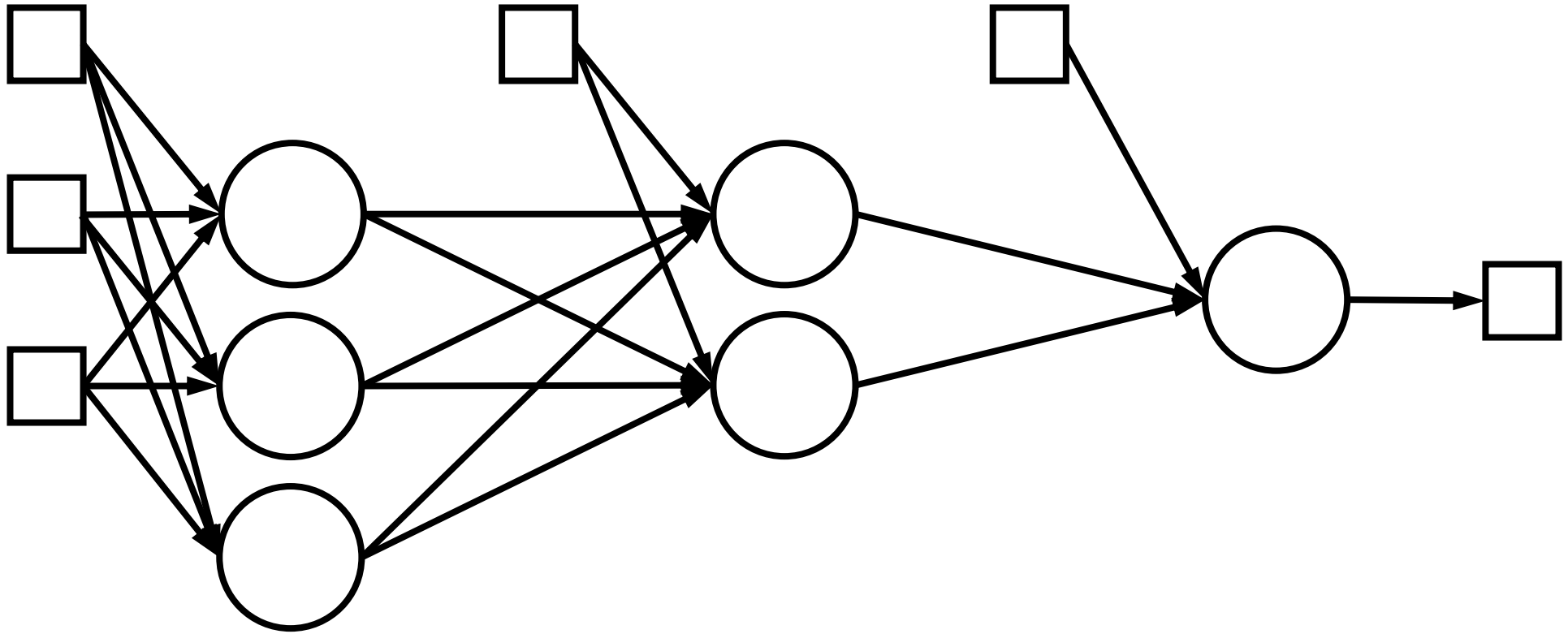
ニューラルネットワーク



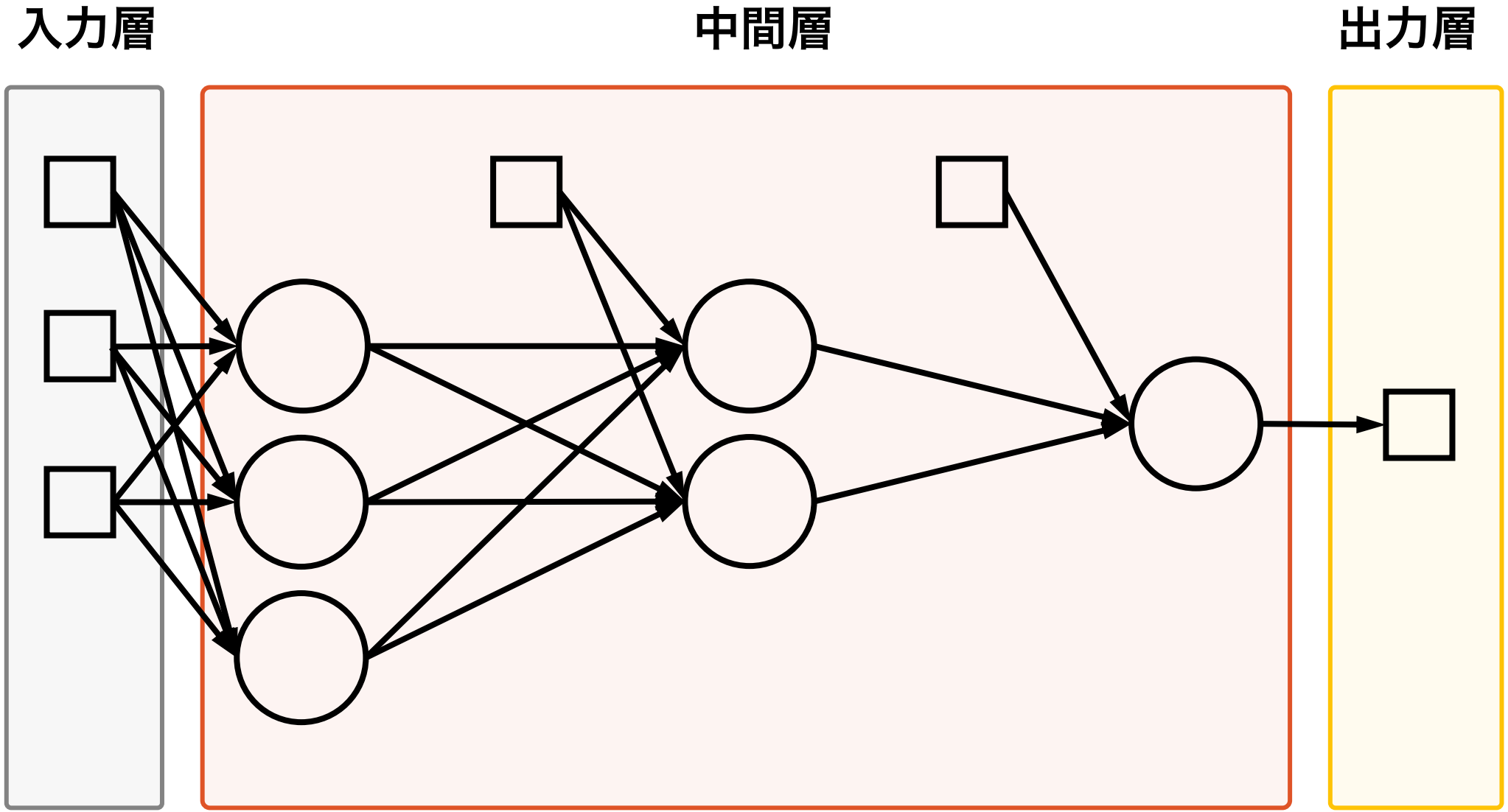
ニューラルネットワーク



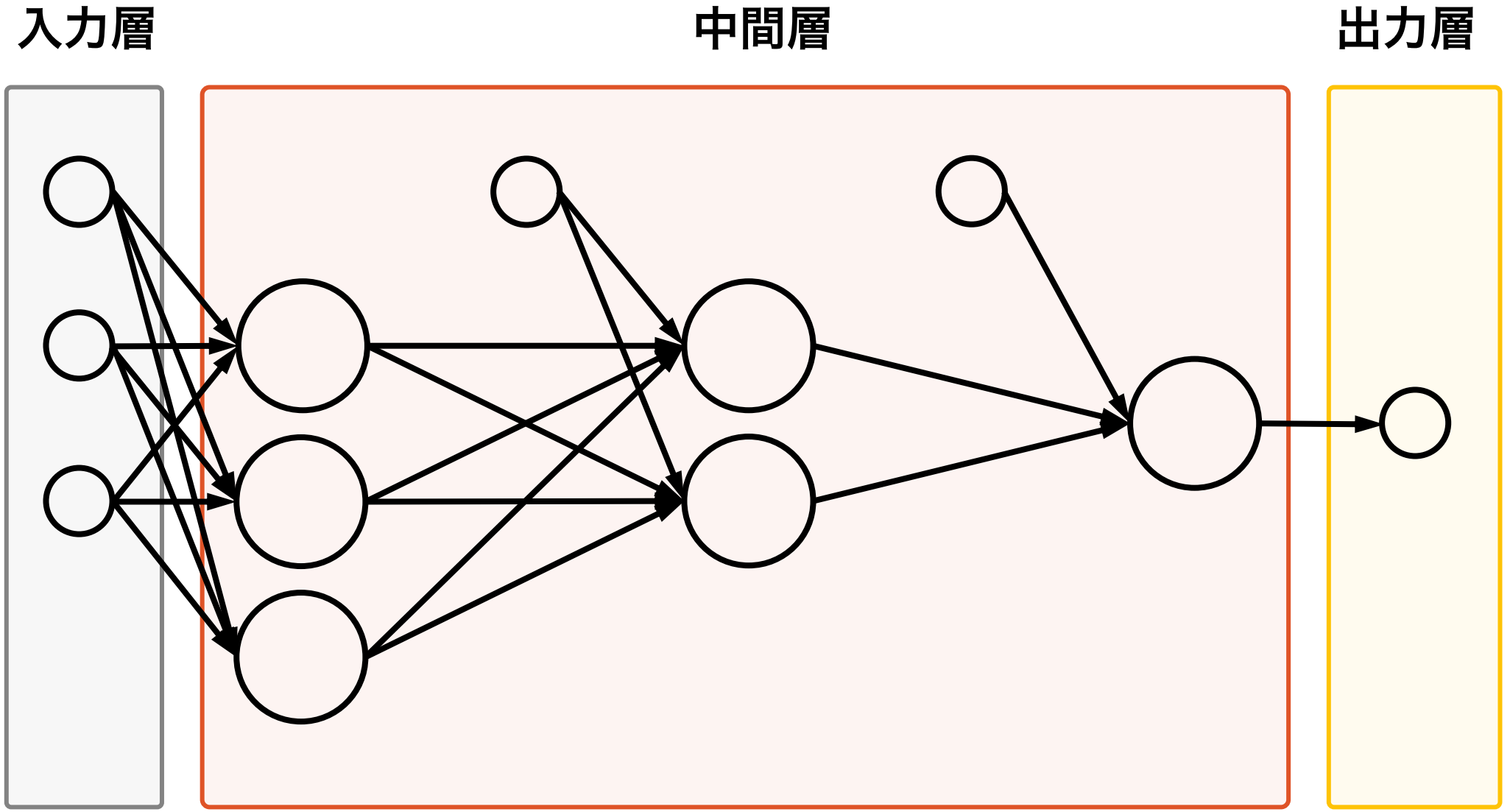
ニューラルネットワーク



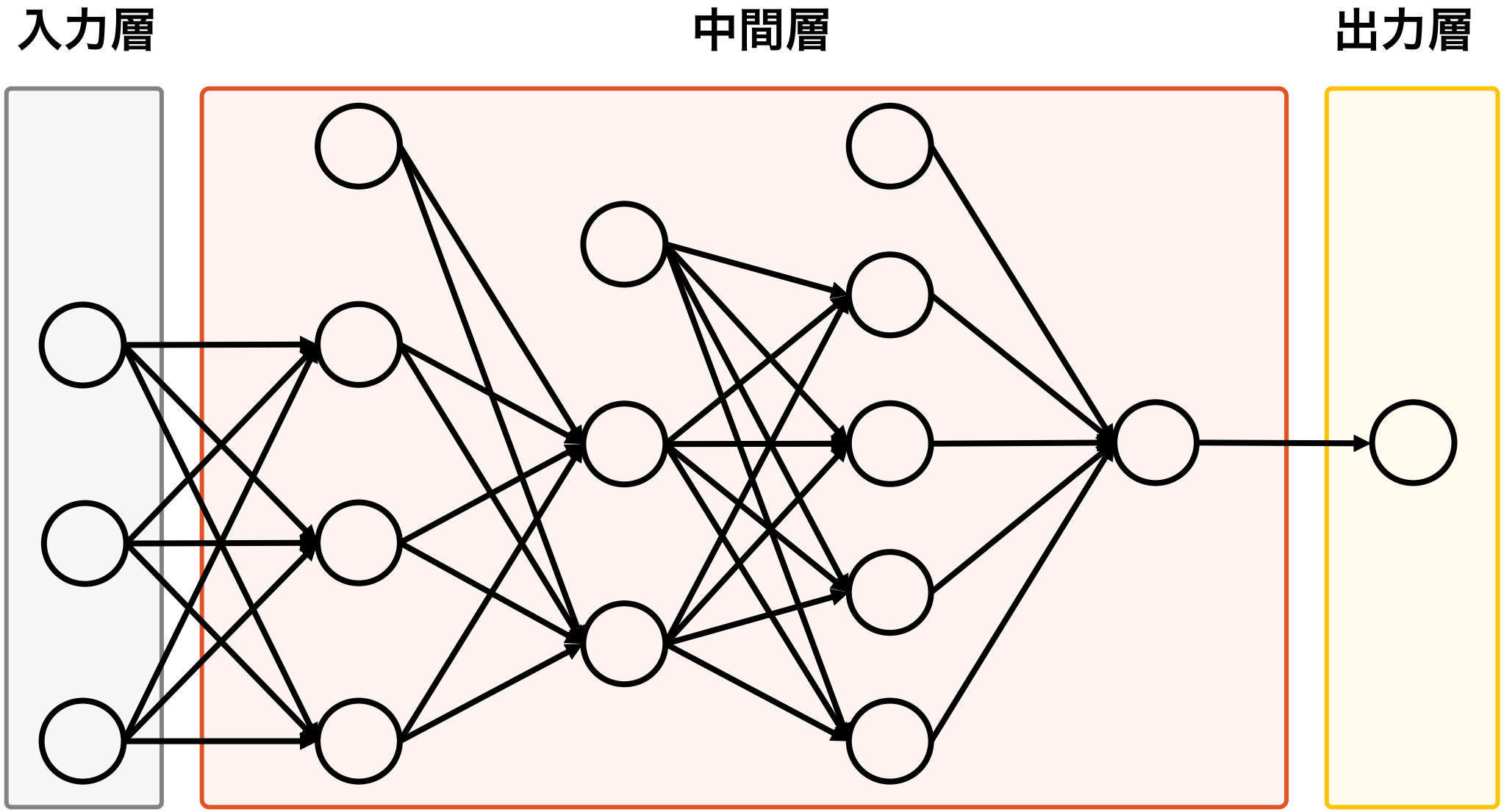
ニューラルネットワーク



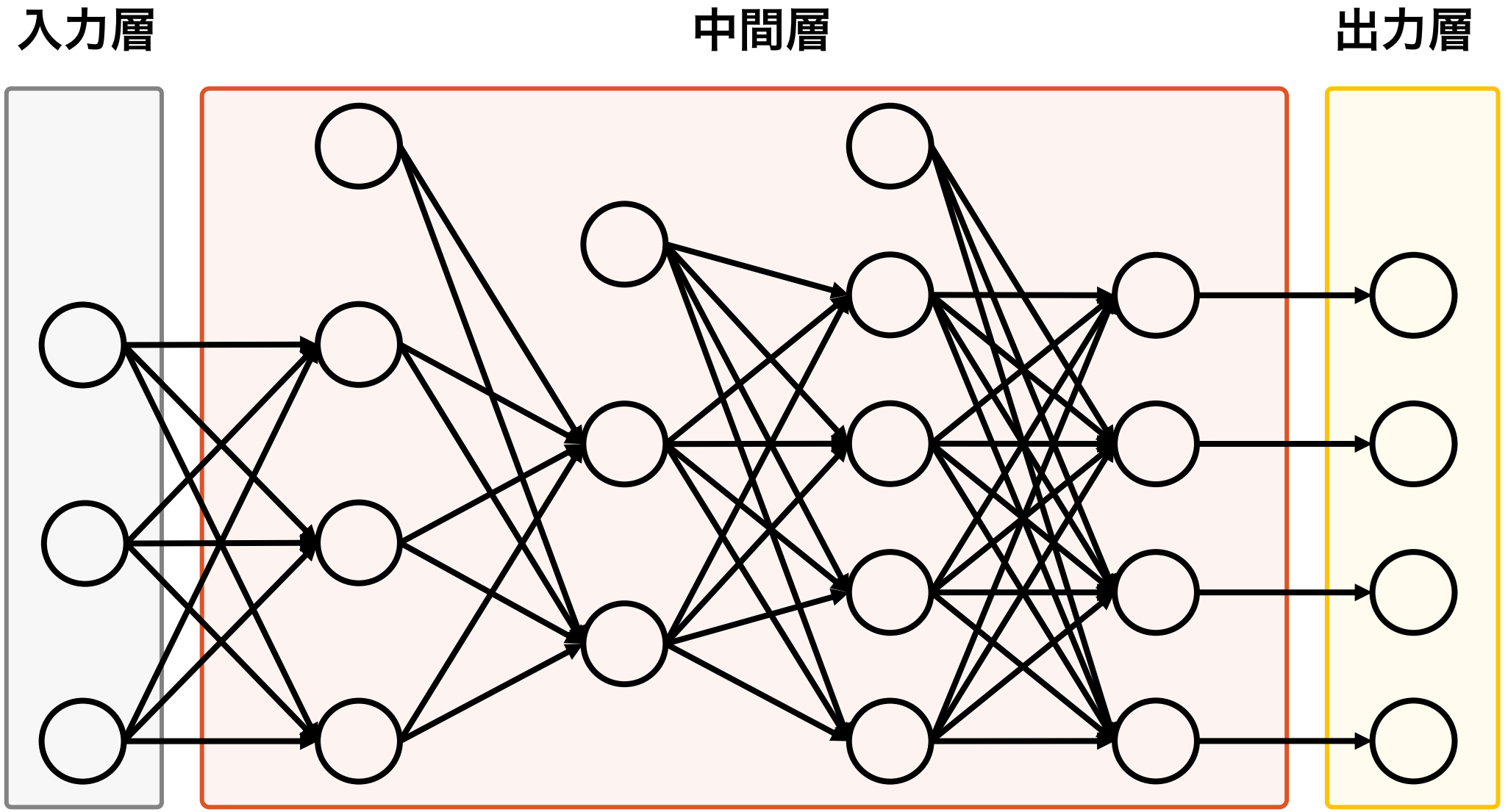
ニューラルネットワーク



ニューラルネットワーク



ニューラルネットワーク

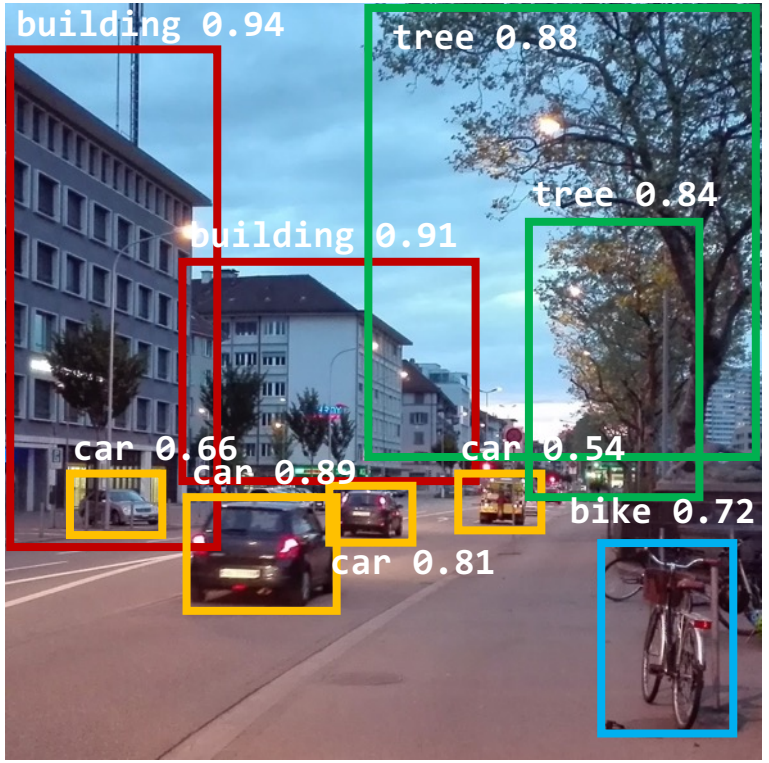


ニューラルネットワーク

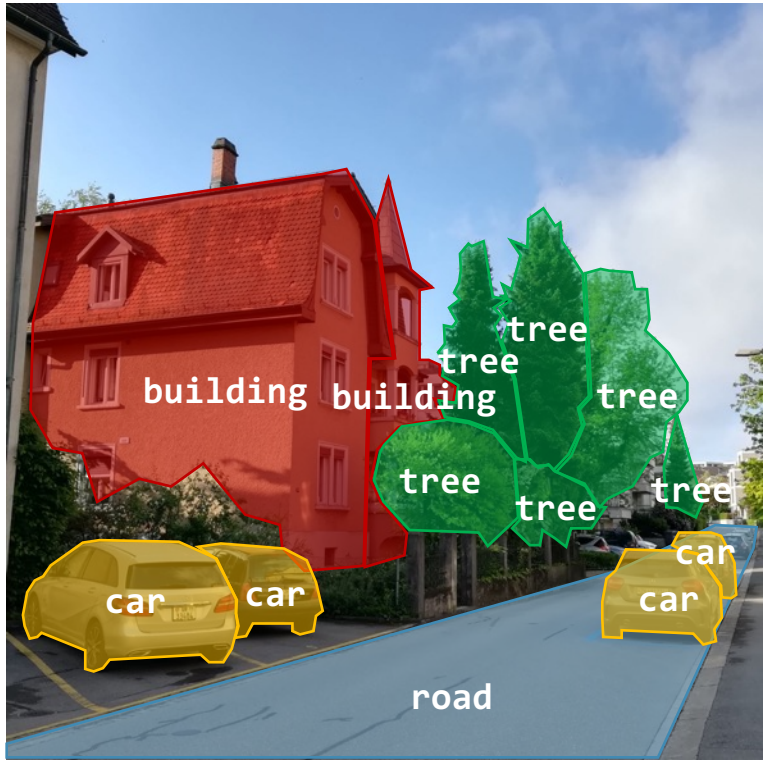
- パーセプトロン
- ニューラルネットワーク
- CNN
- RNN

物体認識

物体検出



セグメンテーション

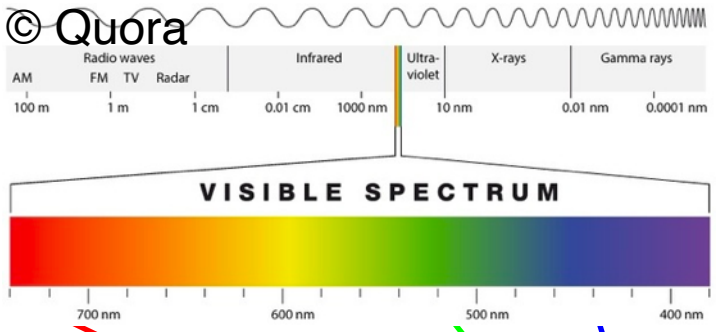


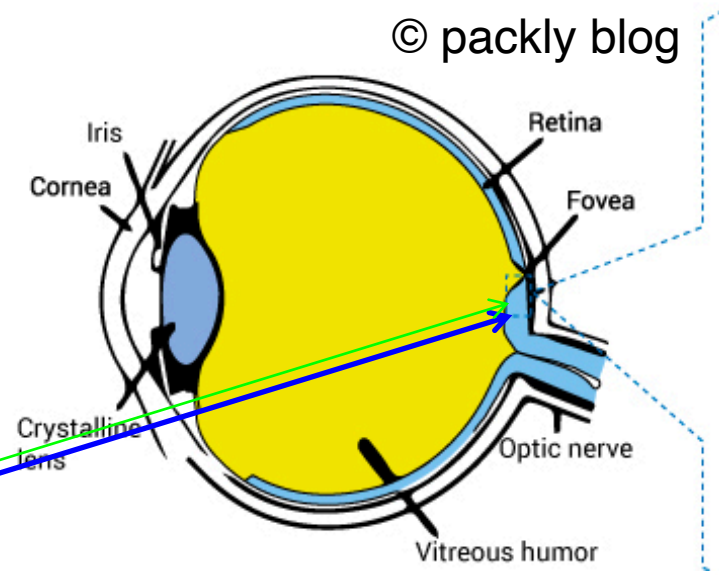
物体認識

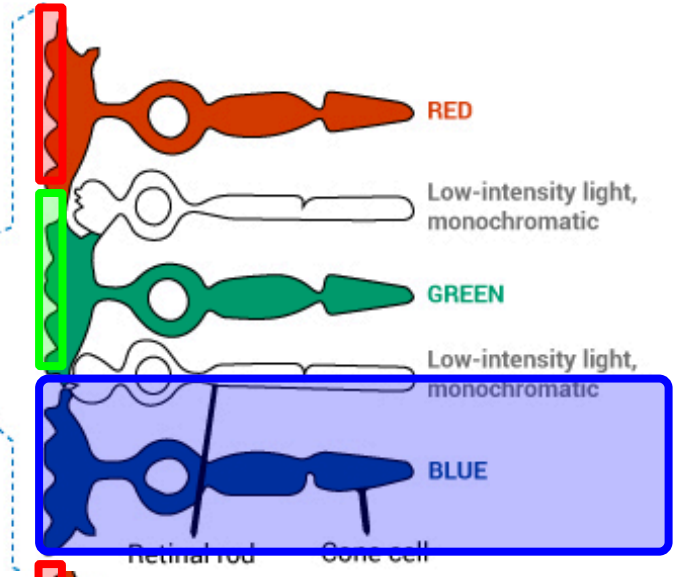
物体検出

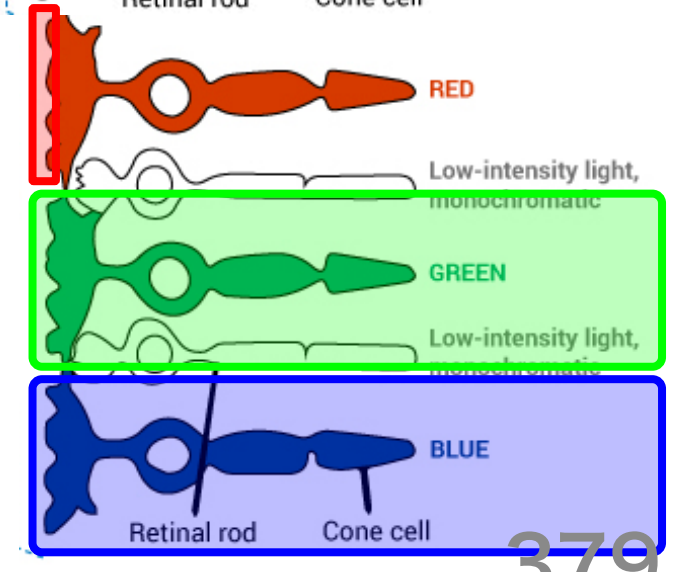
セグメンテーション

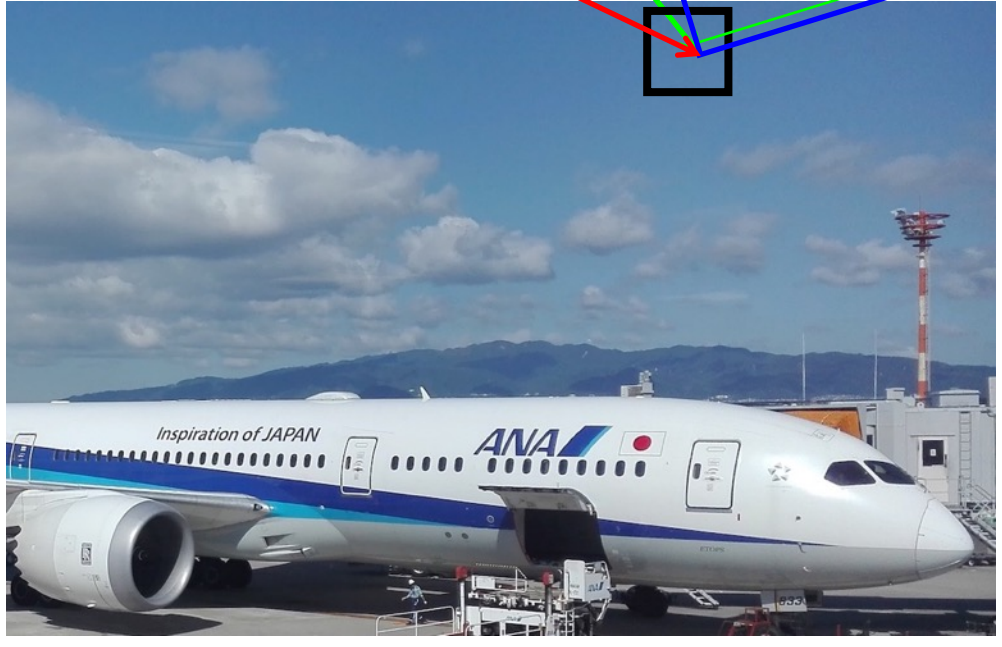
物体認識





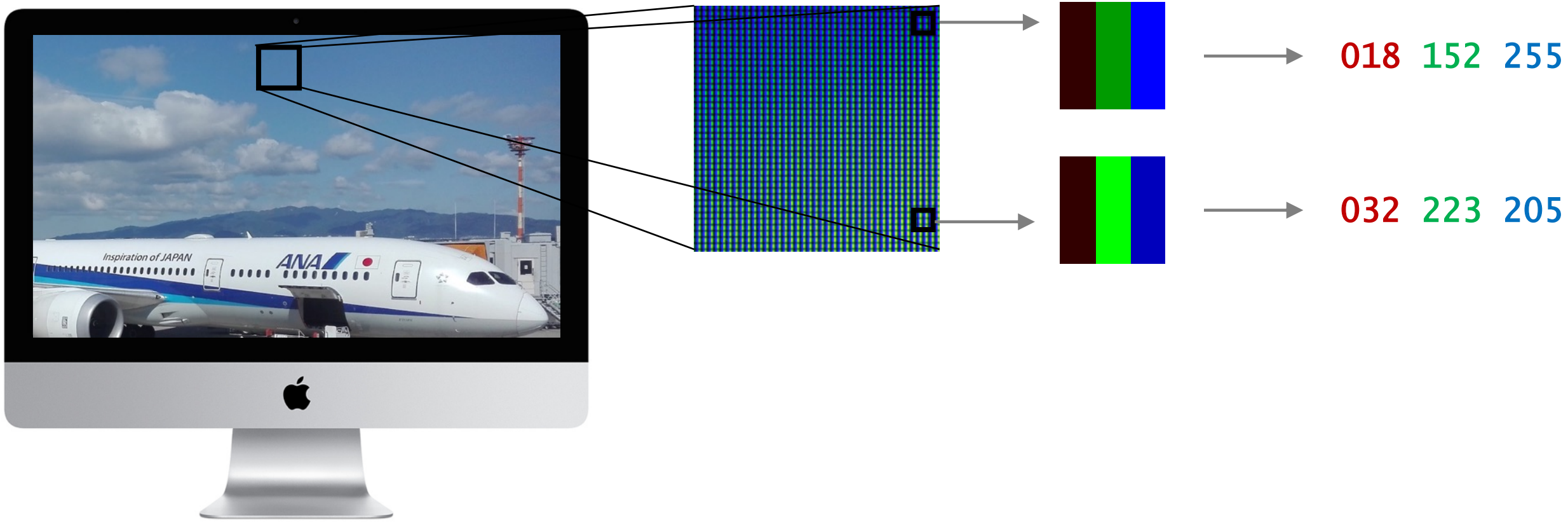






青色反射光が強いので、ほぼすべての青色桿体細胞の神経が励起する。緑色反射光がそれほど強くないので、一部だけの緑色桿体細胞が励起する。赤色反射光がほとんどないので、赤色反対細胞がほとんど励起しない。脳は、励起する桿体細胞の割合を総合判断し、色として認識する。

物体認識



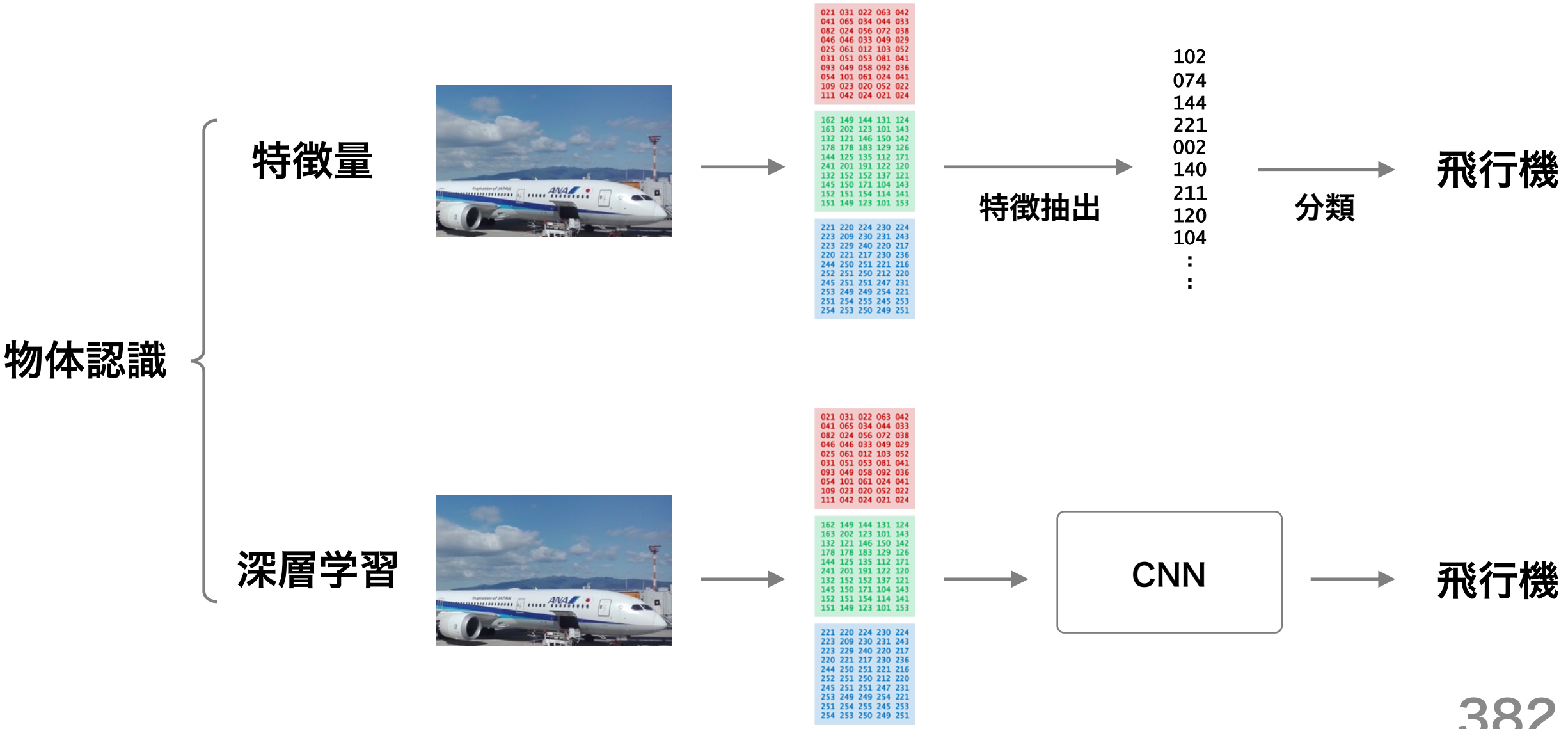
物体認識

021	031	022	063	042
041	065	034	044	033
082	024	056	072	038
046	046	033	049	029
025	061	012	103	052
031	051	053	081	041
093	049	058	092	036
054	101	061	024	041
109	023	020	052	022
111	042	024	021	024

162	149	144	131	124
163	202	123	101	143
132	121	146	150	142
178	178	183	129	126
144	125	135	112	171
241	201	191	122	120
132	152	152	137	121
145	150	171	104	143
152	151	154	114	141
151	149	123	101	153

221	220	224	230	224
223	209	230	231	243
223	229	240	220	217
220	221	217	230	236
244	250	251	221	216
252	251	250	212	220
245	251	251	247	231
253	249	249	254	221
251	254	255	245	253
254	253	250	249	251

物体認識


物体認識
特徴量
深層学習
特徴抽出
102
074
144
221
002
140
211
120
104
分類
飛行機
CNN
飛行機

物体認識（特徴量）

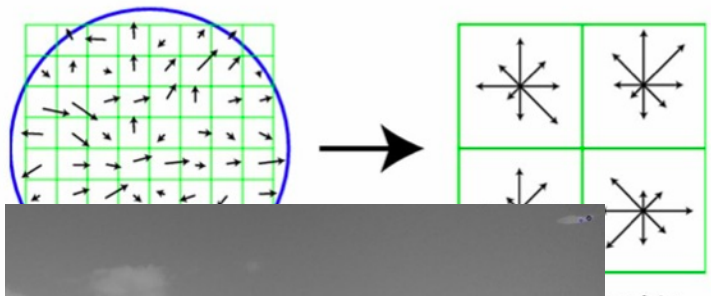
021	031	022	063	042
041	065	034	044	033
082	024	056	072	038
046	046	033	049	029
025	061	012	103	052
031	051	053	081	041
093	049	058	092	036
054	101	061	024	041
109	023	020	052	022
111	042	024	021	024

162	149	144	131	124
163	202	123	101	143
132	121	146	150	142
178	178	183	129	126
144	125	135	112	171
241	201	191	122	120
132	152	152	137	121
145	150	171	104	143
152	151	154	114	141
151	149	123	101	153

221	220	224	230	224
223	209	230	231	243
223	229	240	220	217
220	221	217	230	236
244	250	251	221	216
252	251	250	212	220
245	251	251	247	231
253	249	249	254	221
251	254	255	245	253
254	253	250	249	251

特徴抽出

SIFT　HoG　textons　RIFT　GLOH



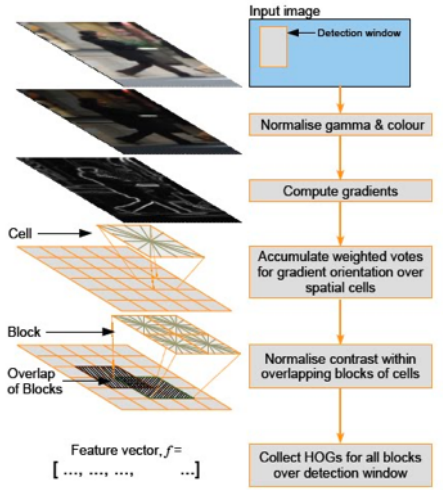


Fig. 3.3. An overview of static HOG feature extraction. The detector window is tiled with a grid of overlapping **blocks**. *Each block contains a grid of spatial* **cells**. *For each cell, the weighted vote of image gradients in orientation histograms is performed. These are locally normalised and collected in one big feature vector.*

物体認識（特徴量）

021	031	022	063	042
041	065	034	044	033
082	024	056	072	038
046	046	033	049	029
025	061	012	103	052
031	051	053	081	041
093	049	058	092	036
054	101	061	024	041
109	023	020	052	022
111	042	024	021	024

162	149	144	131	124
163	202	123	101	143
132	121	146	150	142
178	178	183	129	126
144	125	135	112	171
241	201	191	122	120
132	152	152	137	121
145	150	171	104	143
152	151	154	114	141
151	149	123	101	153

221	220	224	230	224
223	209	230	231	243
223	229	240	220	217
220	221	217	230	236
244	250	251	221	216
252	251	250	212	220
245	251	251	247	231
253	249	249	254	221
251	254	255	245	253
254	253	250	249	251

特徴抽出

102
074
144
221
002
211
104
085
101
113
032
111
046
210
142
228
107
123
192
126
114
125
135
171
201

物体認識（特徴量）

021	031	022	063	042
041	065	034	044	033
082	024	056	072	038
046	046	033	049	029
025	061	012	103	052
031	051	053	081	041
093	049	058	092	036
054	101	061	024	041
109	023	020	052	022
111	042	024	021	024

162	149	144	131	124
163	202	123	101	143
132	121	146	150	142
178	178	183	129	126
144	125	135	112	171
241	201	191	122	120
132	152	152	137	121
145	150	171	104	143
152	151	154	114	141
151	149	123	101	153

221	220	224	230	224
223	209	230	231	243
223	229	240	220	217
220	221	217	230	236
244	250	251	221	216
252	251	250	212	220
245	251	251	247	231
253	249	249	254	221
251	254	255	245	253
254	253	250	249	251

特徴抽出


102	002
074	103
144	211
221	104
002	085
211	101
104	113
085	032
101	144
113	221
032	052
111	111
046	146
210	104
142	142
228	228
107	107
123	123
192	015
126	135
114	211
125	201
135	192
171	162
201	141

物体認識（特徴量）

021	031	022	063	042
041	065	034	044	033
082	024	056	072	038
046	046	033	049	029
025	061	012	103	052
031	051	053	081	041
093	049	058	092	036
054	101	061	024	041
109	023	020	052	022
111	042	024	021	024

162	149	144	131	124
163	202	123	101	143
132	121	146	150	142
178	178	183	129	126
144	125	135	112	171
241	201	191	122	120
132	152	152	137	121
145	150	171	104	143
152	151	154	114	141
151	149	123	101	153

221	220	224	230	224
223	209	230	231	243
223	229	240	220	217
220	221	217	230	236
244	250	251	221	216
252	251	250	212	220
245	251	251	247	231
253	249	249	254	221
251	254	255	245	253
254	253	250	249	251

特徴抽出



102	002	141
074	103	146
144	211	104
221	104	142
002	085	002
211	101	103
104	113	211
085	032	144
101	144	221
113	221	052
032	052	228
111	111	107
046	146	123
210	104	015
142	142	135
228	228	211
107	107	201
123	123	192
192	015	162
126	135	141
114	211	104
125	201	085
135	192	101
171	162	113
201	141	032

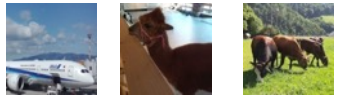
物体認識（特徴量）

021	031	022	063	042
041	065	034	044	033
082	024	056	072	038
046	046	033	049	029
025	061	012	103	052
031	051	053	081	041
093	049	058	092	036
054	101	061	024	041
109	023	020	052	022
111	042	024	021	024

162	149	144	131	124
163	202	123	101	143
132	121	146	150	142
178	178	183	129	126
144	125	135	112	171
241	201	191	122	120
132	152	152	137	121
145	150	171	104	143
152	151	154	114	141
151	149	123	101	153

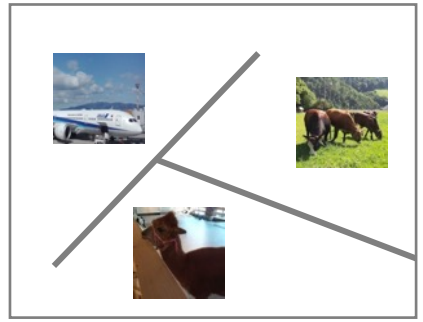
221	220	224	230	224
223	209	230	231	243
223	229	240	220	217
220	221	217	230	236
244	250	251	221	216
252	251	250	212	220
245	251	251	247	231
253	249	249	254	221
251	254	255	245	253
254	253	250	249	251

特徴抽出

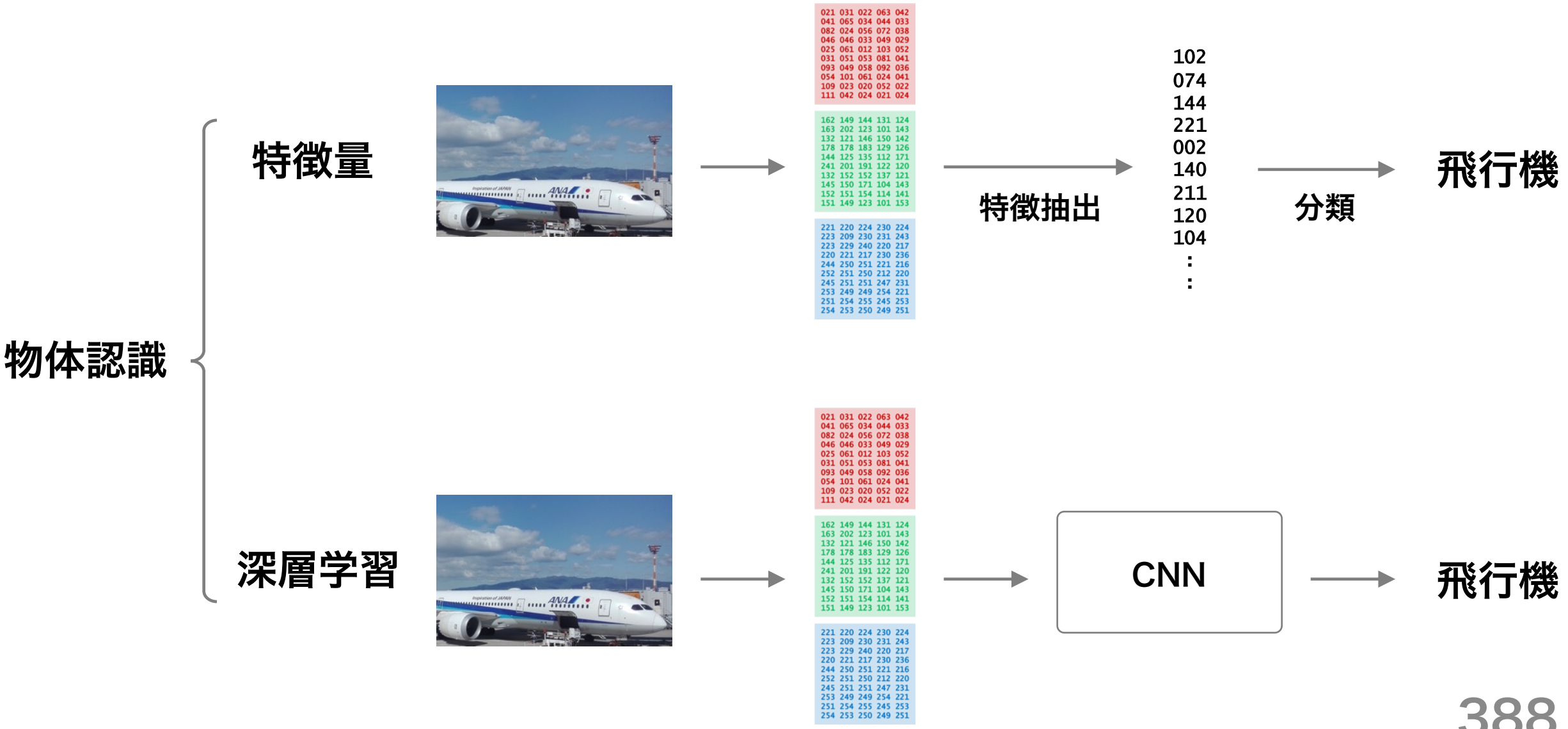


102	002	141
074	103	146
144	211	104
221	104	142
002	085	002
211	101	103
104	113	211
085	032	144
101	144	221
113	221	052
032	052	228
111	111	107
046	146	123
210	104	015
142	142	135
228	228	211
107	107	201
123	123	192
192	015	162
126	135	141
114	211	104
125	201	085
135	192	101
171	162	113
201	141	032

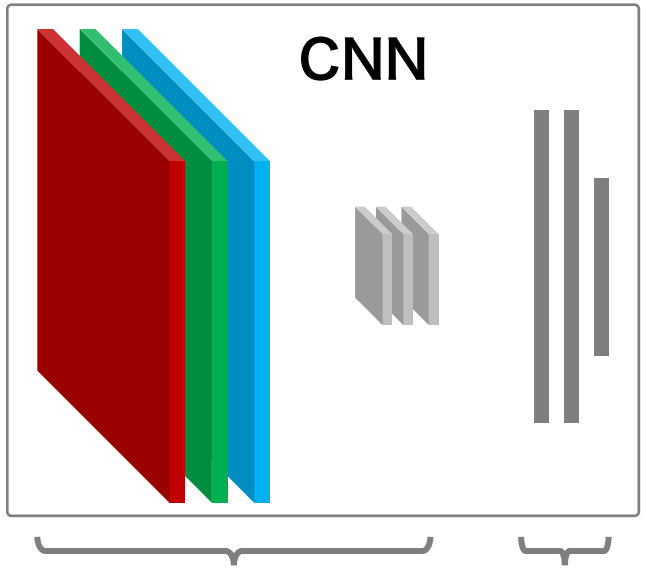
機械学習



物体認識

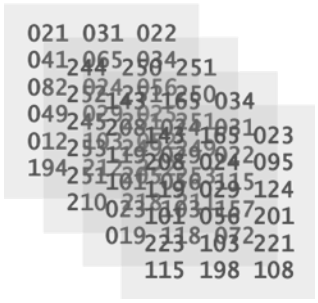

物体認識
特徴量
深層学習
021 031 022 063 042
041 065 034 044 033
082 024 056 072 038
046 046 033 049 029
025 061 012 103 052
031 051 053 081 041
093 049 058 092 036
054 101 061 024 041
109 023 020 052 022
111 042 024 021 024
162 149 144 131 124
163 202 123 101 143
132 121 146 150 142
178 178 183 129 126
144 125 135 112 171
241 201 191 122 120
132 152 152 137 121
145 150 171 104 143
152 151 154 114 141
151 149 123 101 153
221 220 224 230 224
223 209 230 231 243
223 229 240 220 217
220 221 217 230 236
244 250 251 221 216
252 251 250 212 220
245 251 251 247 231
253 249 249 254 221
251 254 255 245 253
254 253 250 249 251
特徴抽出
102
074
144
221
002
140
211
120
104
分類
飛行機
CNN
飛行機

物体認識（深層学習）

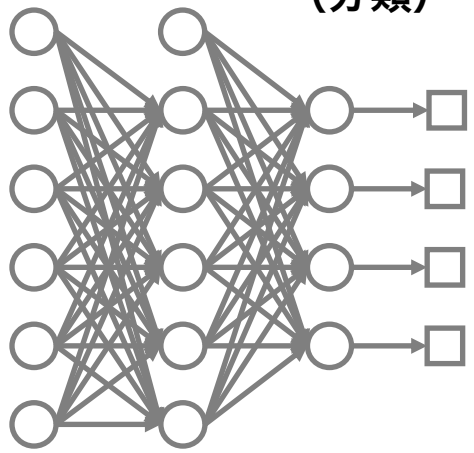



```
airplane 56.1%
sky      25.0%
cloud    12.4%
mountain  6.5%
```

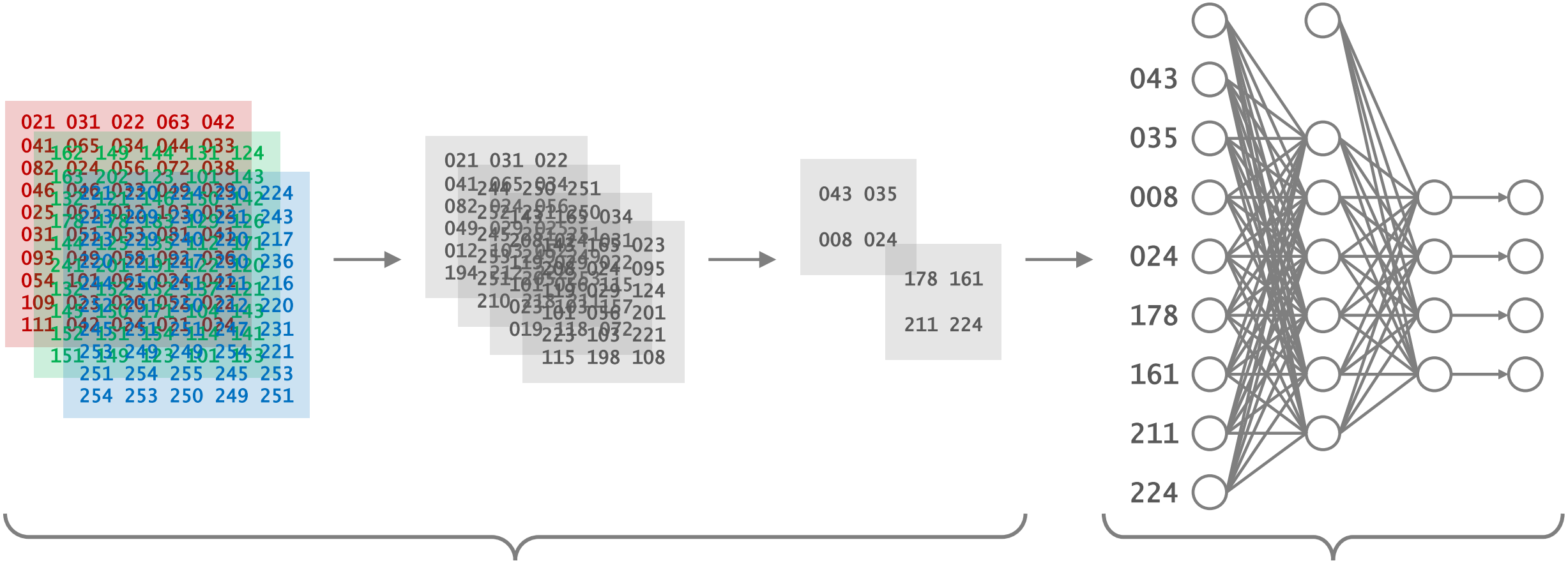
畳み込み
（特徴抽出）



ニューラルネットワーク
（分類）

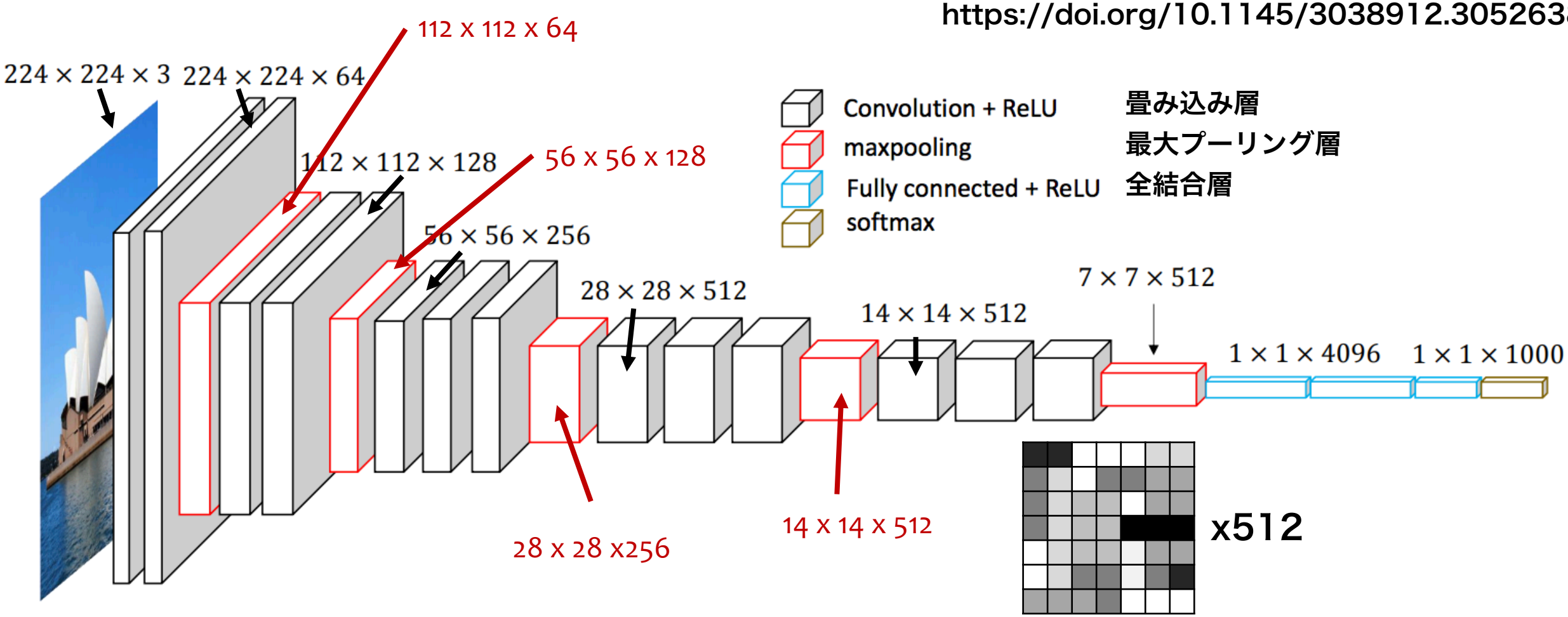


畳み込みニューラルネットワーク



畳み込みニューラルネットワーク

https://doi.org/10.1145/3038912.3052638



畳み込みニューラルネットワーク

畳み込み層

- 画像から特徴量を抽出する層
- 入力層に近い方の畳み込み層は、縦線、横線、斜線などの抽象的な特徴を抽出
- 出力層に近い方の畳み込み層は、顔の形や車の形などの具体的な特徴を抽出

プーリング層

- 重要な特徴量を残しながら画像のサイズを削減する層
- 後に続く処理が少数のパラメーターで計算できるようになり、過学習を抑制

全結合層

- 畳み込み層およびプーリング層から抽出された特徴量をニューラルネットワークで分類する層

Grayscale Photo

162	149	144	131	124
163	202	123	101	143
132	121	146	150	142
178	178	183	129	126
144	125	135	112	171
241	201	191	122	120
132	152	152	137	121
145	150	171	104	143
152	151	154	114	141
151	149	123	101	153

1 channel

Color Photo

021	031	022	063	042
041	065	034	044	033
082	024	056	072	038
046	046	033	049	029
025	061	013	103	053
031	051	053	081	041
093	049	058	093	036
054	101	065	024	042
109	023	026	053	023
111	042	024	021	024

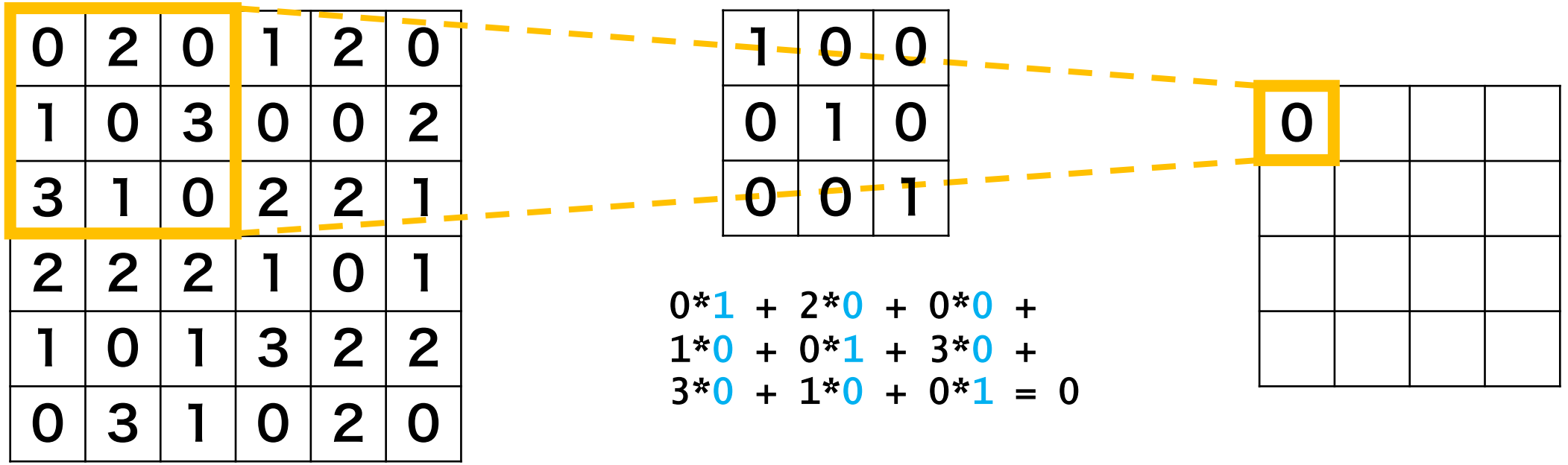
162	149	144	131	124
163	202	123	101	143
132	121	146	150	142
178	178	183	129	126
144	125	135	112	171
241	201	191	122	120
132	152	152	137	121
145	150	171	104	143
152	151	154	114	141
151	149	123	101	153

224	243	217	236	216
220	231	221		

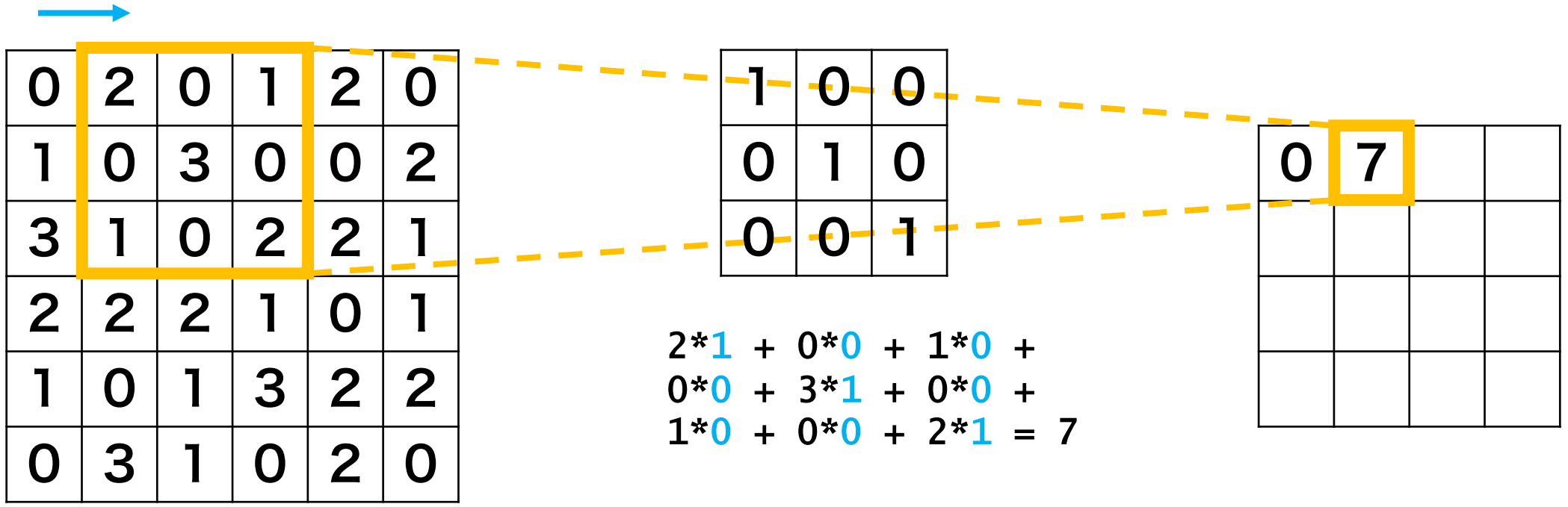
251	254	255	245	253
254	253	250	249	251

3 channels

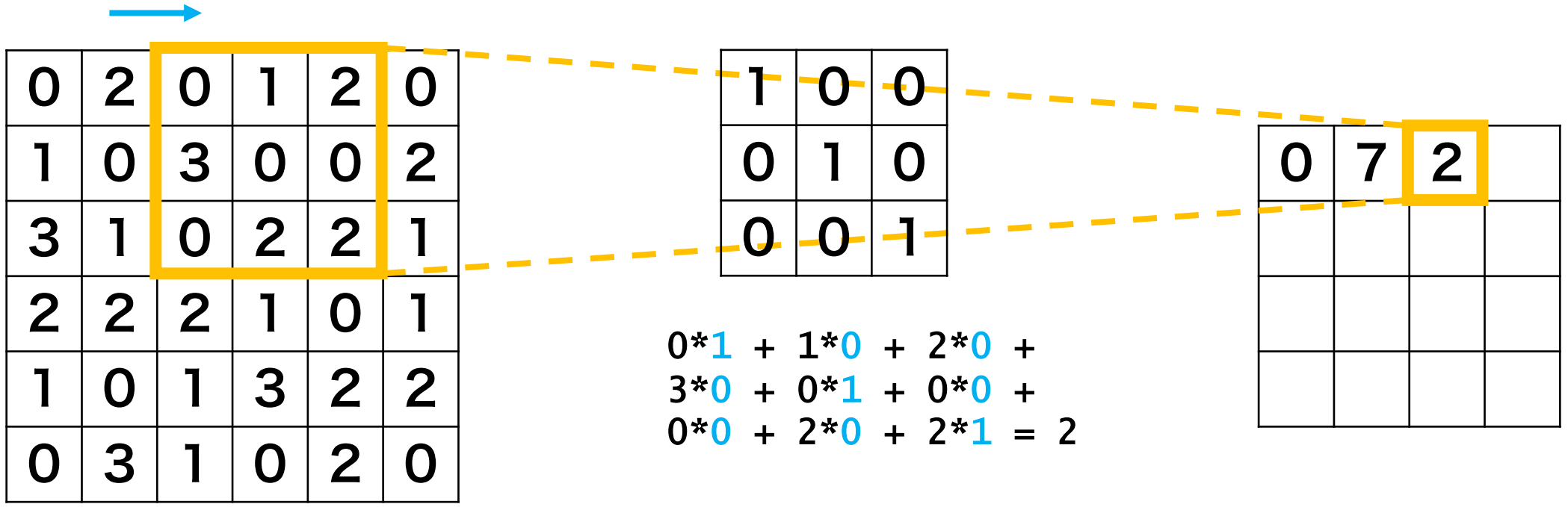
畳み込み演算


0 2 0 1 2 0
1 0 3 0 0 2
3 1 0 2 2 1
2 2 2 1 0 1
1 0 1 3 2 2
0 3 1 0 2 0
1 0 0
0 1 0
0 0 1
0*1 + 2*0 + 0*0 +
1*0 + 0*1 + 3*0 +
3*0 + 1*0 + 0*1 = 0
0

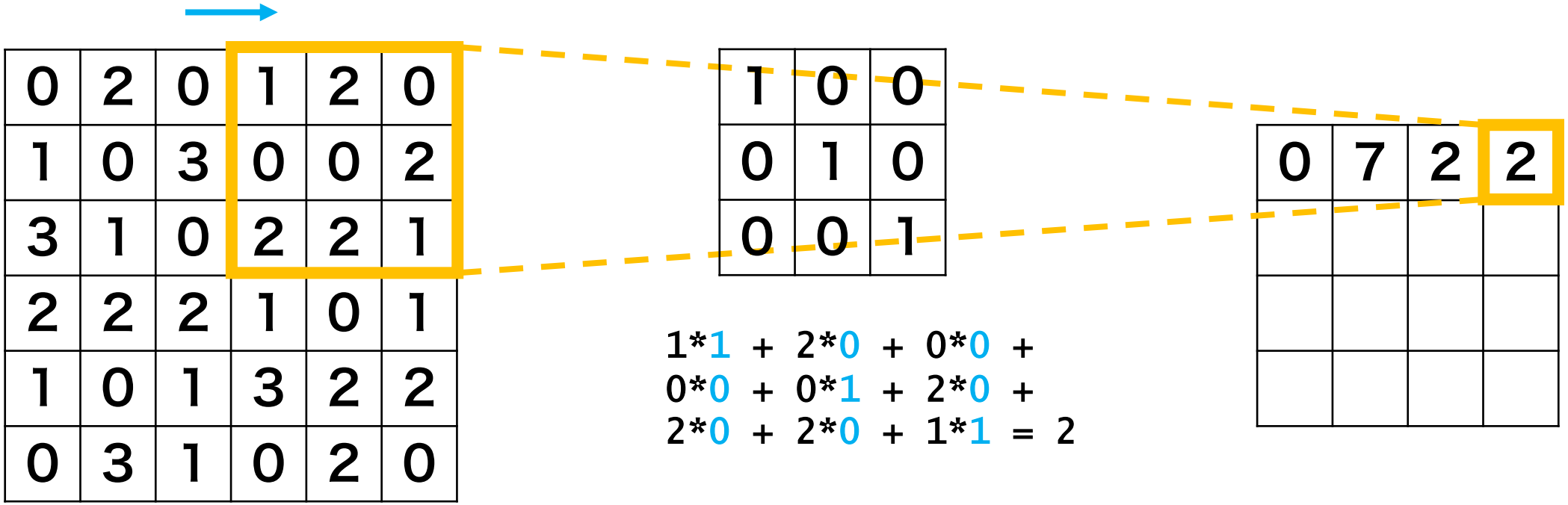
畳み込み演算


0 2 0 1 2 0
1 0 3 0 0 2
3 1 0 2 2 1
2 2 2 1 0 1
1 0 1 3 2 2
0 3 1 0 2 0
1 0 0
0 1 0
0 0 1
2*1 + 0*0 + 1*0 +
0*0 + 3*1 + 0*0 +
1*0 + 0*0 + 2*1 = 7
0 7

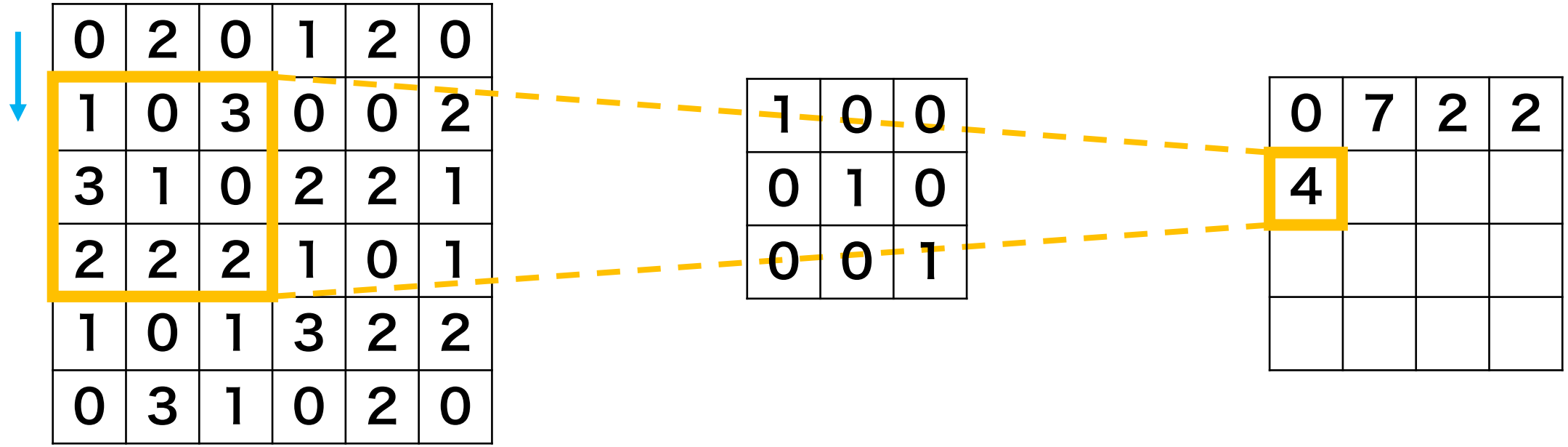
畳み込み演算


0 2 0 1 2 0
1 0 3 0 0 2
3 1 0 2 2 1
2 2 2 1 0 1
1 0 1 3 2 2
0 3 1 0 2 0
1 0 0
0 1 0
0 0 1
0*1 + 1*0 + 2*0 +
3*0 + 0*1 + 0*0 +
0*0 + 2*0 + 2*1 = 2
0 7 2

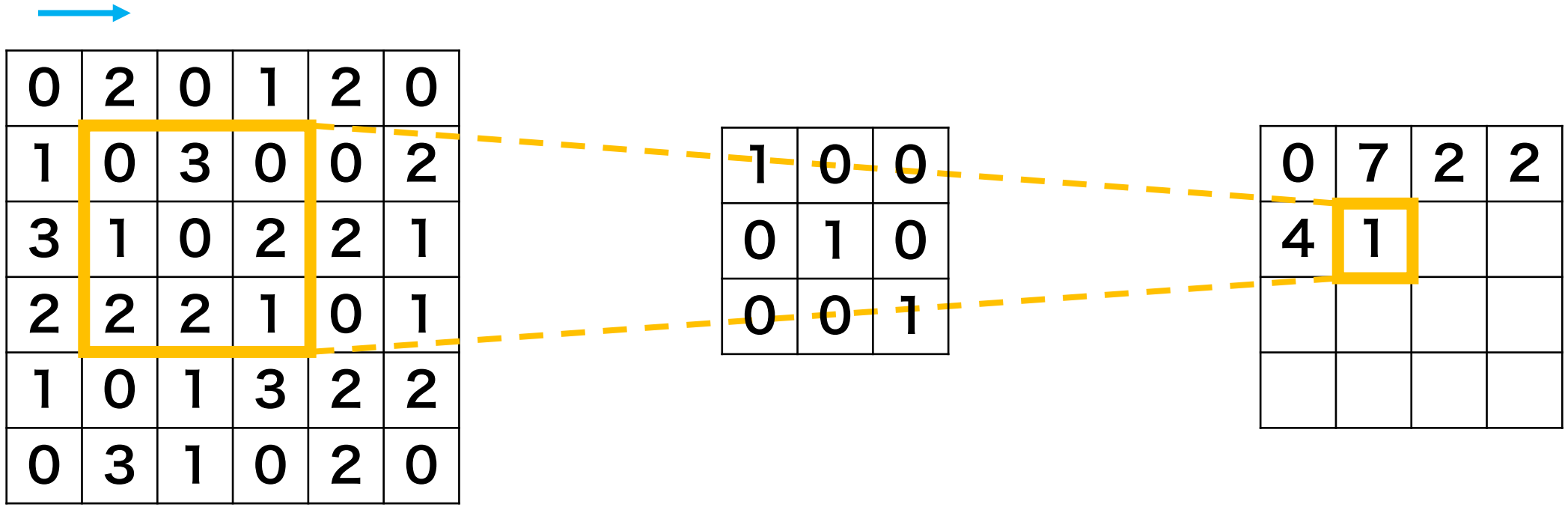
畳み込み演算


0 2 0 1 2 0
1 0 3 0 0 2
3 1 0 2 2 1
2 2 2 1 0 1
1 0 1 3 2 2
0 3 1 0 2 0
1 0 0
0 1 0
0 0 1
1*1 + 2*0 + 0*0 +
0*0 + 0*1 + 2*0 +
2*0 + 2*0 + 1*1 = 2
0 7 2 2

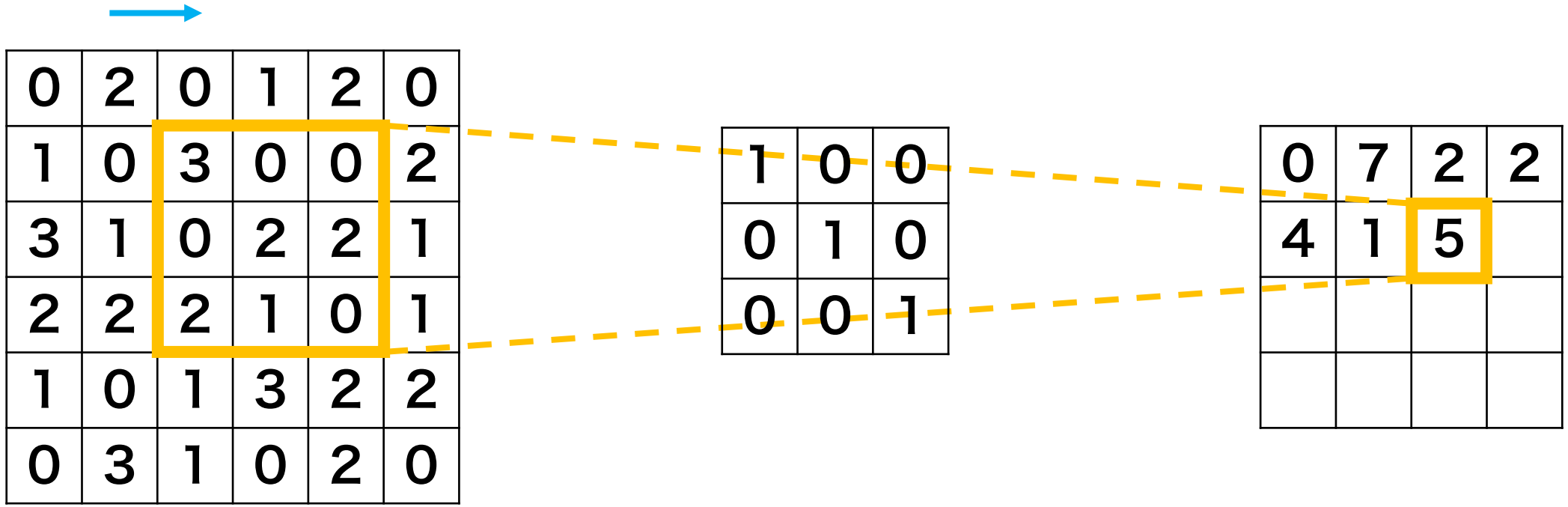
畳み込み演算


0 2 0 1 2 0
1 0 3 0 0 2
3 1 0 2 2 1
2 2 2 1 0 1
1 0 1 3 2 2
0 3 1 0 2 0
1 0 0
0 1 0
0 0 1
0 7 2 2
4

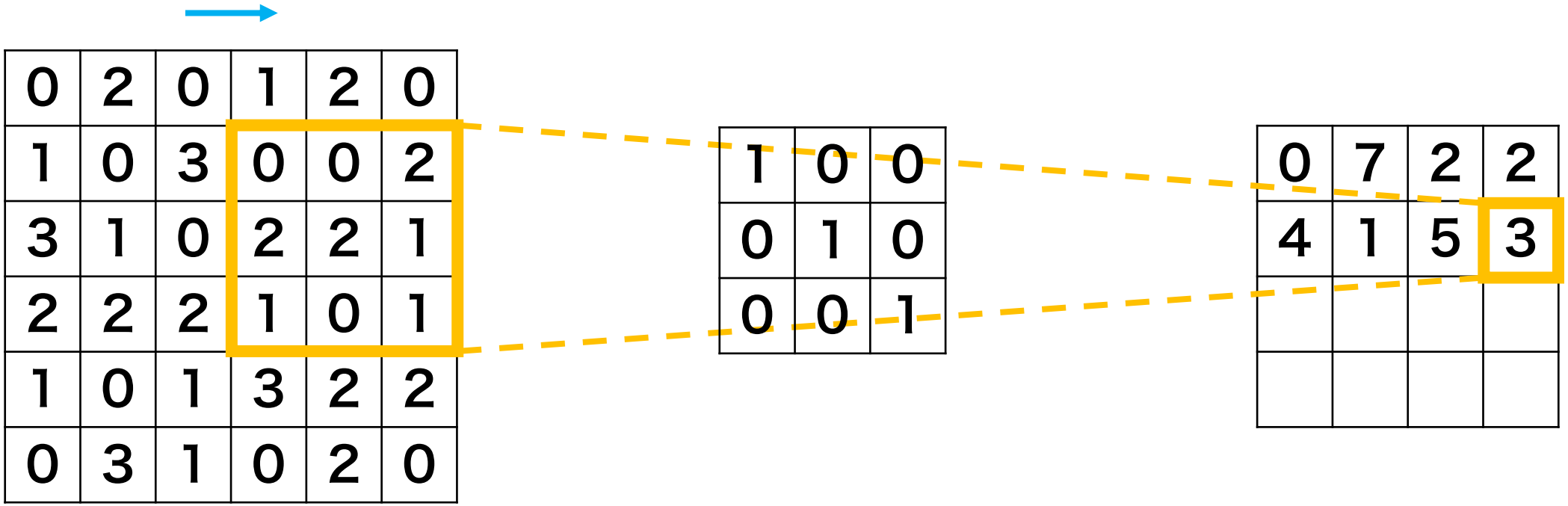
畳み込み演算


0 2 0 1 2 0
1 0 3 0 0 2
3 1 0 2 2 1
2 2 2 1 0 1
1 0 1 3 2 2
0 3 1 0 2 0
1 0 0
0 1 0
0 0 1
0 7 2 2
4 1

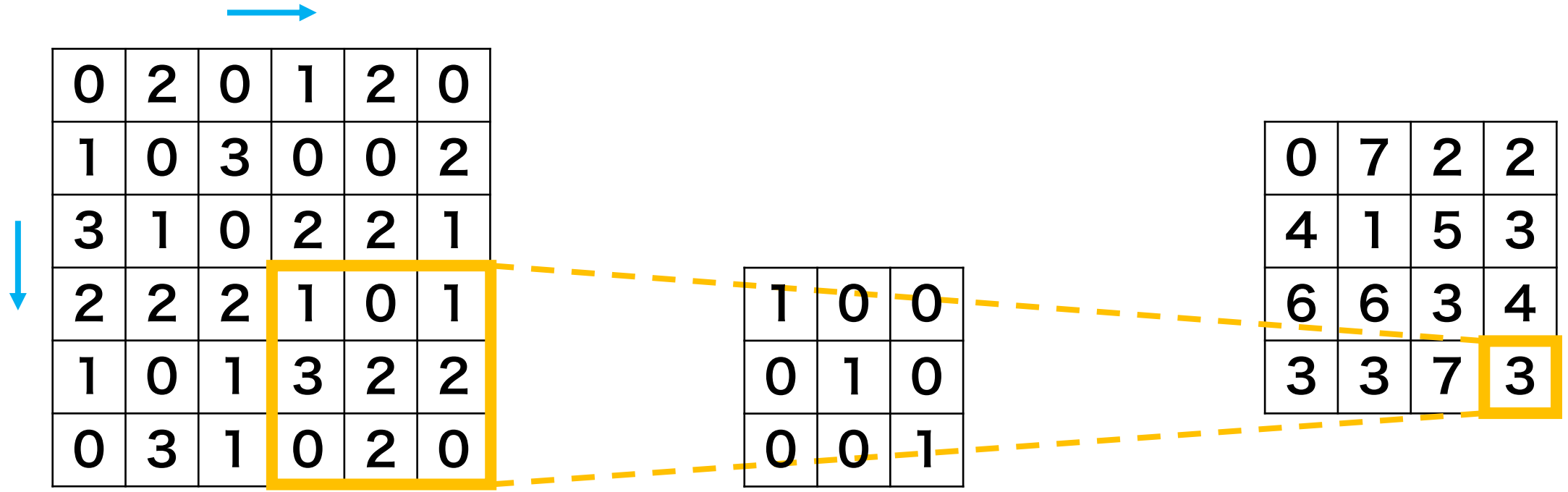
畳み込み演算


0 2 0 1 2 0
1 0 3 0 0 2
3 1 0 2 2 1
2 2 2 1 0 1
1 0 1 3 2 2
0 3 1 0 2 0
1 0 0
0 1 0
0 0 1
0 7 2 2
4 1 5

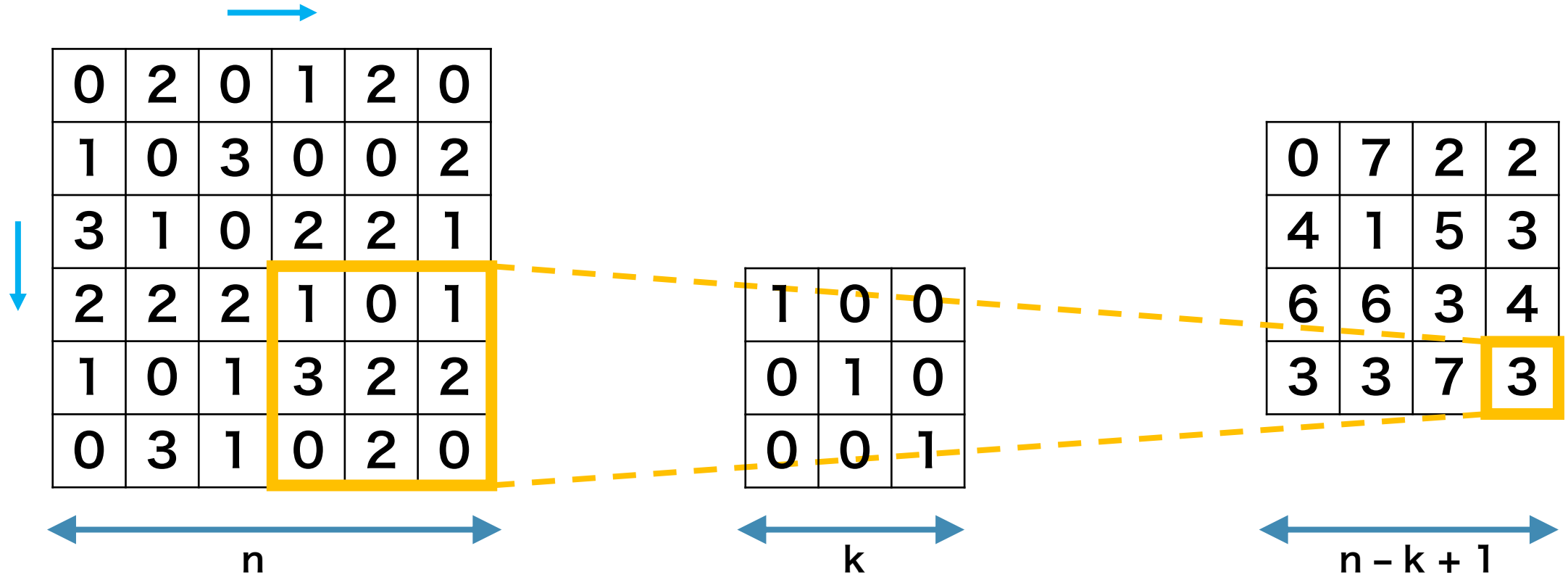
畳み込み演算


0 2 0 1 2 0
1 0 3 0 0 2
3 1 0 2 2 1
2 2 2 1 0 1
1 0 1 3 2 2
0 3 1 0 2 0
1 0 0
0 1 0
0 0 1
0 7 2 2
4 1 5 3

畳み込み演算


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

畳み込み演算



畳み込み演算を行うことによって、6x6 の入力行列が 4x4 の行列に縮小される。そのため、畳み込みを複数回行うことによって、画像サイズが大幅に小さくなるだけでなく、画像中央にある特徴が極端に強調され、画像のエッジにある情報が徐々に消えてしまうリスクが生じる。

畳み込み演算

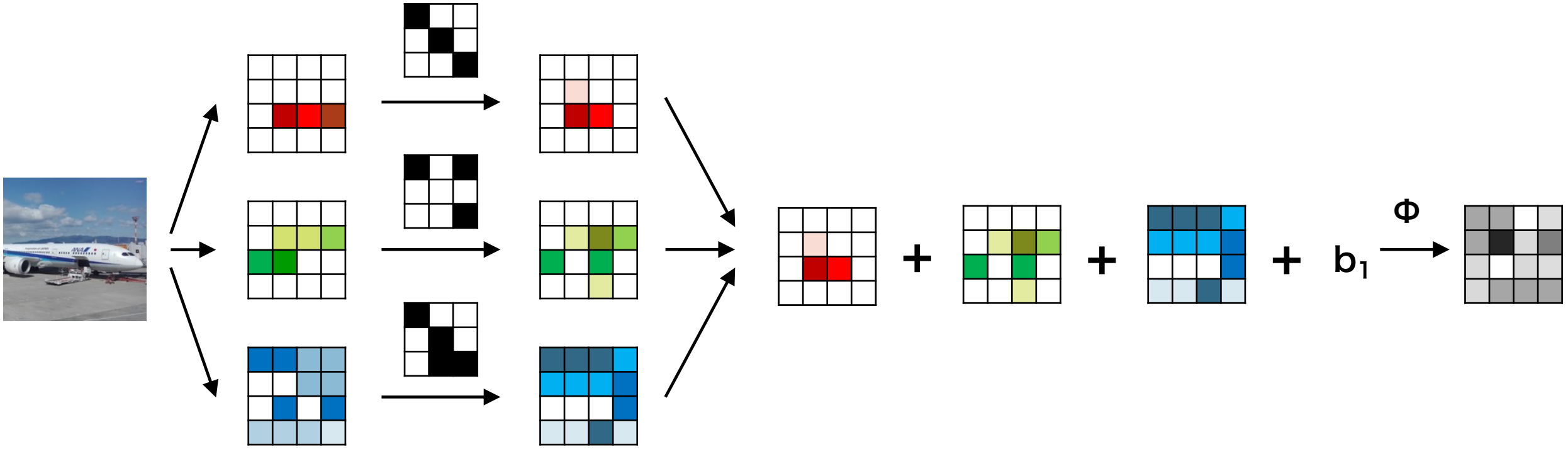
0	0	0	0	0	0	0	0
0	0	2	0	1	2	0	0
0	1	0	3	0	0	2	0
0	3	1	0	2	2	1	0
0	2	2	2	1	0	1	0
0	1	0	1	3	2	2	0
0	0	3	1	0	2	0	0
0	0	0	0	0	0	0	0

1	0	0
0	1	0
0	0	1

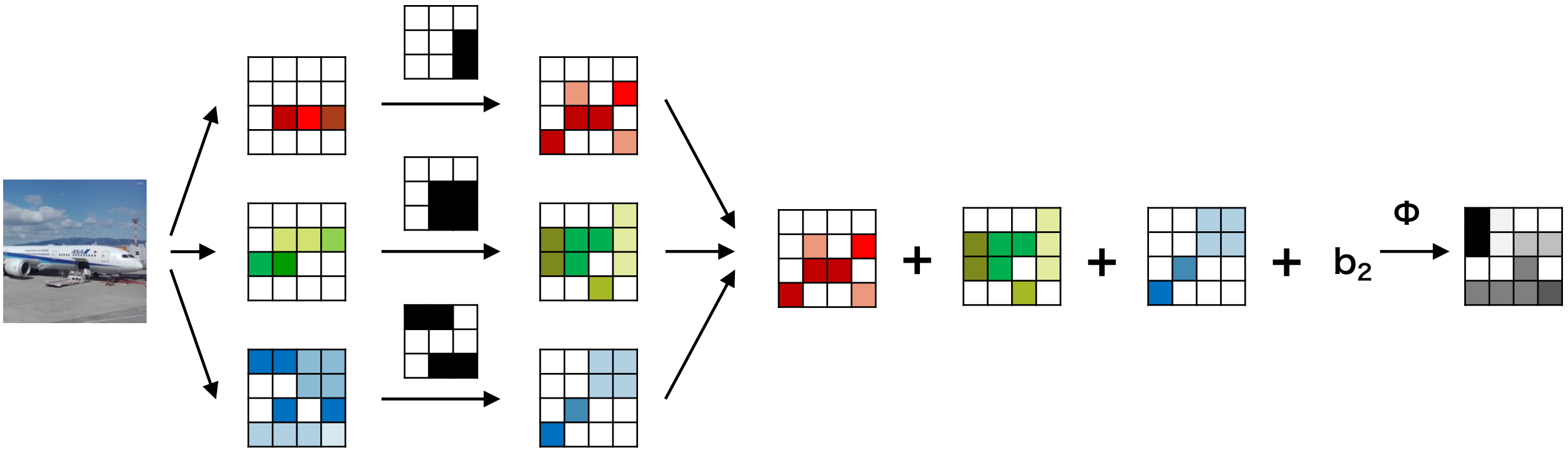
0	5	0	1	2	0
2	0	7	2	2	4
5	4	1	5	3	1
2	6	6	3	4	3
4	3	3	7	3	2
0	4	1	1	5	0

入力行列の周りにゼロを埋めることで、畳み込み後の行列のサイズは変化しない。

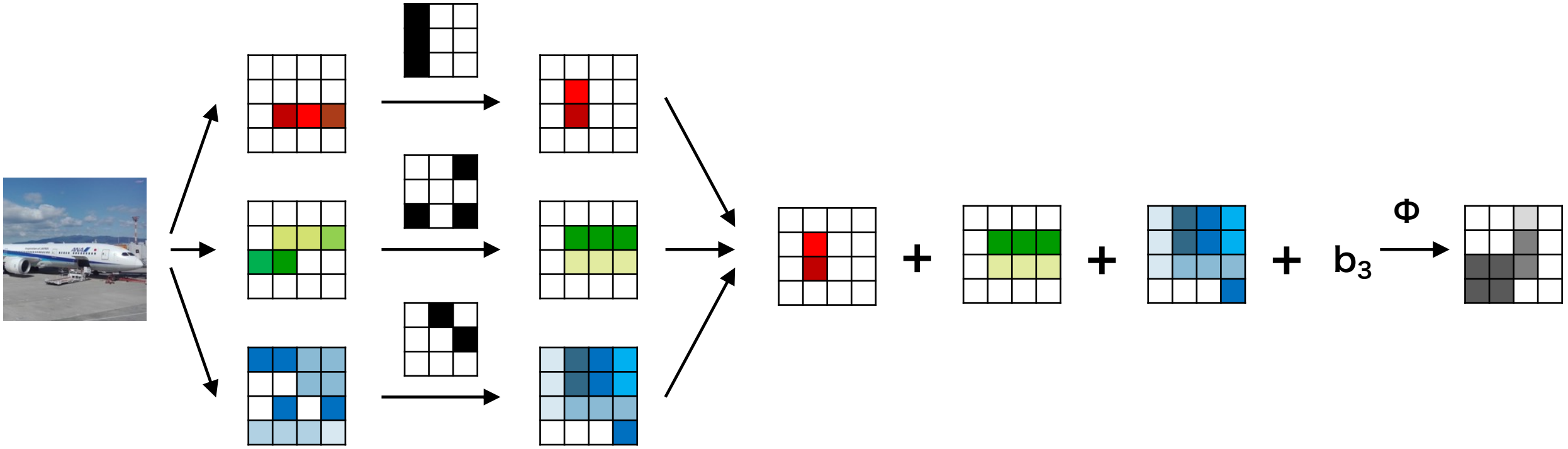
畳み込み演算



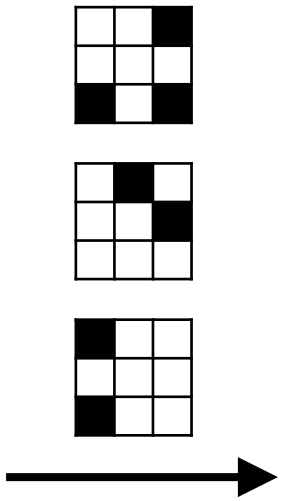
畳み込み演算



畳み込み演算

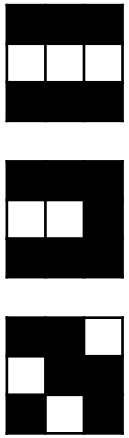


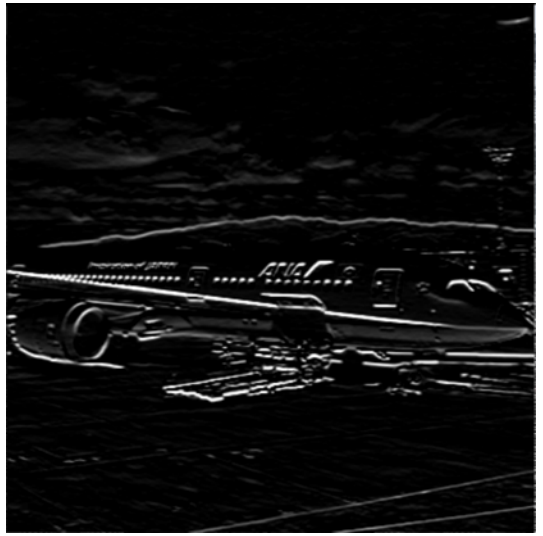
畳み込み演算



カーネル行列を変えることで、1枚の画像から様々な特徴を持つ画像を生成できる。

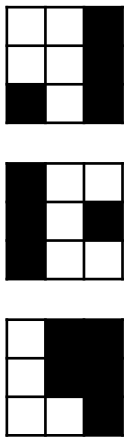
畳み込み演算

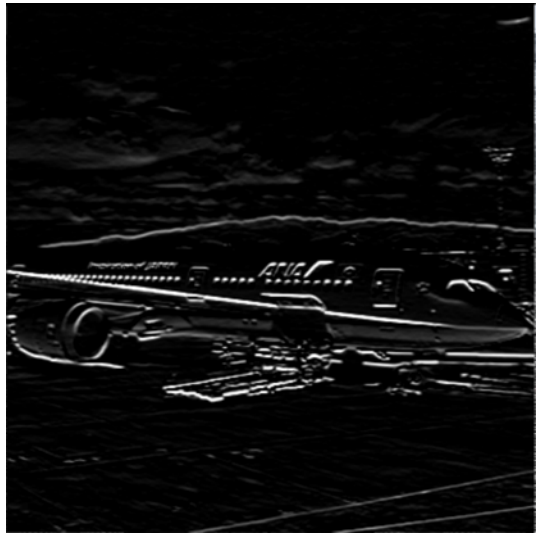




カーネル行列を変えることで、1枚の画像から様々な特徴を持つ画像を生成できる。

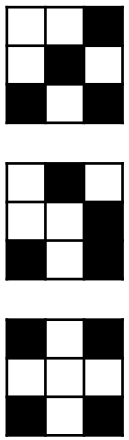
畳み込み演算

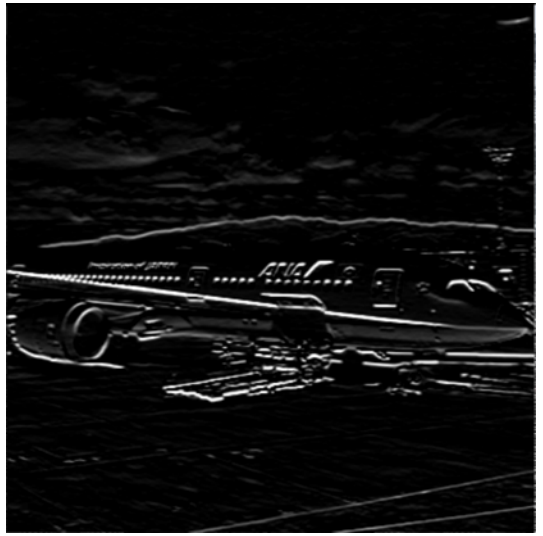




カーネル行列を変えることで、1枚の画像から様々な特徴を持つ画像を生成できる。

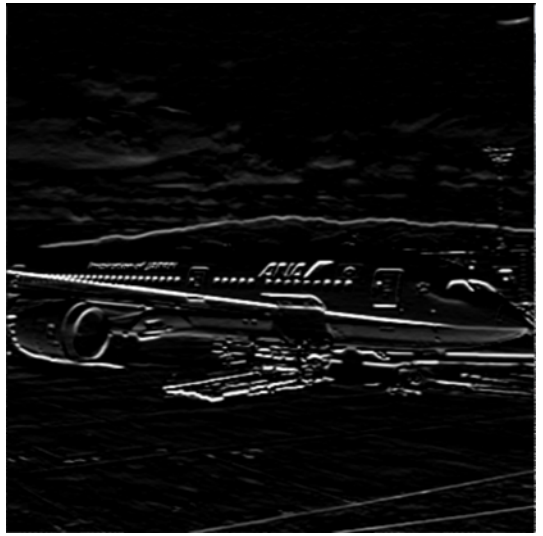
畳み込み演算





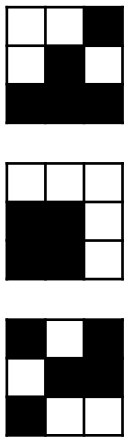
カーネル行列を変えることで、1枚の画像から様々な特徴を持つ画像を生成できる。

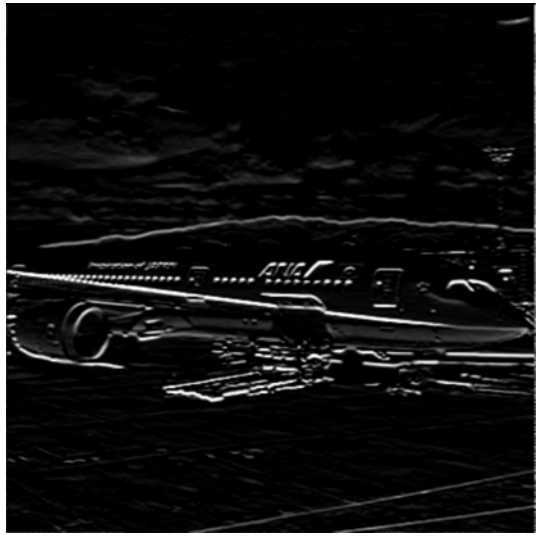
畳み込み演算



カーネル行列を変えることで、1枚の画像から様々な特徴を持つ画像を生成できる。

畳み込み演算

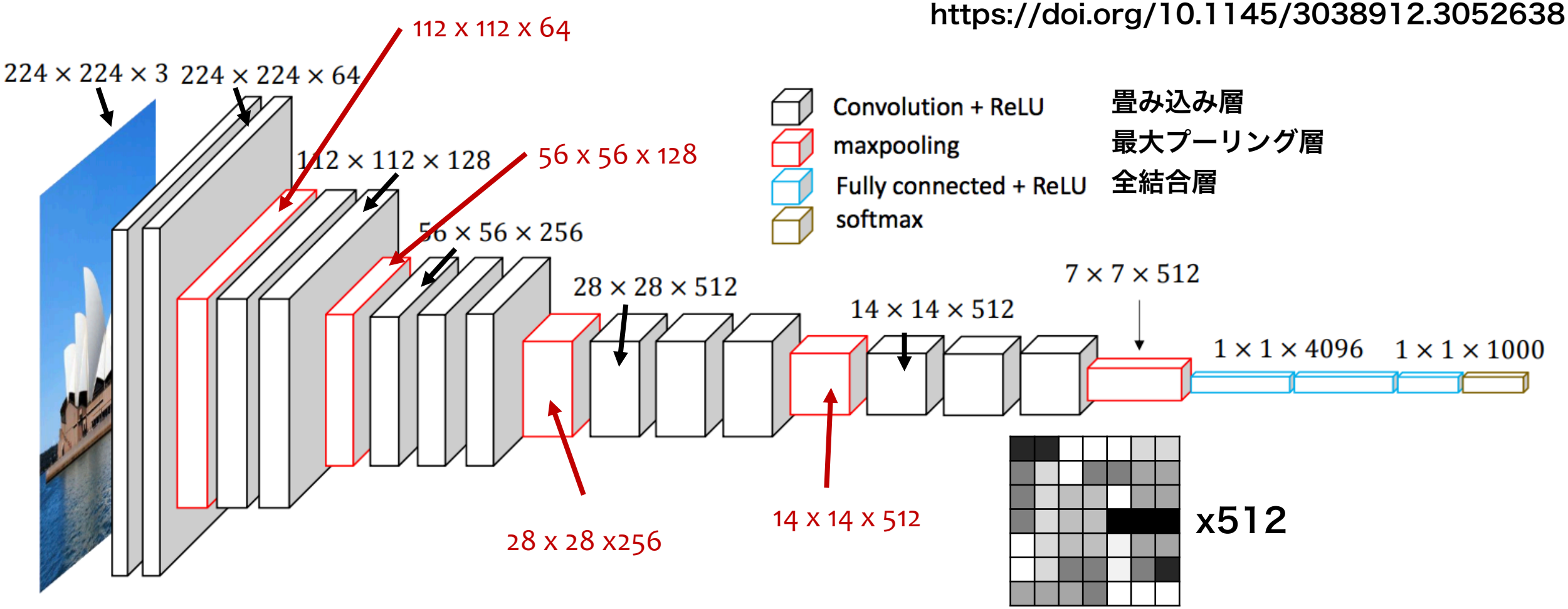




カーネル行列を変えることで、1枚の画像から様々な特徴を持つ画像を生成できる。

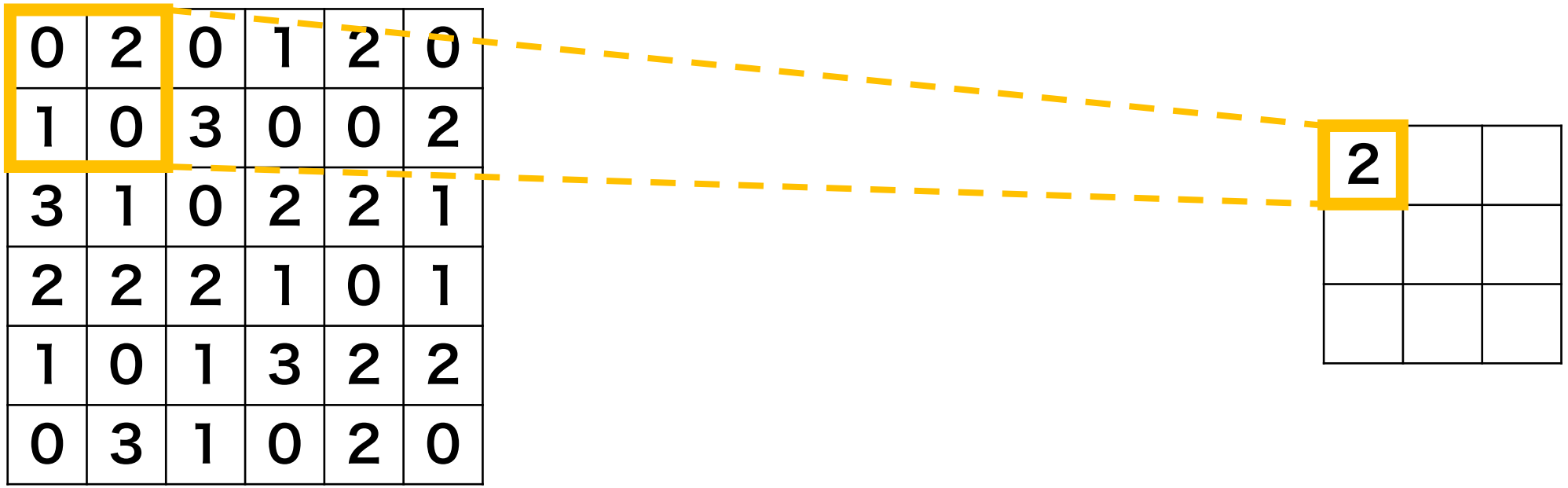
畳み込みニューラルネットワーク

https://doi.org/10.1145/3038912.3052638

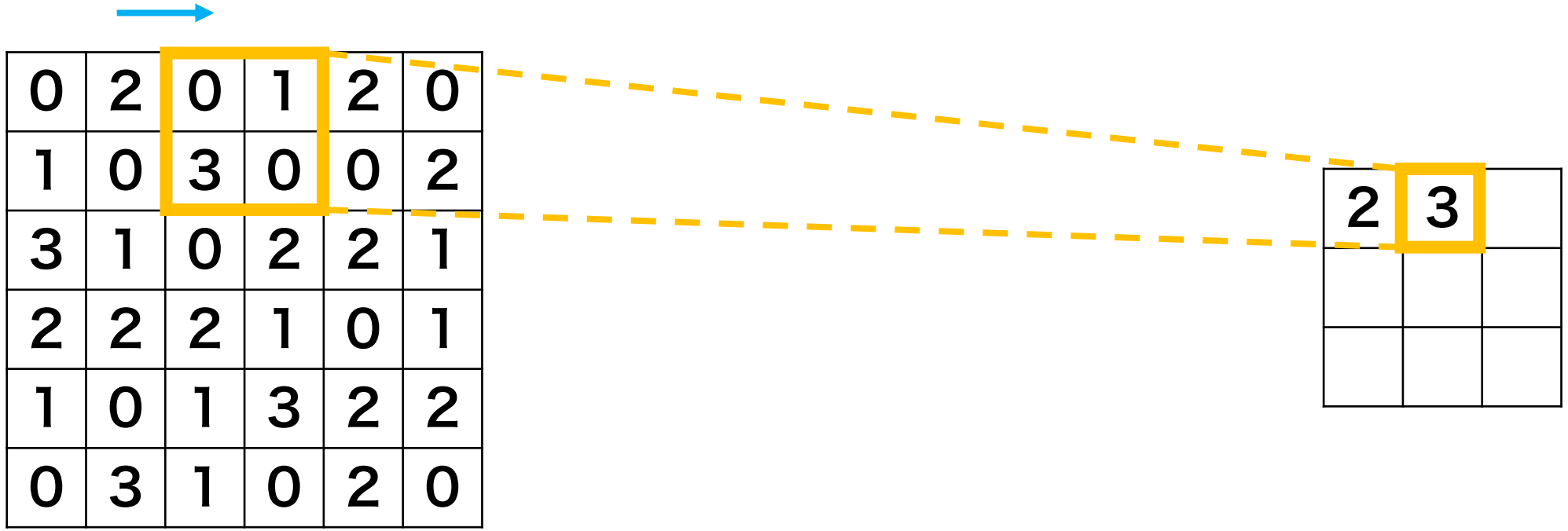


プーリング演算

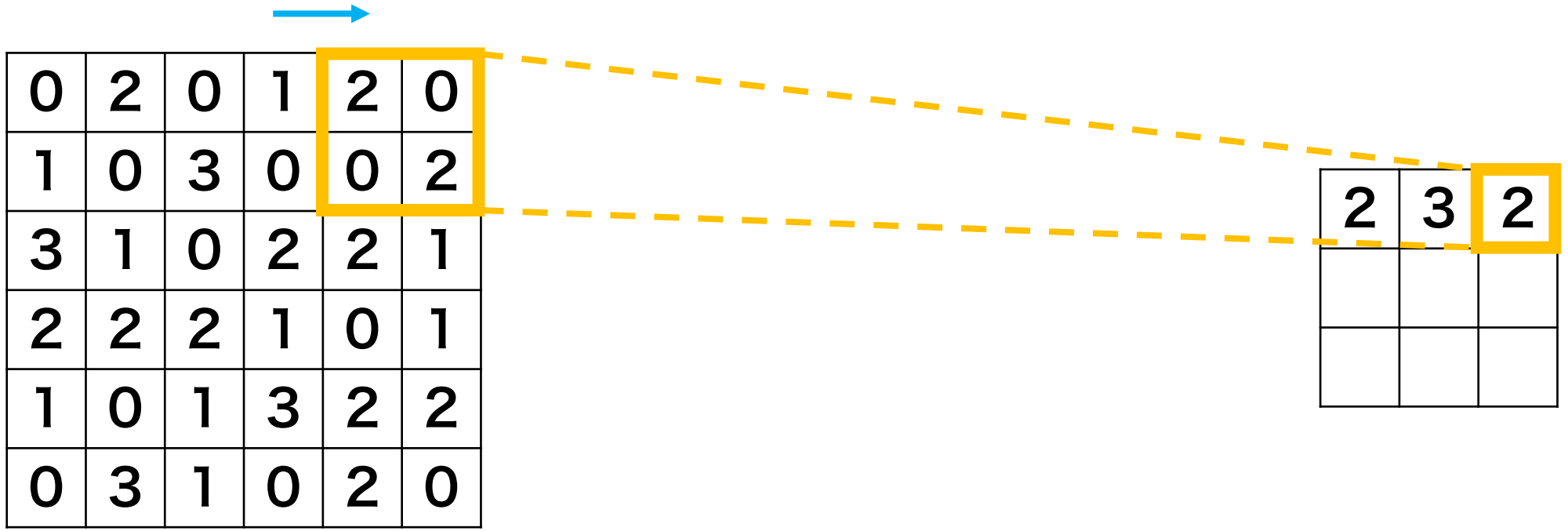
画像の局所的な部分が少し変化しても、その最大値プーリング結果が変化しないため、最大値プーリングは入力データのノイズに対してより頑健な特徴量を生成するのに役立つ。



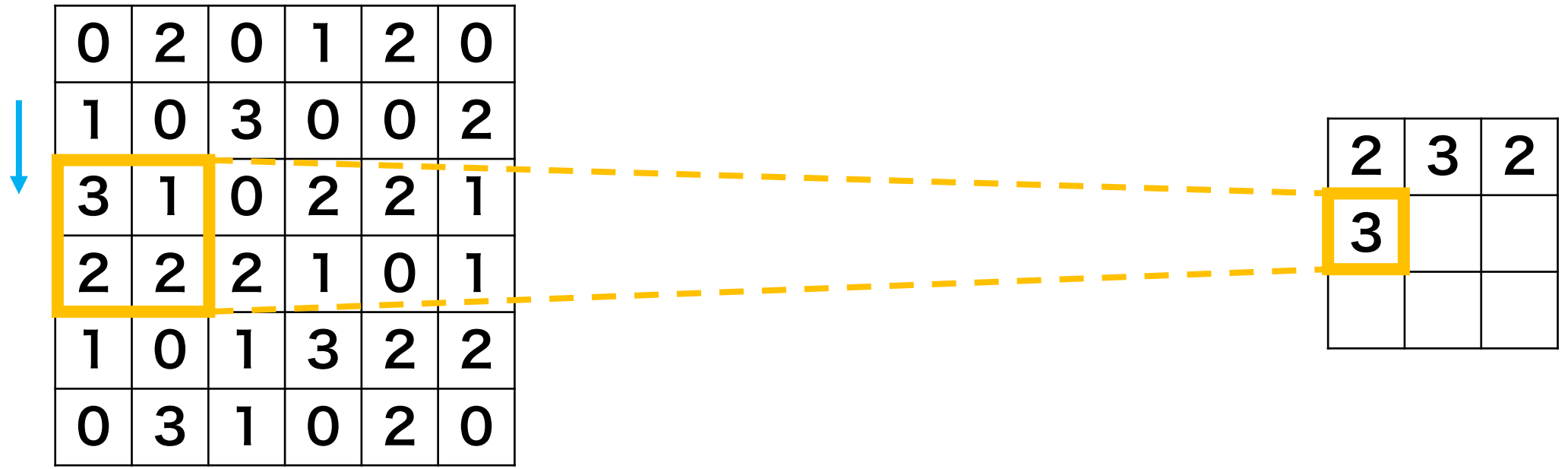
プーリング演算



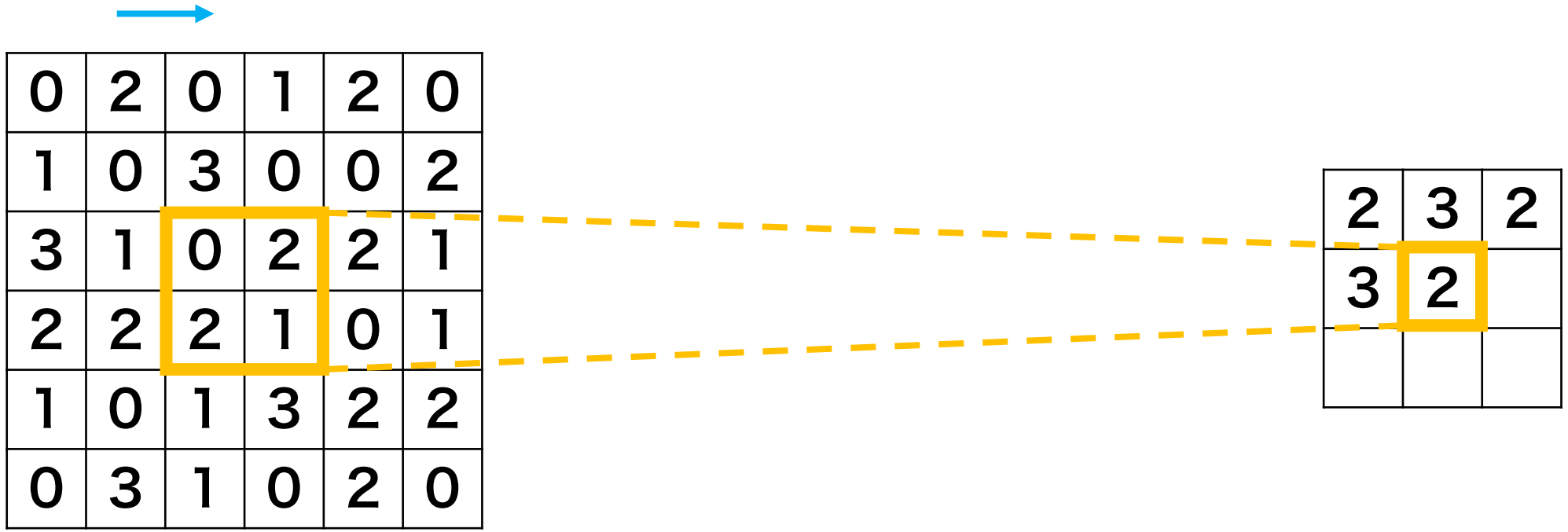
プーリング演算



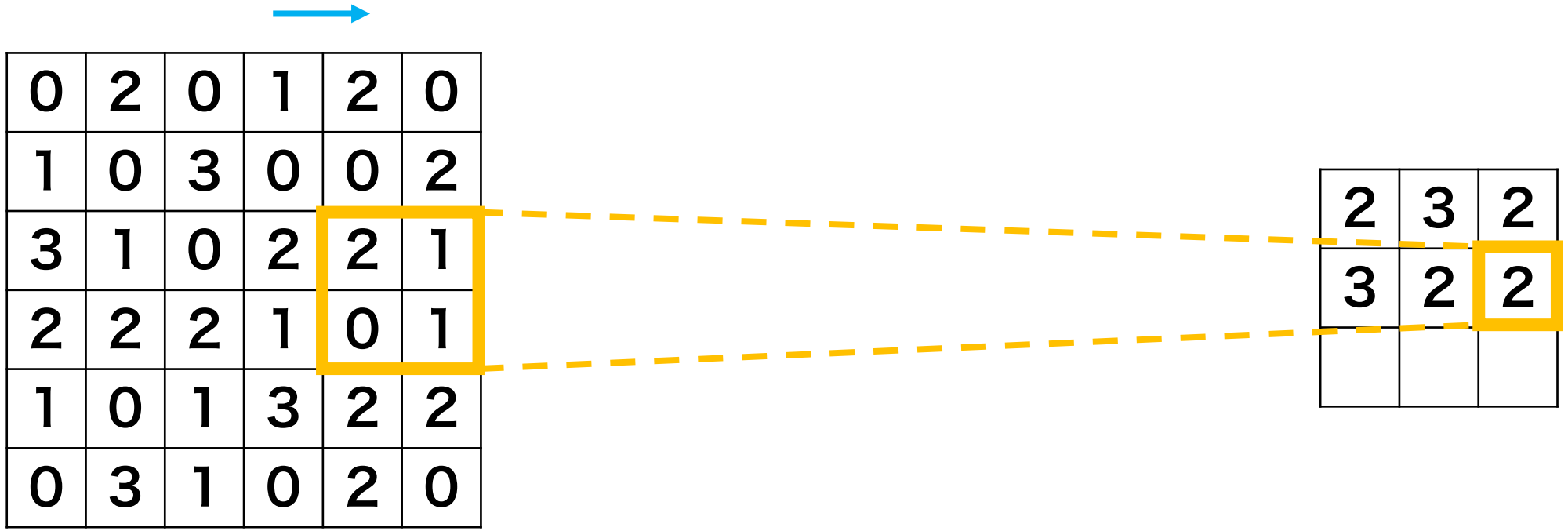
プーリング演算



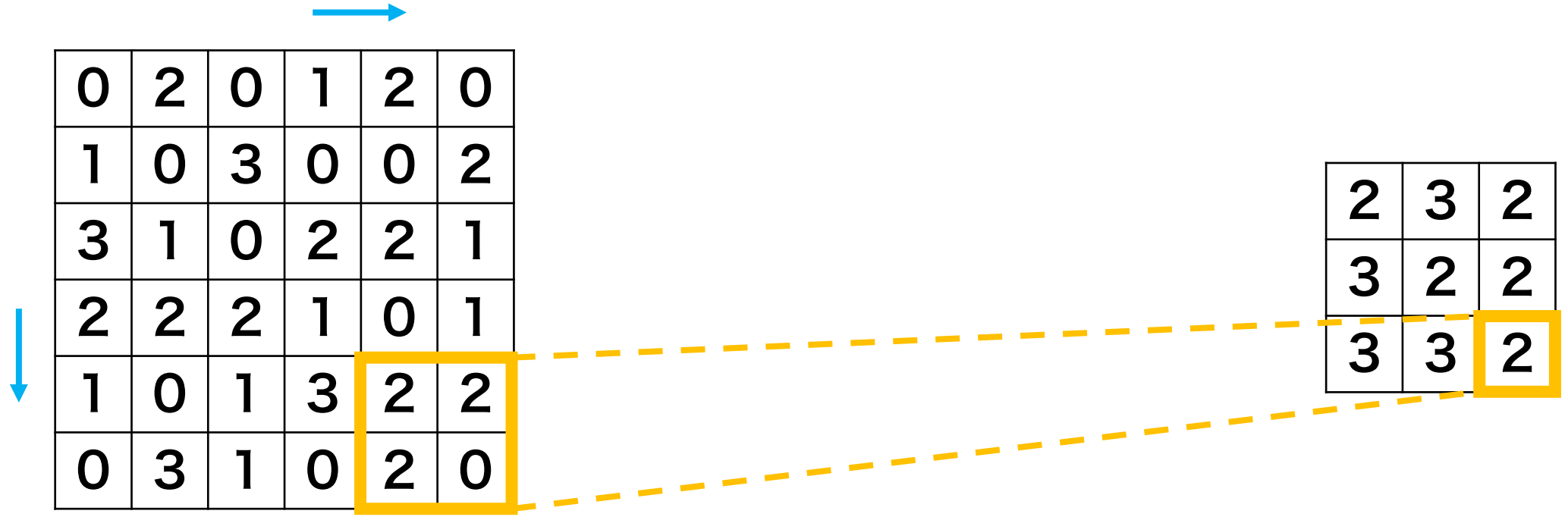
プーリング演算



プーリング演算

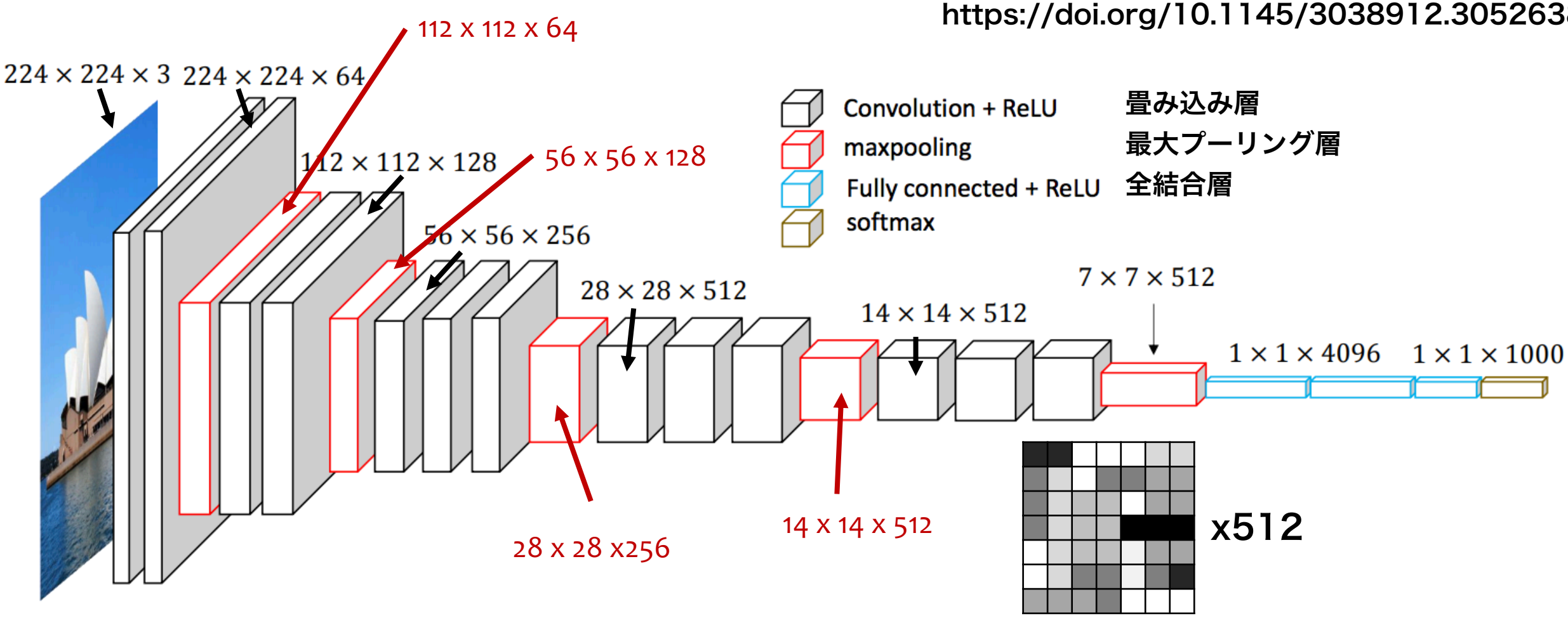


プーリング演算

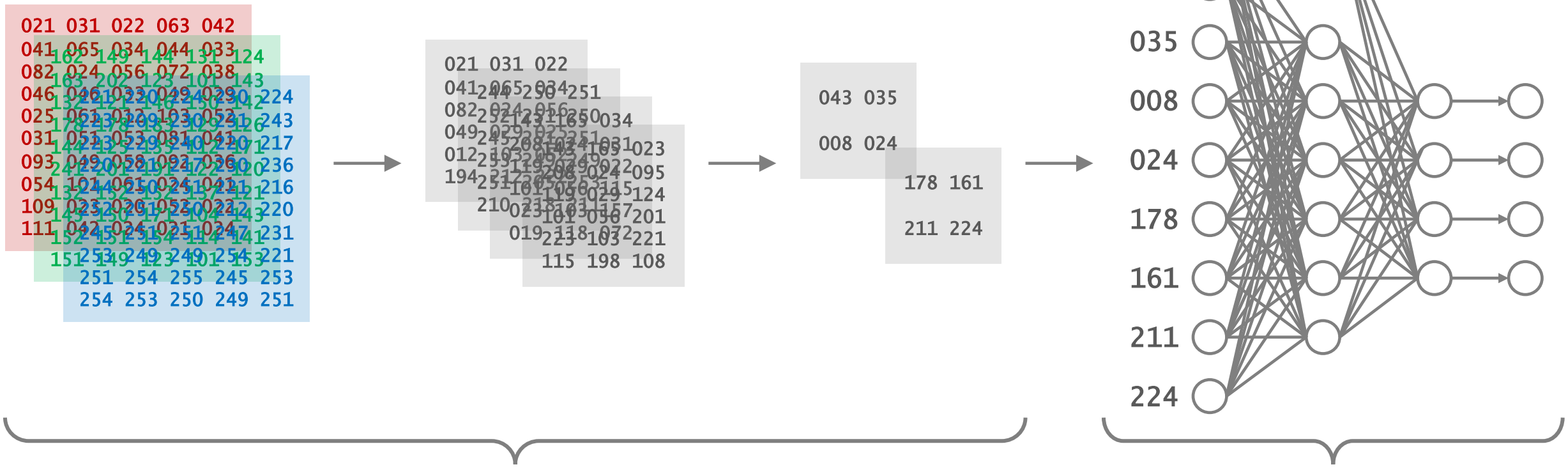


畳み込みニューラルネットワーク

https://doi.org/10.1145/3038912.3052638

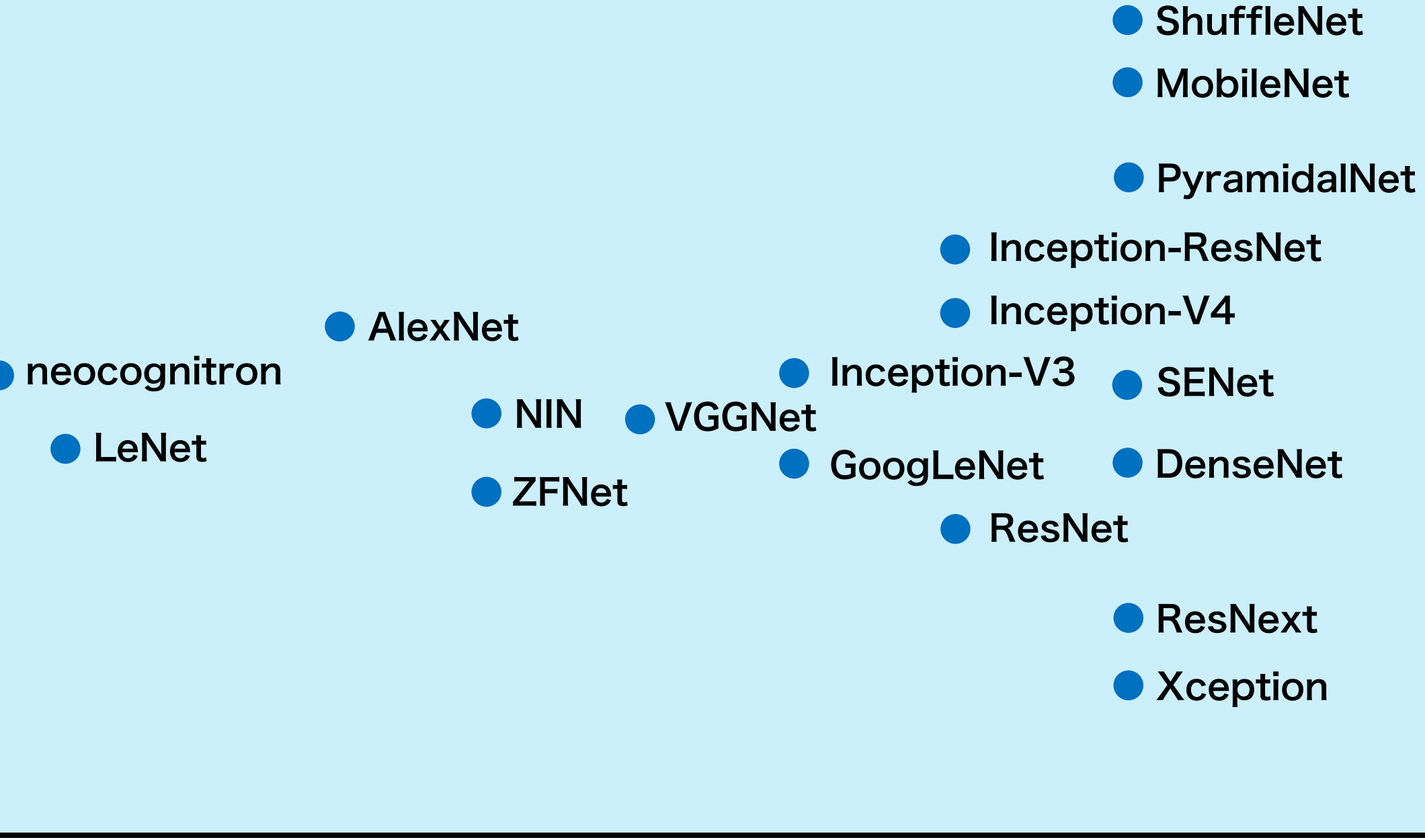


畳み込みニューラルネットワーク



畳み込み
（特徴抽出）

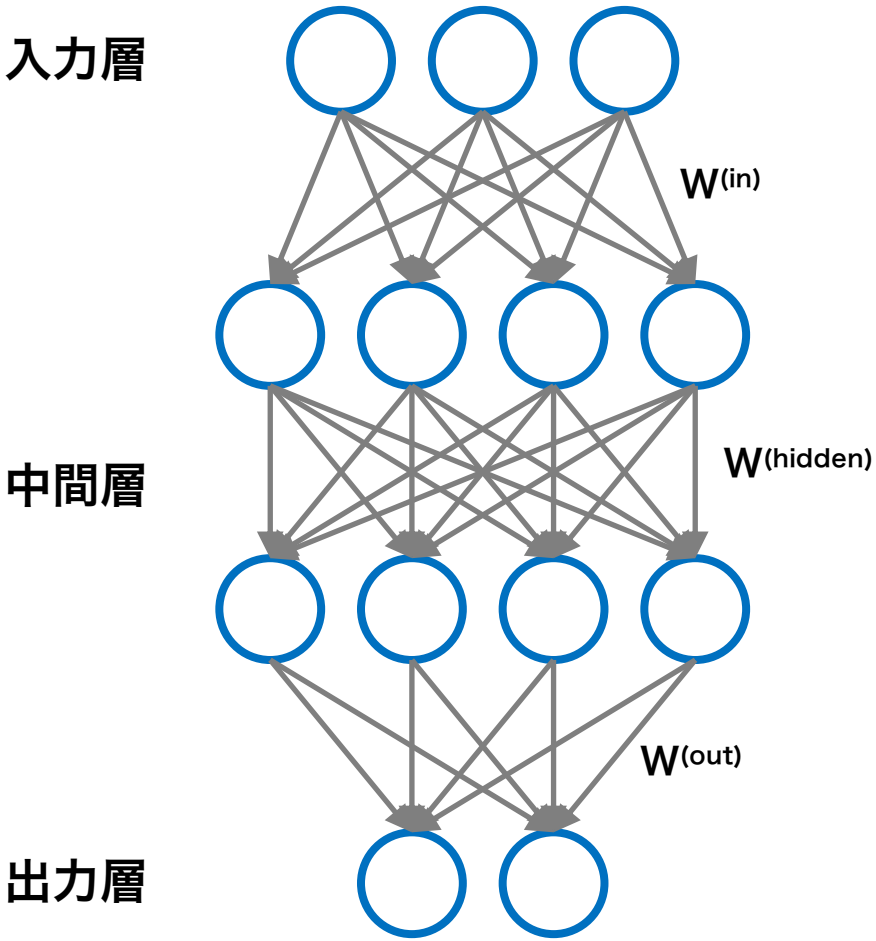
ニューラルネットワーク
（分類）


ShuffleNet
MobileNet
PyramidalNet
Inception-ResNet
Inception-V4
AlexNet
neocognitron
Inception-V3
SENet
NIN
VGGNet
LeNet
GoogLeNet
DenseNet
ZFNet
ResNet
ResNext
Xception
1998
2012
2014
2016
2018
2020

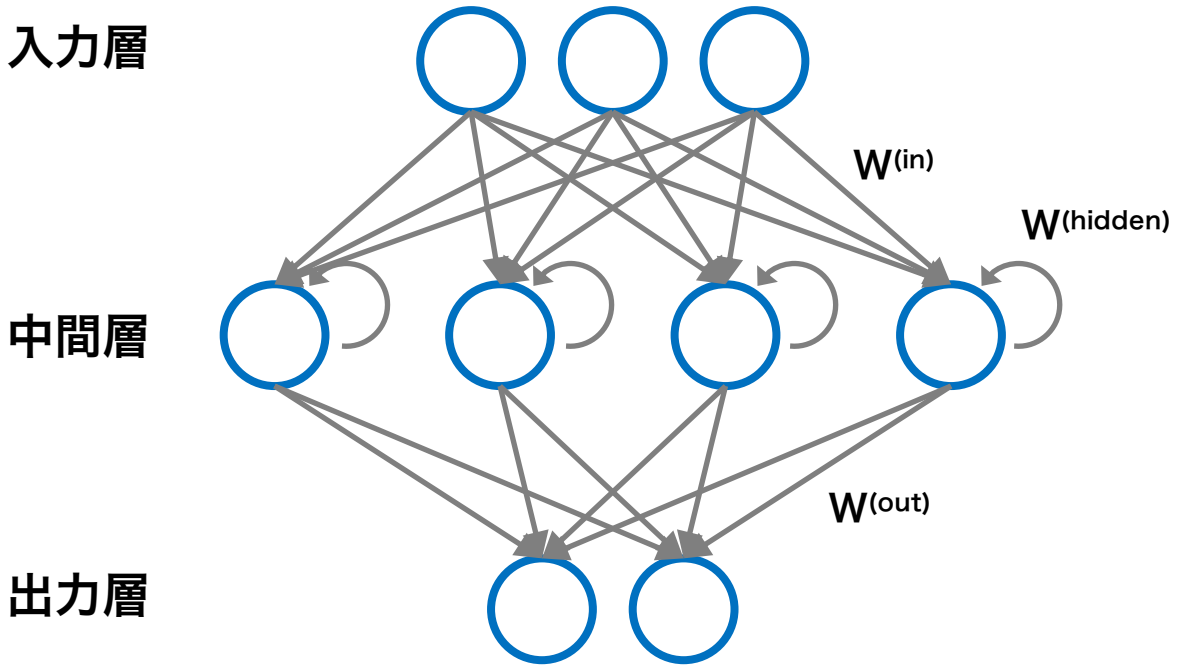
ニューラルネットワーク

- パーセプトロン
- ニューラルネットワーク
- CNN
- RNN

深層ニューラルネットワーク



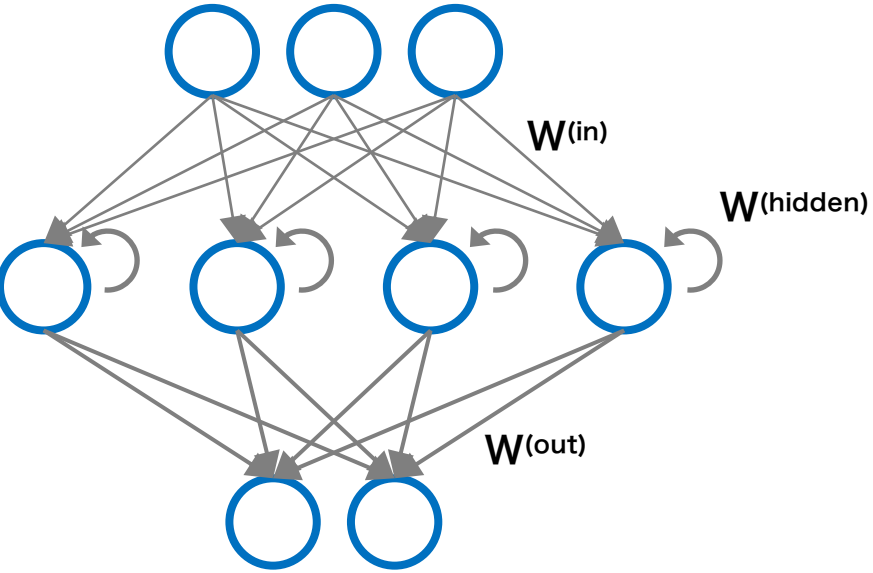
再帰型ニューラルネットワーク



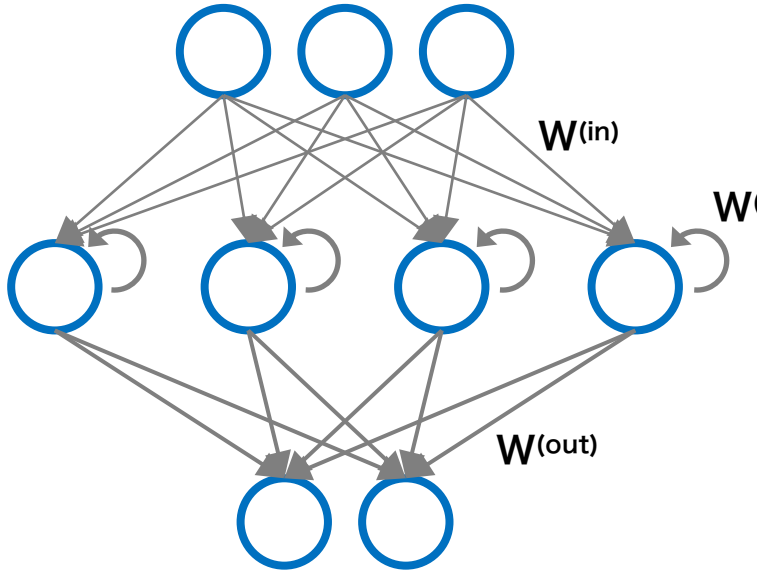
再帰型ニューラルネットワーク

RNN は状態（時間）を保持でき、そのネットワーク構造を時間軸方向に展開できる。RNN の中間層は、入力層から情報を受け取り、演算を行い、その結果を出力層に伝播するとともに、同じ情報を次の状態の自分自身（中間層）にも伝播する。これによって、状態 t の中間層は、状態 t – 1 の中間層の情報を保持できるようになる。

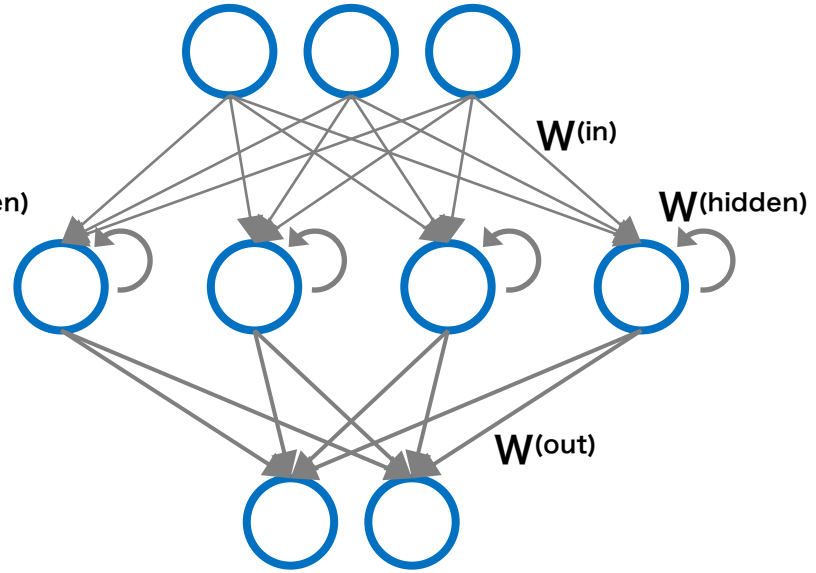
状態 t - 1



状態 t

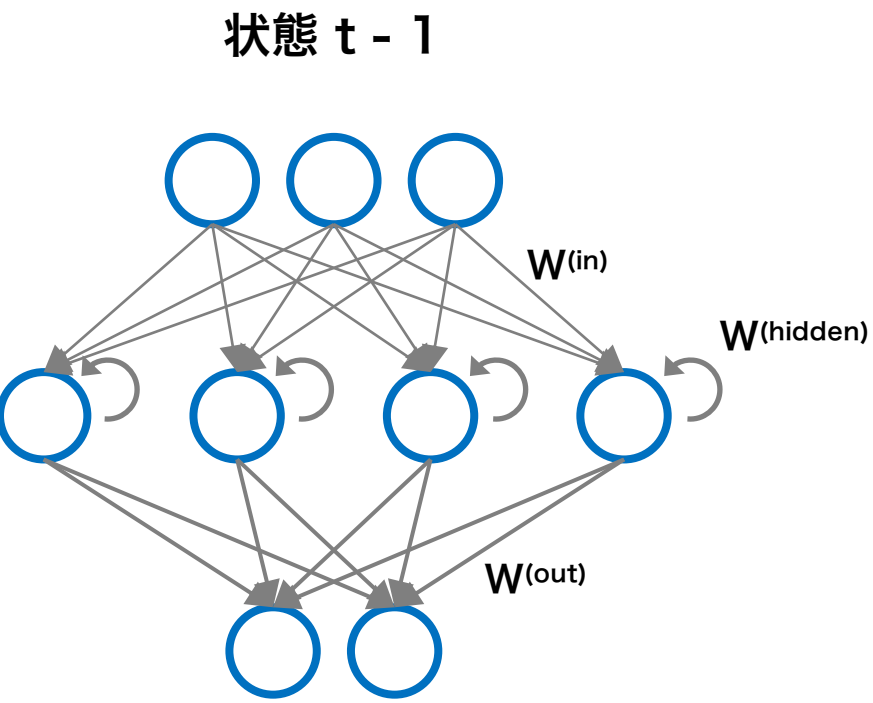


状態 t + 1



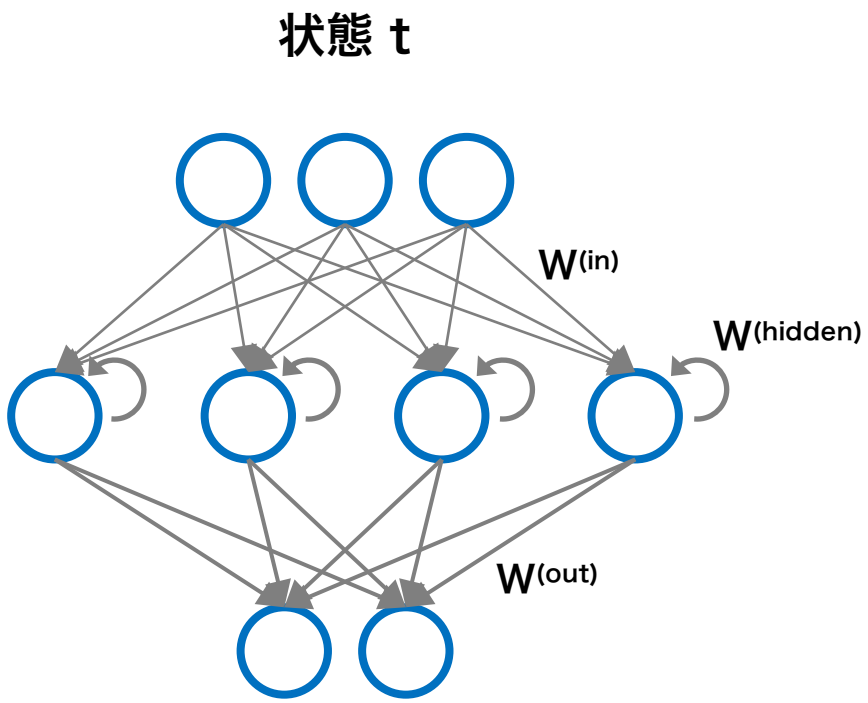
再帰型ニューラルネットワーク

RNN は状態（時間）を保持でき、そのネットワーク構造を時間軸方向に展開できる。RNN の中間層は、入力層から情報を受け取り、演算を行い、その結果を出力層に伝播するとともに、同じ情報を次の状態の自分自身（中間層）にも伝播する。これによって、状態 t の中間層は、状態 t – 1 の中間層の情報を保持できるようになる。



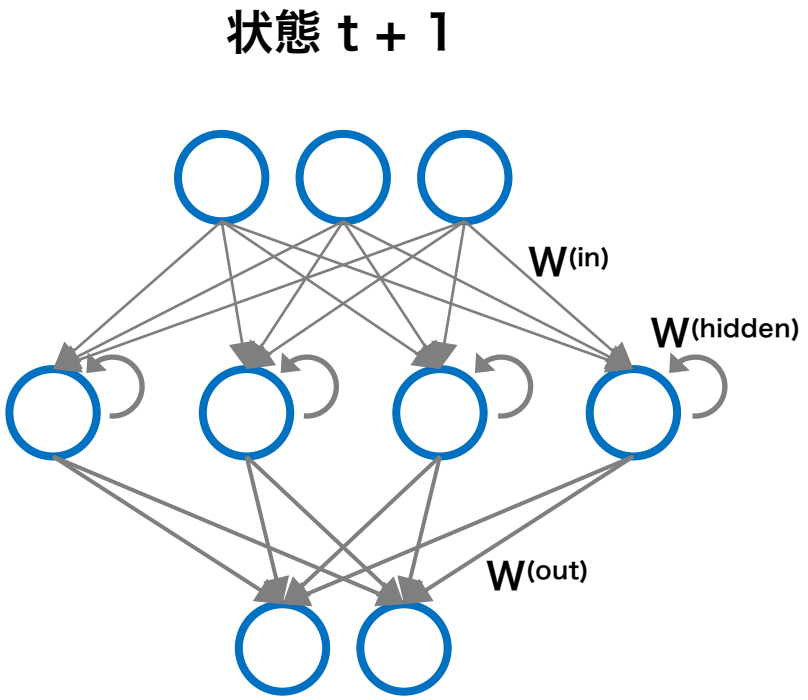
再帰型ニューラルネットワーク

RNN は状態（時間）を保持でき、そのネットワーク構造を時間軸方向に展開できる。RNN の中間層は、入力層から情報を受け取り、演算を行い、その結果を出力層に伝播するとともに、同じ情報を次の状態の自分自身（中間層）にも伝播する。これによって、状態 t の中間層は、状態 t – 1 の中間層の情報を保持できるようになる。



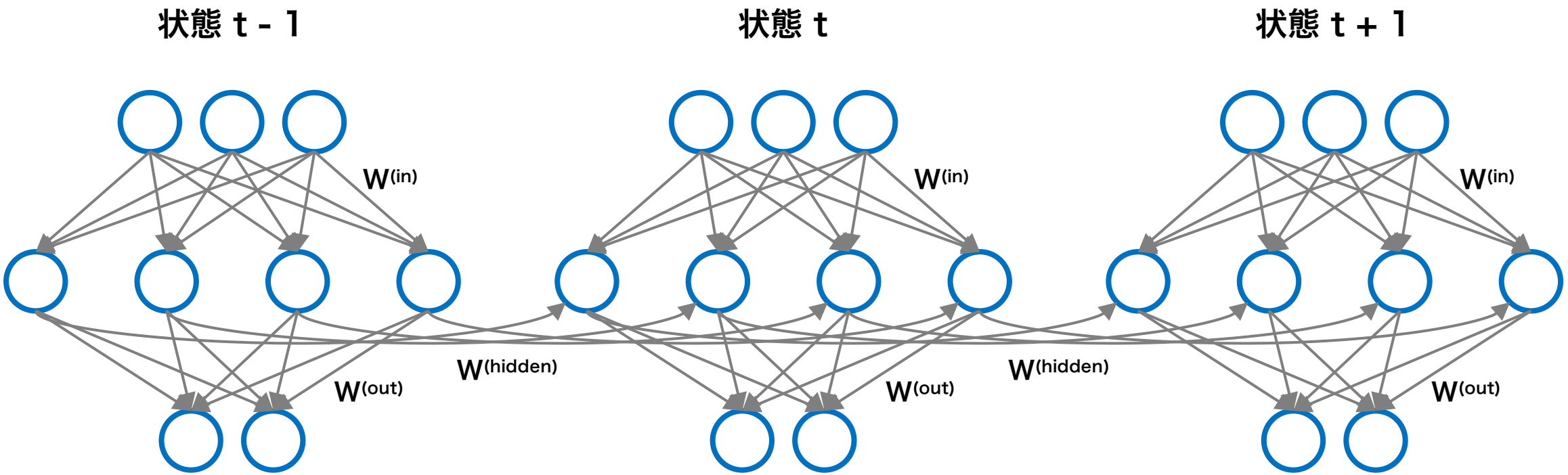
再帰型ニューラルネットワーク

RNN は状態（時間）を保持でき、そのネットワーク構造を時間軸方向に展開できる。RNN の中間層は、入力層から情報を受け取り、演算を行い、その結果を出力層に伝播するとともに、同じ情報を次の状態の自分自身（中間層）にも伝播する。これによって、状態 t の中間層は、状態 t – 1 の中間層の情報を保持できるようになる。



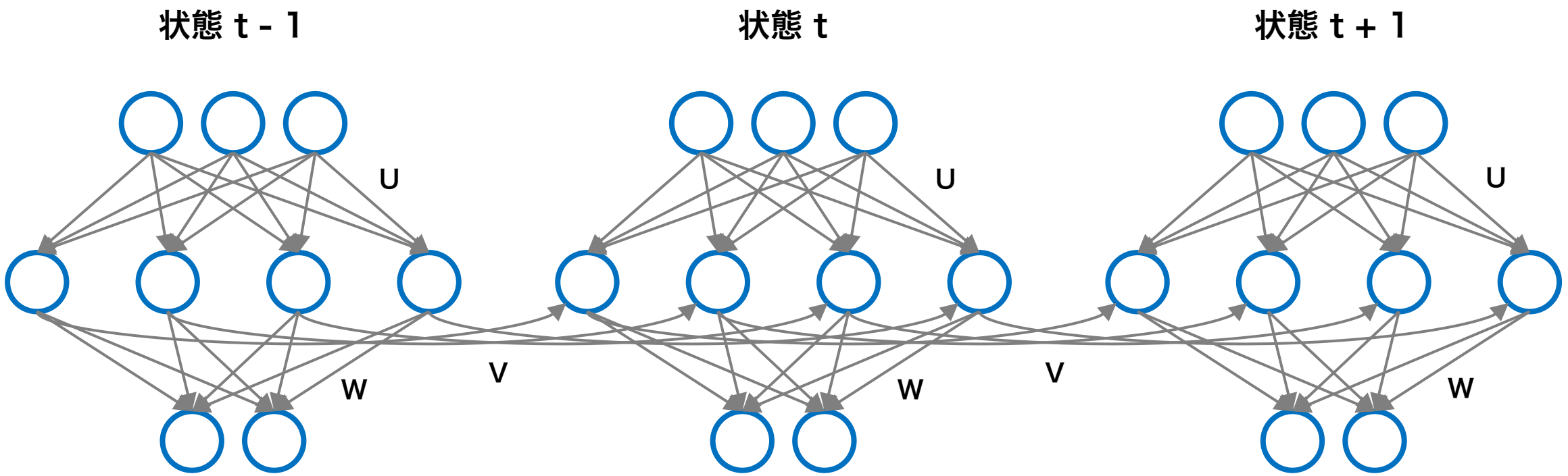
再帰型ニューラルネットワーク

RNN は状態（時間）を保持でき、そのネットワーク構造を時間軸方向に展開できる。RNN の中間層は、入力層から情報を受け取り、演算を行い、その結果を出力層に伝播するとともに、同じ情報を次の状態の自分自身（中間層）にも伝播する。これによって、状態 t の中間層は、状態 t – 1 の中間層の情報を保持できるようになる。



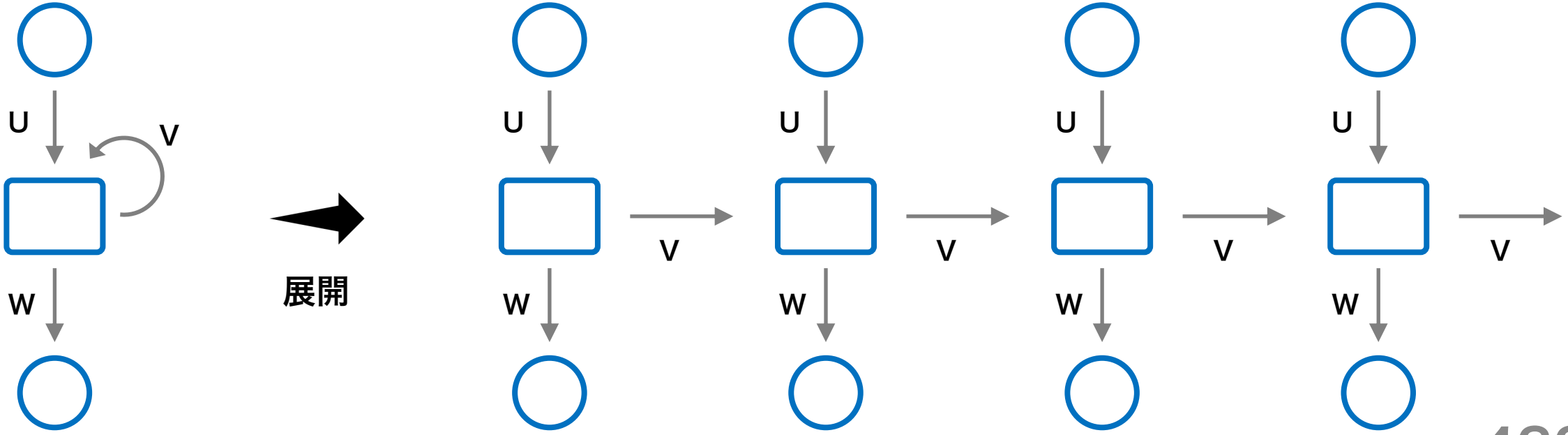
再帰型ニューラルネットワーク

RNN は状態（時間）を保持でき、そのネットワーク構造を時間軸方向に展開できる。RNN の中間層は、入力層から情報を受け取り、演算を行い、その結果を出力層に伝播するとともに、同じ情報を次の状態の自分自身（中間層）にも伝播する。これによって、状態 t の中間層は、状態 t – 1 の中間層の情報を保持できるようになる。



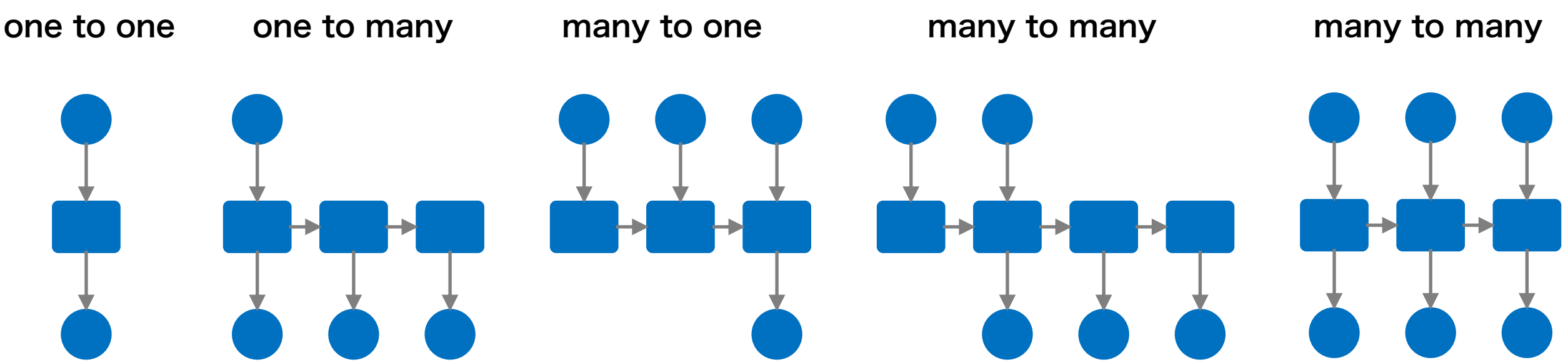
再帰型ニューラルネットワーク

RNN は状態（時間）を保持でき、そのネットワーク構造を時間軸方向に展開できる。RNN の中間層は、入力層から情報を受け取り、演算を行い、その結果を出力層に伝播するとともに、同じ情報を次の状態の自分自身（中間層）にも伝播する。これによって、状態 t の中間層は、状態 t – 1 の中間層の情報を保持できるようになる。



再帰型ニューラルネットワーク

RNN は状態（時間）を保持でき、そのネットワーク構造を時間軸方向に展開できる。RNN の中間層は、入力層から情報を受け取り、演算を行い、その結果を出力層に伝播するとともに、同じ情報を次の状態の自分自身（中間層）にも伝播する。これによって、状態 t の中間層は、状態 t – 1 の中間層の情報を保持できるようになる。

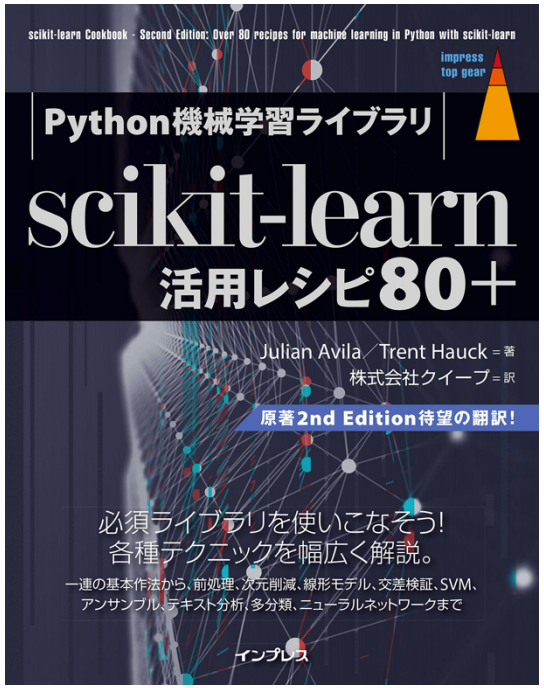


http://karpathy.github.io/2015/05/21/rnn-effectiveness/

参考図書

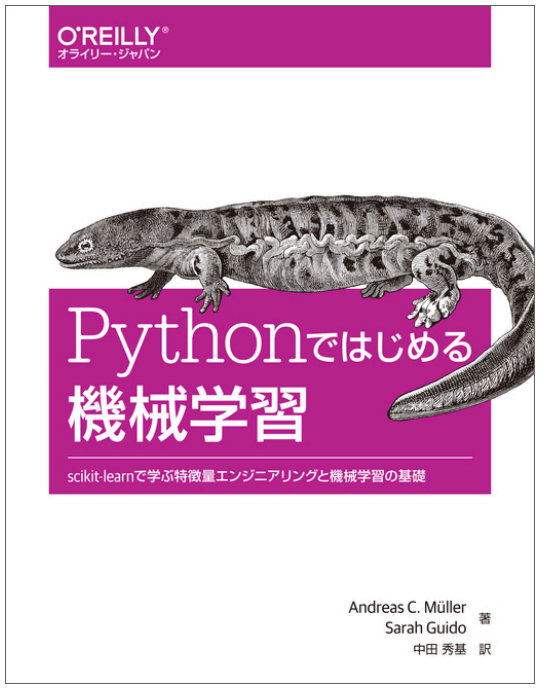
Python 機械学習プログラミング – 達人データサイエンティストによる理論と実践 –

インプレス



Python 機械学習ライブラリ scikit-learn 活用レシピ 80+

インプレス



Python ではじめる機械学習 – scikit-learn で学ぶ特徴量エンジニアリングと機械学習の基礎 –

オライリー・ジャパン

機械学習のための特徴量エンジニアリング – その原理と Python による実践 –

オライリー・ジャパン