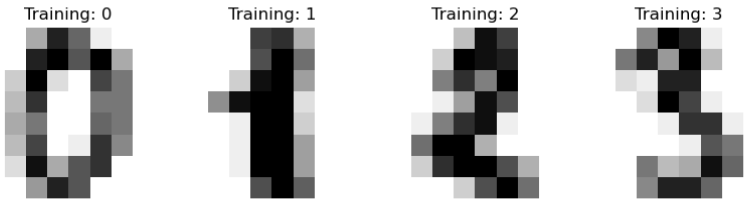
Digits データ

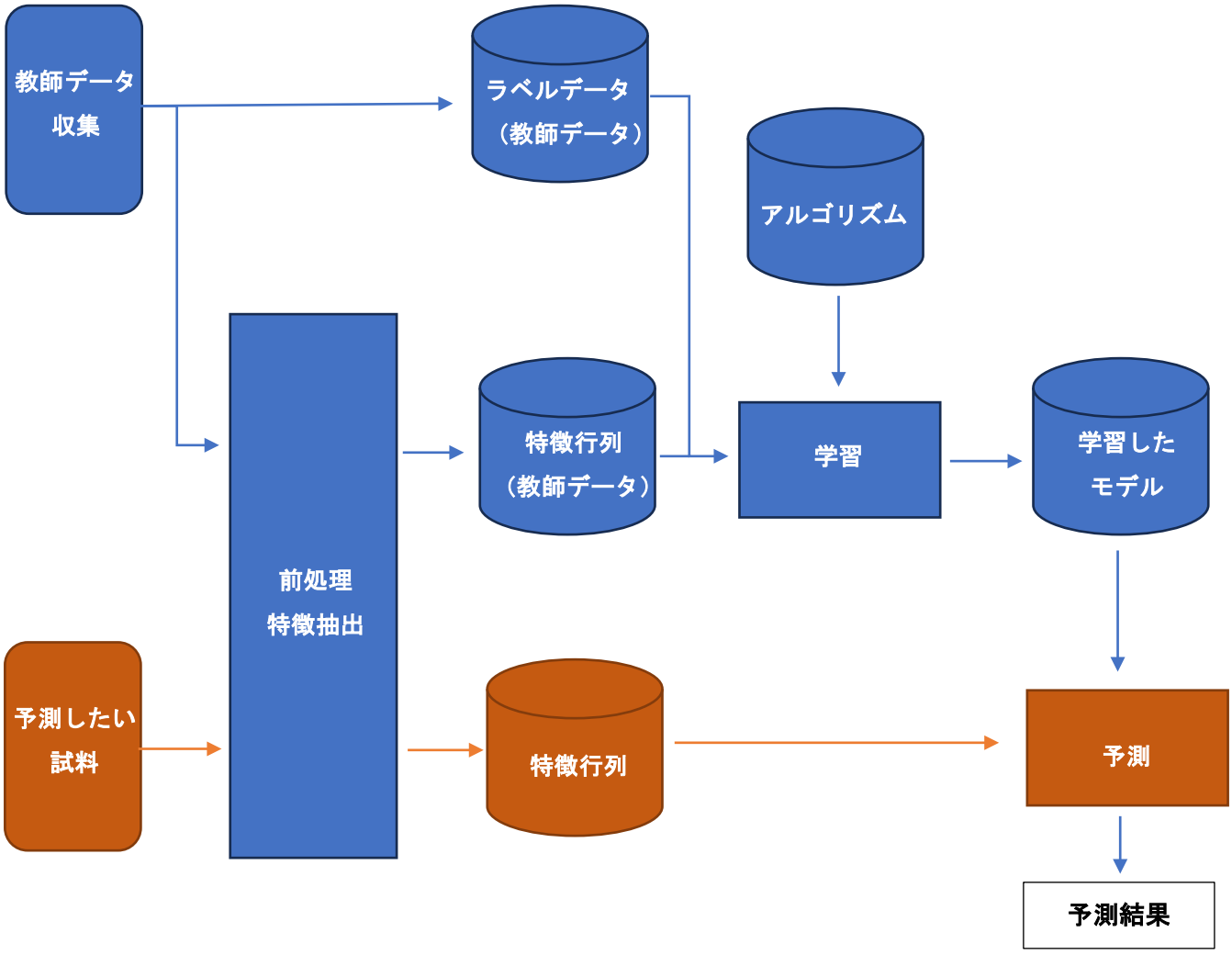
iris データは、単に「数値」を扱った基本的な学習モデルでした。機械学習には、数値だけではなく、イメージを用いた学習もできます。

scikit-learn には、手書きの数字の「digits」というデータセットが用意されています。このセットを使って文字認識の教師あり学習を見ていきます。

データは、0 から 9 までの手書きの数字で、1 つのデータは「8 ドット×8 ドット（=8 ビット）」、17 段階のグレースケールです。



■認識のプロセス



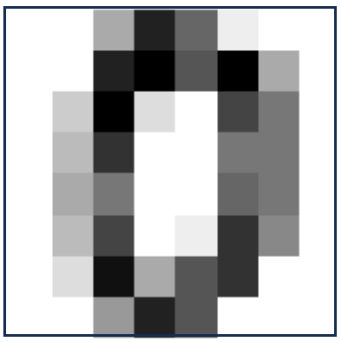
◆教師データの収集

手書きのデータに対応する「**ラベル付け**」（数値）をする。

画像	*0*	*1*	*2*	・・・	*9*
ラベル	0	1	2	・・・	9

◆前処理

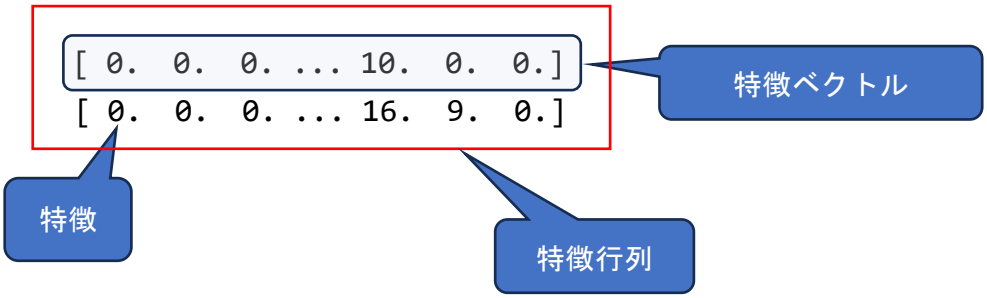
画像データから背景色、文字色、影など不必要な情報を取り除く「**前処理**」が行われる。この作業には `pillow` が使われ、グレースケールに落とされる。



各ピクセルの濃さを数値に置き換え、（ベクトル）行列を作る

`[0, 0, 5, 15, 7, 1, 0, 0, 0, ・・・・0, 0, 0]`

画像的な処理をされたものを「**特徴ベクトル**」とし、全体で「**特徴行列**」をつくる。



◆分類をおこなう教師あり学習

手書きの文字認識は教師あり学習です。

ラベルと教師データから抽出した「特徴」とアルゴリズムを使って学習をする。手書き文字認識は「分類」[1]を行う教師あり学習。

ここで用いられる「分類」法には、「**ロジスティック回帰**」と「**ランダムフォレスト**」が使われる。

---

[1] 教師あり学習には「分類」と「回帰」がある

■データセットの構造

scikit-learn に含まれている手書きデータは、カルフォルニア大学アーバイン校（UCI）で作成されたものです。そこから「UCI の手書きデータセット」と呼ばれています。

この UCI のデータセットは、2 つのデータから構成されている。1 つ目は「**特徴行列**」で、2 つ目は「**ラベルデータ**」。

「特徴ベクトル」は、手書きデータを 8×8 のますに対応させ、17 階調のグレースケールに置き換え、さらにそれを 64 個の一列のデータに並べている。

「ラベルデータ」は、1797 件分の画像が表わす数（正解）が入っている。

これらのデータセットは Numpy の ndarray というデータ型（shape）で格納されている。
特徴行列は行と列を持つので 2 次元の ndarray であり、特徴ベクトルは数字の並びなので 1 次元の ndarray になる。慣習的に、2 次元の ndarray の変数は大文字（X、Y など）で表わす。

◆digits データをロードする

まずは、iris 同様にデータを読み込みます。
データの読み込みには、sklearn.datasets モジュールの load_digits 関数を使います。

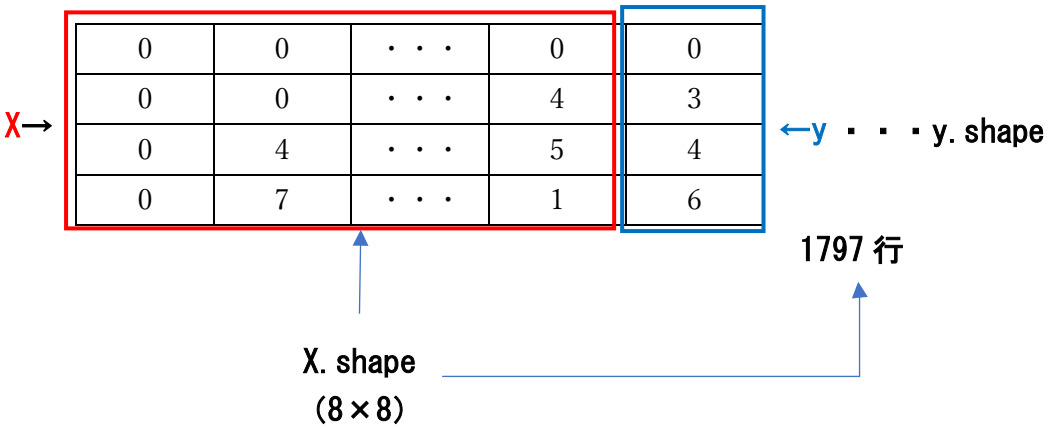
```
#1
from sklearn.datasets import load_digits
digits = load_digits()

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

```
[[ 0.  0.  5. ...  0.  0.  0.]
 [ 0.  0.  0. ... 10.  0.  0.]
 [ 0.  0.  0. ... 16.  9.  0.]
 ...
 [ 0.  0.  1. ...  6.  0.  0.]
 [ 0.  0.  2. ... 12.  0.  0.]
 [ 0.  0. 10. ... 12.  1.  0.]]
(1797, 64)
[0 1 2 ... 8 9 8]
[0 1 2 3 4 5 6 7 8 9]
```

ここでは、1797 個のデータが用意されていることが分かります。データは 8×8＝64 個の値で構成されています。

教師データである target には 0 から 9 の整数値で解答が指定されている。解答の名前は「target_name」として同じく 0 から 9 の整数値が指定されているのがわかる。

◆変数の表わすもの



◆データを割り振る

読み込んだデータを、学習用（8 割）と予測用（2 割）に振り分ける。

```
#2
from sklearn.model_selection import train_test_split

(train_X, test_X, train_Y, test_Y) = train_test_split(digits.data,
                                    digits.target, test_size=0.2, random_state=0)
```

	イメージデータ	教師データ
学習用	train_X	train_Y
予測用	test_X	text_Y

■K 近傍法

K 近傍法は、KNeighborsClassifier「クラス」として準備されている。
したがって、インスタンスを作成する必要がある。fit でデータを設定して学習を行わせる。

```
#3
from sklearn.neighbors import KNeighborsClassifier
model =  KNeighborsClassifier()
model.fit(train_X, train_Y)
print(model)
```

◆KNeighborsClassifier で予測する

学習できたら、これを利用して以下のコードのように予測を行ってみます。

```
#4
from sklearn import metrics
from sklearn.metrics import accuracy_score, confusion_matrix,
                                                    classification_report
pred = model.predict(test_X)
score = accuracy_score(test_Y, pred)
print('score:%s' % score)

print(pred[:20])
print(test_Y[:20])
print(classification_report(test_Y, pred))
print(confusion_matrix(test_Y, pred))
```

◆予測の結果

この結果でのスコアは 0.975 でした。

かなり粗いイメージでも、それなりの結果が出せています。

```
score:0.975
[2 8 2 6 6 7 1 9 8 5 2 8 6 6 6 6 1 0 5 8]
[2 8 2 6 6 7 1 9 8 5 2 8 6 6 6 6 1 0 5 8]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        27
           1       0.97      0.97      0.97        35
           2       1.00      0.97      0.99        36
           3       0.91      1.00      0.95        29
           4       1.00      0.97      0.98        30
           5       0.95      0.97      0.96        40
           6       1.00      1.00      1.00        44
           7       0.95      1.00      0.97        39
           8       1.00      0.90      0.95        39
           9       0.98      0.98      0.98        41
```

最初の 20 個を見ても、上段の予測した値と下段の正解の辺りが一致しているのがわかります。

行列データでも、ところどころ間違ったデータがみられるが、全体にわたってほぼ正しい値を導いています。

```
[[27  0  0  0  0  0  0  0  0  0]
 [ 0 34  0  0  0  1  0  0  0  0]
 [ 0  0 35  1  0  0  0  0  0  0]
 [ 0  0  0 29  0  0  0  0  0  0]
 [ 0  0  0  0 29  0  0  1  0  0]
 [ 0  0  0  0  0 39  0  0  0  1]
 [ 0  0  0  0  0  0 44  0  0  0]
 [ 0  0  0  0  0  0  0 39  0  0]
 [ 0  1  0  2  0  0  0  1 35  0]
 [ 0  0  0  0  0  1  0  0  0 40]]
```

■データを画像として表示

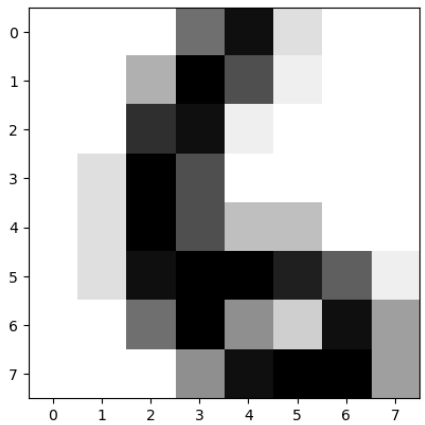
```
print(train_X)
X0=train_X[0]

X0_square = X0.reshape(8,8)
X0_square

X0_square.shape
```

X0=train_X[0]　先頭のデータを取り出す
X0.shape　変形前の shape を確認する
X0_square = X0.reshape(8,8)　8 ピクセル四方に変形する
X0_square.shape　変形後の shape を確認する

◆imshow()メソッドで表示

```
from matplotlib import pyplot
fig, ax = pyplot.subplots()
ax.imshow(X0_square, cmap='binary')
```

UCI データには、0 に白が 16 に黒が割り当てられています。
imshow（）は 2 次元データ配列を受け取り、これを画像として描画します。cmap は、Matplotlib で使われるカラーマップを意味しています。引数に「binary」を指定するとグラデーションになります。



■UCI データセットを学習させる

◆ロジスティック回帰を使った学習

sklearn.linear_model から LogisticRegression クラスをインポートしてロジスティック回帰のモデルを作成する。

分類をおこなうモデルは慣習的に「clf」（classifier、分類機）という変数名を付ける。

```
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression(random_state=0, solver='liblinear', multi_class='auto')
clf
```

手書きを学習させる。

```
clf.fit(train_X,train_Y)
```

scikit-learn で作成したモデルに教師データを学習させるには、fit メソッドを使う。

文法：

clf.fit（X, y）

第 1 引数：教師データの特徴行列

第 2 引数：教師データのラベルデータ

■手書き文字を認識させる

◆データの前処理と特徴抽出

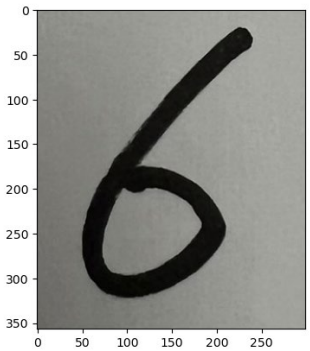
手書き文字をデータ化する前処理には次のような段階がある（Pillow を使う）

・ファイルの切り抜き
・明瞭化
・グレースケール化
・画像の縮小
・明暗の反転
・ndarray に変換
・階調の削減（NumPy を使って）

◆画像の matplotlib での表示

手書きの画像を準備する。(太いものではっきりした方が後処理も楽)
画像は、加工ソフトで文字周辺ギリギリまでトリミングし、名前を付ける。
名前を付けて Python ファイルと同じディレクトリに設定する。

```
from PIL import Image
import matplotlib.pyplot as plt
im = Image.open('moji.jpeg')

fig, ax = plt.subplots()
ax.imshow(im)
plt.show()
```



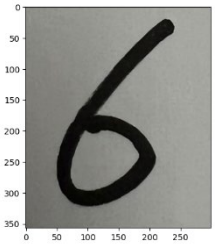
◆ImageEnhance で明暗をつける

ImageEnhnce モジュールにある Brightness オブジェクトを使い、画像の明暗を調整する。引数が大きいほど画像は全体に明るくなる。最適な引数を見つける。

```
from PIL import ImageEnhance
enhancer = ImageEnhance.Brightness(im)
im_enhanced = enhancer.enhance(2.0)
```

変更した画像を表示させるために、次を実行

```
fig, ax = plt.subplots()
ax.imshow(im)
plt.show()
```

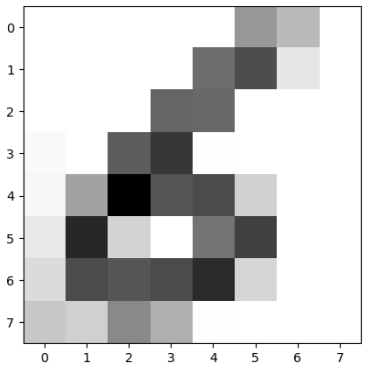


◆グレースケース化

```
im_enhanced = ImageEnhance.Brightness(im).enhance(2.0)
im_gray = im_enhanced.convert(mode='L')
fig, ax = plt.subplots()
ax.imshow(im_gray, cmap='gray')
plt.show()
```

◆文字を 8×8 に

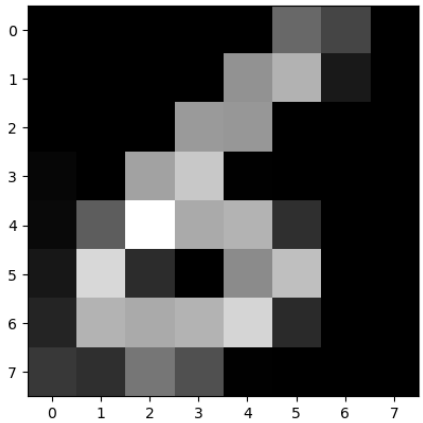
グレースケール化したデータを供しデータと同じ「8×8」ピクセル四方に変換する。変換には Image オブジェクトの resize()メソッドを使う。



```
import matplotlib.pyplot as plt
from PIL import Image, ImageEnhance

im = Image.open('digi.jpeg')
im_enhanced = ImageEnhance.Brightness(im).enhance(2.0)
im_gray = im_enhanced.convert(mode='L')

im_88 = im_gray.resize((8, 8))

fig, ax = plt.subplots()
ax.imshow(im_88, cmap='gray')
plt.show()
```

◆明暗の反転

```
im_enhanced = ImageEnhance.Brightness(im).enhance(2.0)

im_gray = im_enhanced.convert(mode='L')
im_88 = im_gray.resize((8, 8))
im_inverted = ImageOps.invert(im_88)

fig, ax = plt.subplots()
ax.imshow(im_inverted, cmap='gray')
plt.show()
```



■画像データを ndarray に変換

scikit-learn で文字を予測するためには、モデルの predict()メソッドを使う。
predict()メソッドには特徴行列を ndarray の形式で渡す必要がある。この場合、NumPy の asarray()関数を使って Pillow の Image オブジェクトを 2 次元の ndarray データに変換する。

```
import numpy
x_im2d = numpy.asarray(im_inverted)  #変換
x_im2d
```

表示結果：

```
array([[  0,   0,   0,   0,   0,  66,  44,   0],
       [  0,   0,   0,   0,  93, 113,  16,   0],
       [  0,   0,   0,  98,  96,   0,   0,   0],
       [  4,   0, 103, 127,   1,   0,   0,   0],
       [  6,  59, 162, 108, 114,  30,   0,   0],
       [ 15, 137,  28,   0,  88, 121,   0,   0],
       [ 23, 114, 108, 114, 135,  27,   0,   0],
       [ 36,  30,  75,  51,   1,   0,   0,   0]], dtype=uint8)
```

この段階では 0 から 16 のグレースケールに収まっていない。

・この配列を「1 次元」のデータに変換する

```
X_im1d = x_im2d.reshape(-1)
X_im1d
```

※大文字と小文字に注意。im1d は数字の「イチ」

```
array([  0,   0,   0,   0,   0,  66,  44,   0,   0,   0,   0,   0,  93,
       113,  16,   0,   0,   0,   0,  98,  96,   0,   0,   0,   4,   0,
       103, 127,   1,   0,   0,   0,   6,  59, 162, 108, 114,  30,   0,
         0,  15, 137,  28,   0,  88, 121,   0,   0,  23, 114, 108, 114,
       135,  27,   0,   0,  36,  30,  75,  51,   1,   0,   0,   0],
      dtype=uint8)
```

◆数値の範囲を変換する

0 から 255 までの範囲の値を 0 から 16 までの値に変換する

```
X_multiplied = X_im1d * (16/ 255)
X_multiplied
```

255 の数から 16 への変換の式は、他でも応用ができる。

```
array([ 0.       ,  0.       ,  0.       ,  0.       ,  0.       ,
       4.14117647,  2.76078431,  0.       ,  0.       ,  0.       ,
       0.       ,  0.       ,  5.83529412,  7.09019608,  1.00392157,
       0.       ,  0.       ,  0.       ,  0.       ,  6.14901961,
       6.02352941,  0.       ,  0.       ,  0.       ,  0.25098039,
       0.       ,  6.4627451 ,  7.96862745,  0.0627451 ,  0.       ,
       0.       ,  0.       ,  0.37647059,  3.70196078, 10.16470588,
       6.77647059,  7.15294118,  1.88235294,  0.       ,  0.       ,
       0.94117647,  8.59607843,  1.75686275,  0.       ,  5.52156863,
       7.59215686,  0.       ,  0.       ,  1.44313725,  7.15294118,
       6.77647059,  7.15294118,  8.47058824,  1.69411765,  0.       ,
       0.       ,  2.25882353,  1.88235294,  4.70588235,  3.2      ,
       0.0627451 ,  0.       ,  0.       ,  0.       ])
```

■手書き文字を予測

◆モデルに文字を予測させる

```
clf.predict([X_multiplied])[0]
```

以下、改良した全文を掲載する

＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝

```
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
import numpy as np
from PIL import Image, ImageEnhance, ImageOps
import matplotlib.pyplot as plt

# 手書き数字データセットの読み込み
digits = load_digits()

# 訓練データとテストデータの分割
(train_X, test_X, train_Y, test_Y) = train_test_split(digits.data, digits.target,
test_size=0.2, random_state=0)

# LogisticRegression モデルの訓練
clf = LogisticRegression(random_state=0, solver='liblinear', multi_class='auto')
clf.fit(train_X, train_Y)

# 新しい画像 'digi.jpeg' の読み込み
im = Image.open('digi.jpeg')

# 画像の前処理
# 明るさの調整、グレースケール変換、リサイズ、色反転
im_enhanced = ImageEnhance.Brightness(im).enhance(2.0)
im_gray = im_enhanced.convert(mode='L')
im_88 = im_gray.resize((8, 8))
im_inverted = ImageOps.invert(im_88)

# 画像データを配列に変換し、適切な形状に変更
```

```
x_im2d = np.asarray(im_inverted)
X_im1d = x_im2d.reshape(-1)
X_multiplied = X_im1d * (16 / 255)

# 画像データに対する予測
prediction = clf.predict([X_multiplied])[0]
print("Predicted label:", prediction)
```