Python ではじめる機械学習

1.7 最初のアプリケーション
アイリスのクラス分類

15T4057F　藤井　真

１．最初のアプリケーション
アイリスのクラス分類

- はじめに
  - アイリス(iris:アヤメ:鳶尾花)

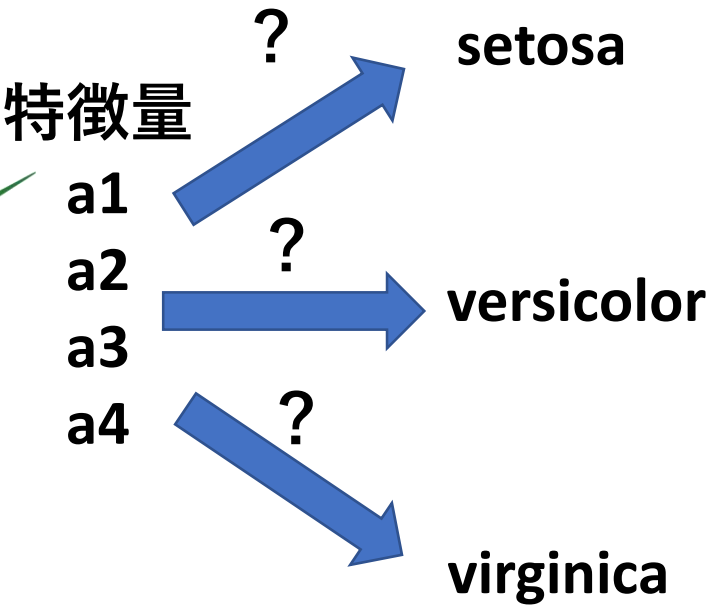
- クラス（3種類）
  - setosa
  - versicolor
  - virginica

クラス分類

- クラス（3種類）
  - setosa
  - versicolor
  - virginica

- 特徴量(4部位）
  - ガクの長さ　sepal length
  - ガクの幅　sepal width
  - 花弁の長さ　petal length
  - 花弁の幅　petal width

アヤメX



irisデータセット

- 全150のデータポイント
  （アイリスデータ）がある

- 1データポイントにつき、
  4特徴量とクラスラベルがある

- python環境構築のAnacondaに含まれている。

1.7.1 データを読む

- irisデータセットを実際に確認する。

  ①データインポート

  ②データをpythonで確認する

①データインポート

```
from sklearn.datasets import load_iris #scikit-learnのdatasetsモジュールカ
iris_dataset = load_iris()              #load_iris関数をiris_datasetと変数

print(iris_dataset.keys())
```

```
dict_keys(['data', 'target', 'target_names', 'DESCR', 'feature_names'])
```

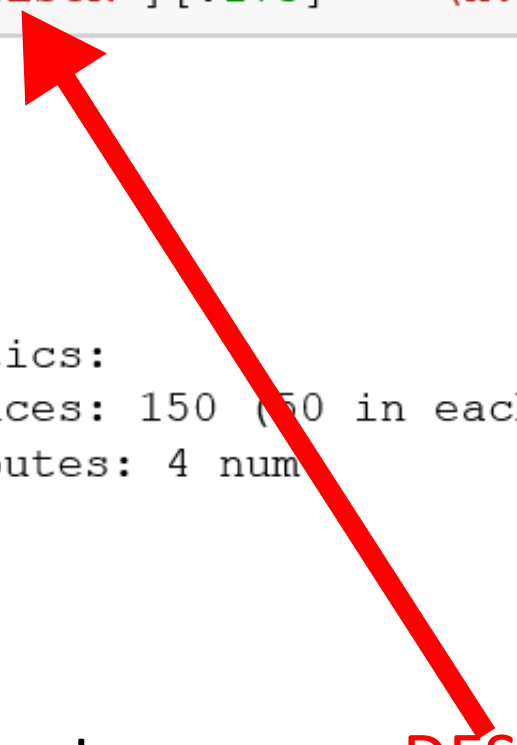
data　target　target_names　DESCR　feature_names

②データをpythonで確認する

```
print(iris_dataset['DESCR'][:173] + "\n...")
```

```
Iris Plants Database
====================

Notes
-----
Data Set Characteristics:
    :Number of Instances: 150 (50 in each of three classes)
    :Number of Attributes: 4 num
...
```



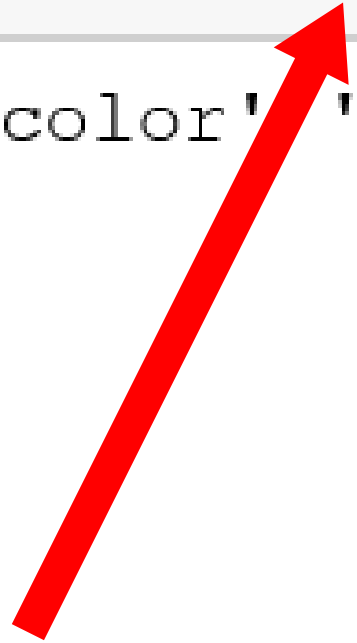
data　target　target_names　DESCR　feature_names

description：説明

②データをpythonで確認する

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

```
['setosa' 'versicolor' 'virginica']
```



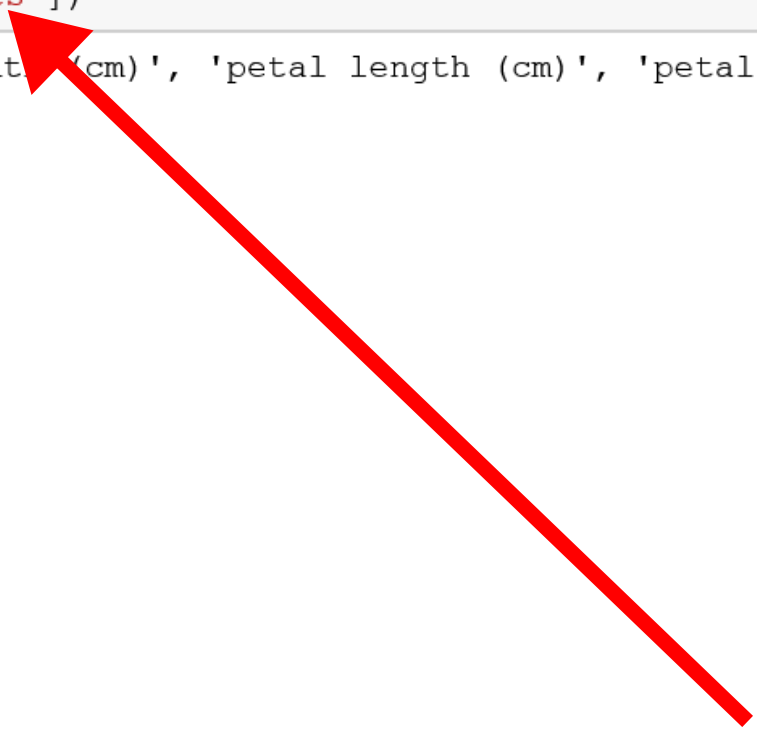
data　target　target_names　DESCR　feature_names

クラスの名称

②データをpythonで確認する

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

```
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
```



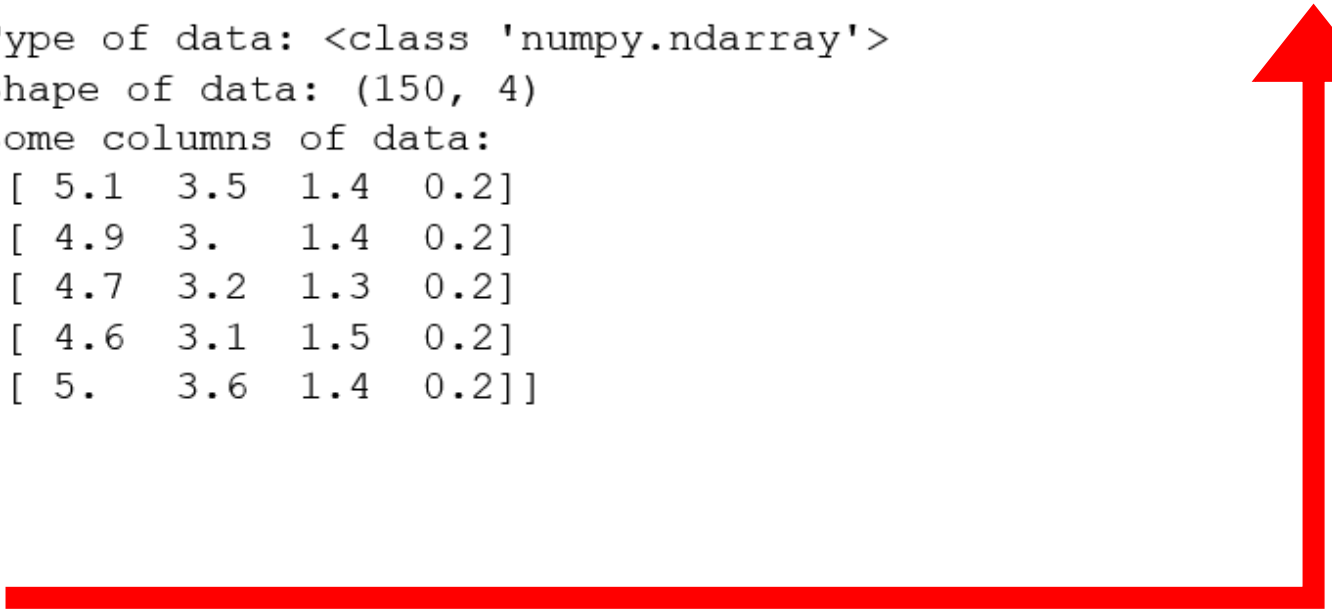
data　target　target_names　DESCR　feature_names

特徴量の名称

②データをpythonで確認する

```
print("Type of data: {}".format(type(iris_dataset['data'])))
print("Shape of data: {}".format(iris_dataset['data'].shape))
print("some columns of data:\n{}".format(iris_dataset['data'][:5]))
```

```
Type of data: <class 'numpy.ndarray'>
Shape of data: (150, 4)
some columns of data:
[[ 5.1  3.5  1.4  0.2]
 [ 4.9  3.   1.4  0.2]
 [ 4.7  3.2  1.3  0.2]
 [ 4.6  3.1  1.5  0.2]
 [ 5.   3.6  1.4  0.2]]
```



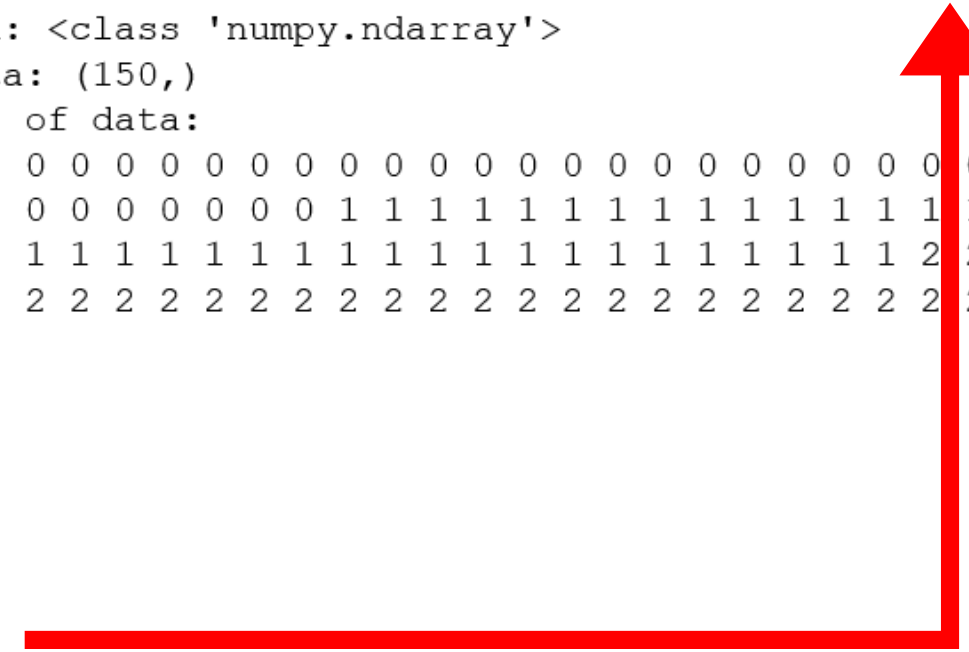
data　target　target_names　DESCR　feature_names

特徴量データ自体

②データをpythonで確認する

```
print("Type of data: {}".format(type(iris_dataset['target'])))
print("Shape of data: {}".format(iris_dataset['target'].shape))
print("some columns of data:\n{}".format(iris_dataset['target']))
```

```
Type of data: <class 'numpy.ndarray'>
Shape of data: (150,)
some columns of data:
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
```



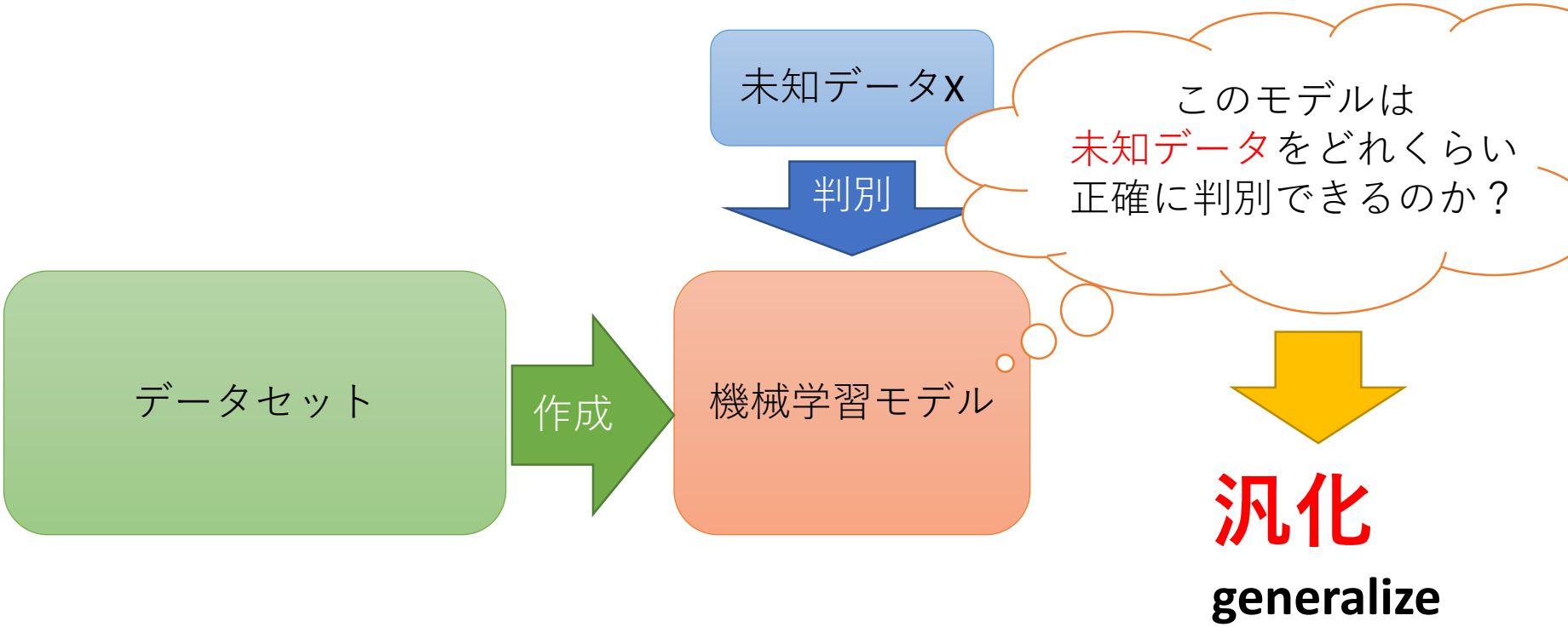
data　target　target_names　DESCR　feature_names

ラベル：

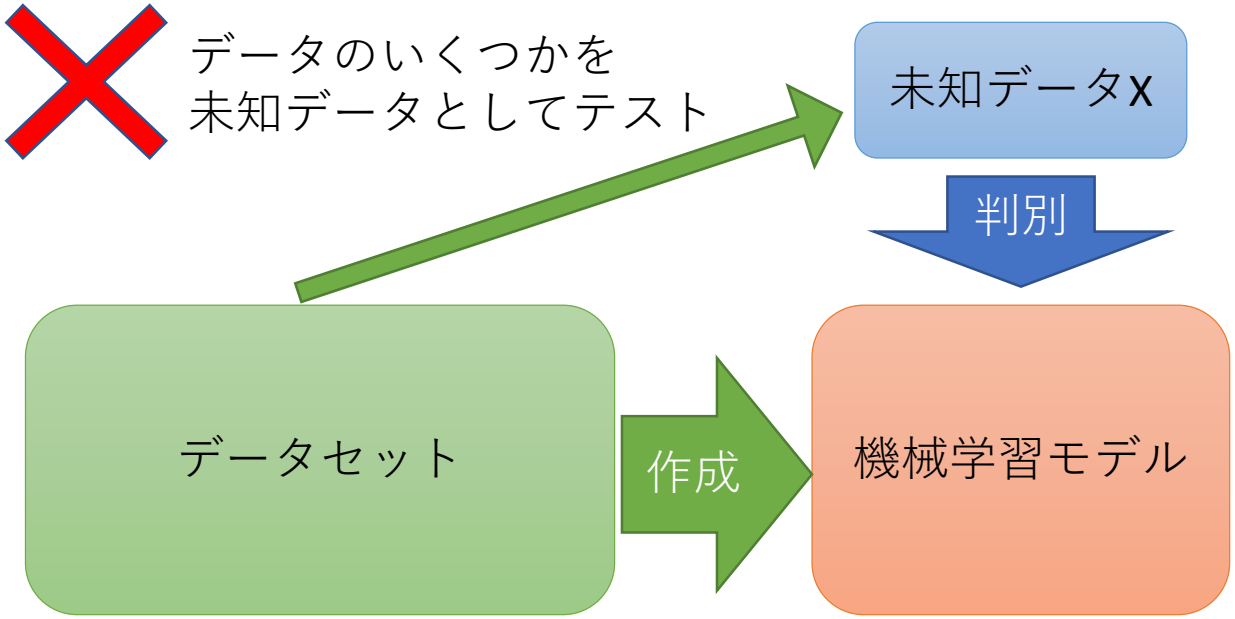
クラス分類の結果。ここでは

0:setosa　1:versicolor 2:virginica

1.7.2 成功度合いの測定 ：訓練データとテストデータ

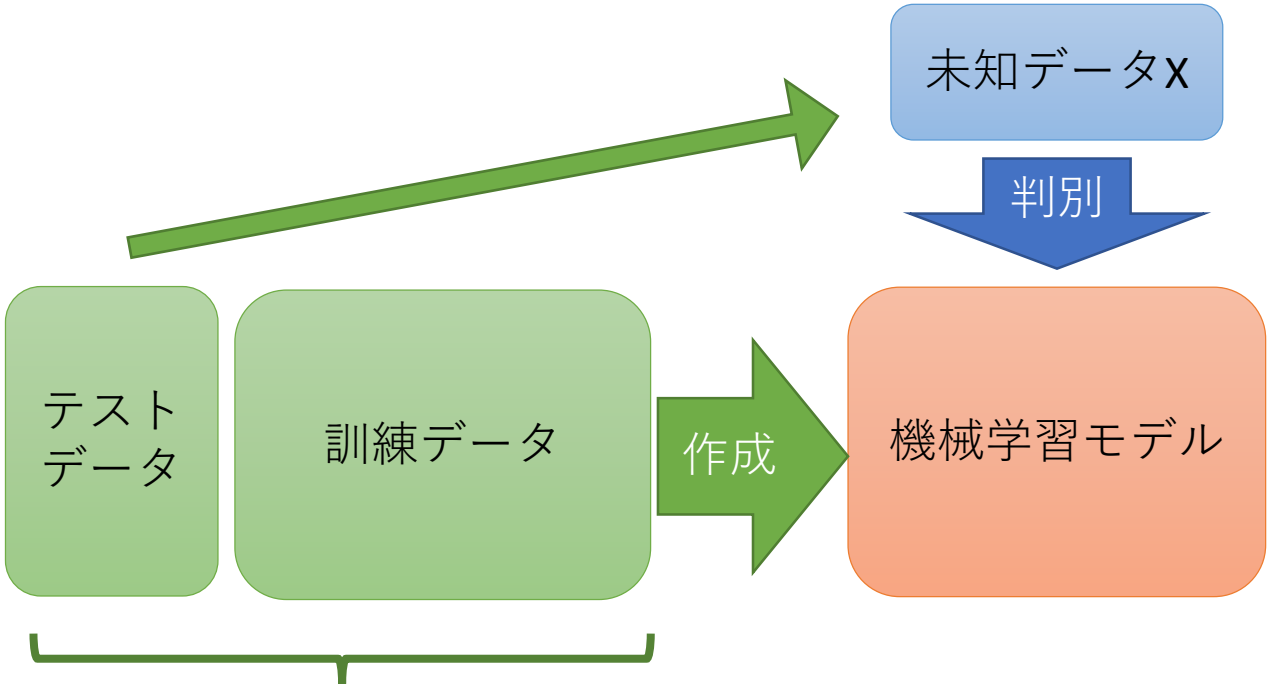


1.7.2 成功度合いの測定 ：訓練データとテストデータ



データセットのデータはモデル作成時に使われているため
「未知」のデータとしてふるまえない。

1.7.2 成功度合いの測定 ：訓練データとテストデータ

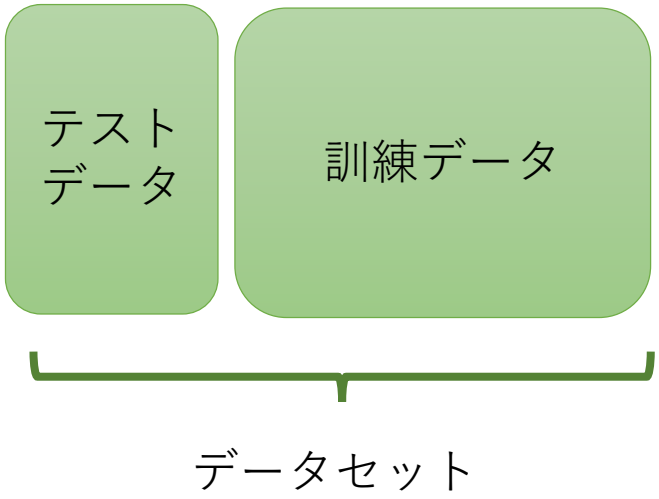


データセットをあらかじめ「テストデータ」と「訓練データ」に分けて利用すればいい。
テストデータはモデル作成時には使われないので未知データとしてふるまえる。

scikit-learnによる データセットの分割

- train_test_split関数

データセットを
テスト　25％
訓練　75％
に分割してくれる



train_test_split関数

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    iris_dataset['data'], iris_dataset['target'], random_state=0)
```

①インポート。

②関数の引数にデータセット、ラベルセット、ランダムを設定。

③引数を受け取る。以下、150のデータが3:1に分かれている。

```
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

```
(112, 4)
(112,)
(38, 4)
(38,)
```

1.7.3 最初にすべきこと
：データをよく観察する

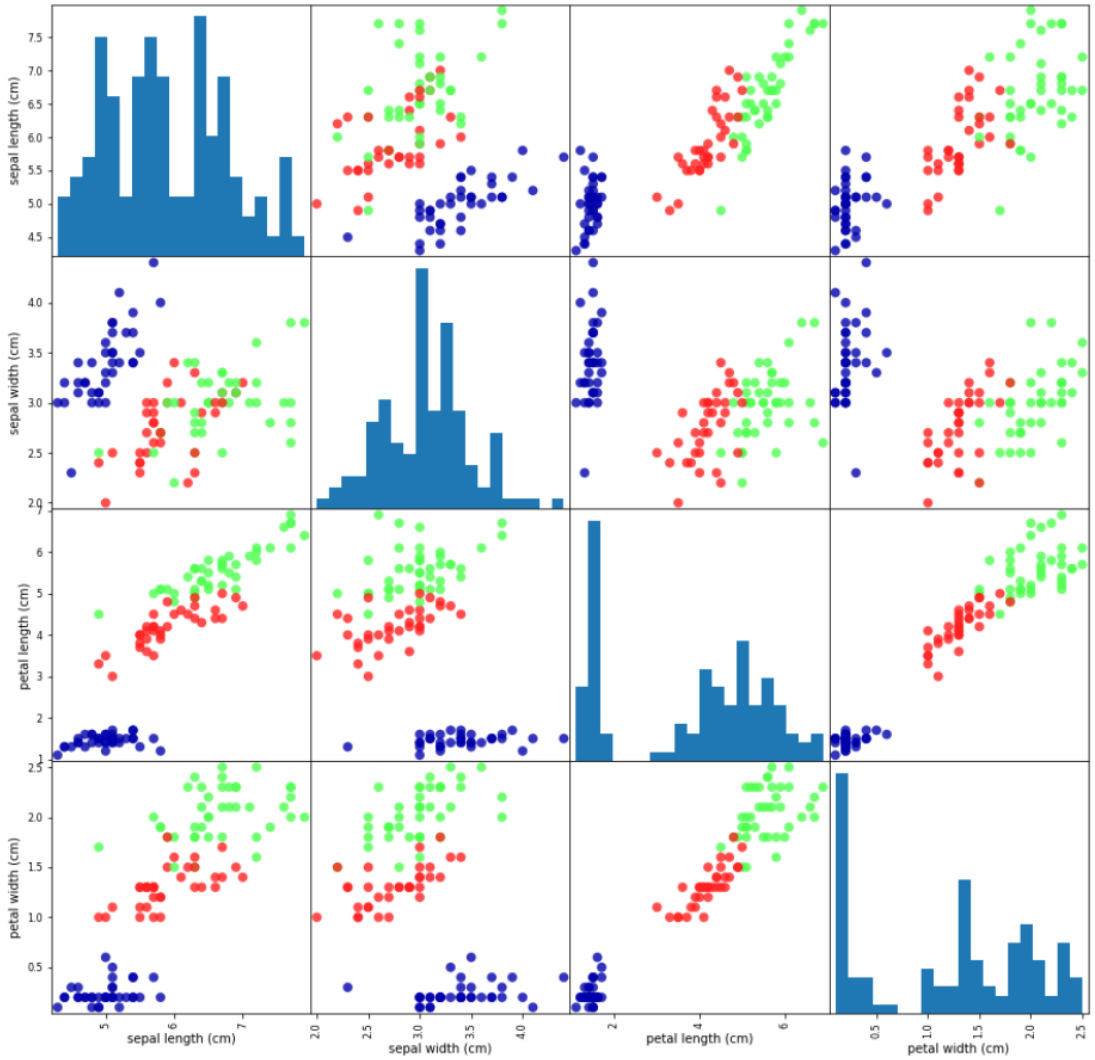
機械学習で最重要なこと
解決しようとする問題とデータを理解する

⇒今回はデータ理解の一例：可視化

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

iris_dataframe = pd.DataFrame(X_train, columns=iris_dataset.feature_names)

pd.plotting.scatter_matrix(
    iris_dataframe, c=y_train, figsize=(15, 15), marker='o',
    hist_kwds={'bins': 20}, s=60, alpha=.8, cmap=mglearn.cm3)
plt.show()
```



X_train
（訓練データ）
のクラス
（赤・青・緑）
はそれなりに
分かれている

1.7.4 最初のモデル
k-最近傍法

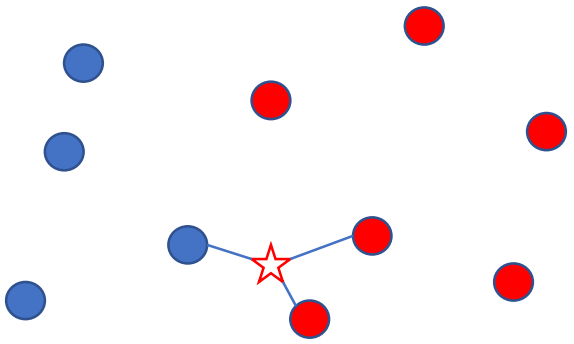
- はじめに

  k-最近傍法

  k-Nearest Neighbors : kNN

  k個の最も近い点を参照するモデル

  例：3NN



☆：新しいデータポイント

モデルを作成する

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=1)
```

①インポート。

②KNeighborsClassifier関数でknnモデルを作成する。今回のkは1。

```
knn.fit(X_train, y_train)
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=1, p=2,
           weights='uniform')
```

③先ほど作った1NNモデルのknnに
訓練データ(X_train,y_train)をfitメソッドで適用する。

1.7.5 予測を行う

```
import numpy as np
X_new = np.array([[5, 2.9, 1, 0.2]])
print("X_new.shape: {}".format(X_new.shape))
```

```
X_new.shape: (1, 4)
```

①numpyインポート。Scikit-learnはnumpy配列を前提としている。

②チェック用の1データX_newをnumpy配列で作成する

```
prediction = knn.predict(X_new)
print(prediction)
print(iris_dataset['target_names'][prediction])
```

```
[0]
['setosa']
```

③X_newを作成済みのknnモデルのpredictメソッドに入れる。

④結果をprediction変数で受けているので表示する。

1.7.6 モデルの評価

```
y_pred = knn.predict(X_test)
print(y_pred)
```

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

```
print("Test set score: {:.2f}".format(np.mean(y_pred == y_test)))
```

```
Test set score: 0.97
```

```
print("Test set score: {:.2f}".format(knn.score(X_test, y_test)))
```

```
Test set score: 0.97
```

同様に先ほど分けておいたテストデータについても予想する。
この結果の評価方法は

（上）numpy配列の値を単に比べて平均を出す。
（下）knnモデルのscoreメソッドを用いる

評価の結果はどちらも同じ。
knnのk=1のようなパラメータを調整（チューニング）して
評価を繰り返すと結果が向上することがある。